ISODE CONSORTIUM

ISODE Volume 4

Administrator's Guide

Messaging Handling Services

This documentation and the software are based on contributions from multiple sources. Acknowledgements for contribution are given in *ISODE Volume 1: Overview*.

This manual is associated with Version 2.0 of March 1994, and was last updated in March 1994.

Contents

List of Tables

List of Figures

Preface

This guide is published in support of the ISODE Consortium Release 1 (IC R2). It is intended for administrators who need to set up or manage X.400 Message Handling Services in an OSI environment.

It is assumed that readers of this guide are already aware of the contents of *ISODE Volume 1: Overview* and *ISODE Volume 3: Administrator's Guide: General Services*. Other guides to other IC R2 products and their uses are listed under Related Publications later in this Preface. Note particularly *ISODE Volume 7: User's Guide: Message Handling Services* and *ISODE Volume 14: Programmer's Guide: X.400 MTA*. There are several papers in the directory *doc* which are relevant, in particular a general overview of the PP MHS in [4].

What this Guide Covers

This volume describes the operation of the IC R2 PP MHS software.

Chapter 1 gives contains general introductory material. Software dependencies and related software packages are identified. Overall MTA layout and performance is discussed. Required ISODE tailoring is presented.

Chapter 2 describes how to use the *tailor* file to configure the MTA at runtime. It explores the variables that exist for *tailor* file entries and indicates how to configure the program's logging output.

Chapter 3 describes how the PP routing algorithms work, based on both the *tailor* file entries and the tables discussed in Chapter 4.

Chapter 4 contains detailed information on the function and syntax of tables used in the routing and delivery of messages. Information on tables for lookup and conversion of addresses is interspersed with details of resources governing message authorisation and authentication.

Chapter 5 covers how to configure a new MTA using the tables described in Chapter 4. It explains X.400 connection procedures and how to configure and test X.400 connections. It also includes instructions on how to handle configuration errors.

Chapter 6 describes additional functionality that can be added to the PP MHS module. Sections cover G3Fax capability and an ASN.1 filter.

Chapter 7 covers the tools available for managing and administering the IC R2 PP MHS module. The queue manager (QMGR) is described with its monitor and control utilities (MTA console and Lconsole). Checking procedures include a probe function to check on the prospective routing of messages.

Chapter 8 discusses managing PP on a day-to-day basis.

Appendix A is contains detailed information on how PP can be configured to make use of the X.500 Directory Service or the Domain Name Server.

Appendix B gives an example of the *tailor* file.

Appendices C and D give examples of the scripts used to start PP and update its operation.

Appendix E lists tables required for correct X.400 operation and tells how to retrieve them.

Appendix F introduces the international MHS Coordination Service for the academic and research community, and tells how to get in touch with the coordinators.

Appendix G contains sample log output of connection attempts from the *x400in84* and *x400out84* channels.

Software and Documentation Versions

IC R2 is the second major release of ISODE software made by the ISODE Consortium. It is derived, with revisions and additions, from the openly available ISODE 8.0 release.

The release is accompanied by a comprehensive documentation set comprising the following volumes:

Volume	Title
01	*ISODE Volume 1: Overview*
02	*ISODE Volume 2: Administrator's Guide: Installation*
03	*ISODE Volume 3: Administrator's Guide: General Services*
04	*ISODE Volume 4: Administrator's Guide: Message Handling Services*
05	*ISODE Volume 5: Administrator's Guide: Directory Services*
07	*ISODE Volume 7: User's Guide: Message Handling Services*
08	*ISODE Volume 8: User's Guide: Directory Services*
09	*ISODE Volume 9: Programmer's Guide: General Services*
10	*ISODE Volume 10: Programmer's Guide: ASEs*
12	*ISODE Volume 12: Programmer's Guide: X.500 DUA*
13	*ISODE Volume 13: Programmer's Guide: Layer Interfaces*
14	*ISODE Volume 14: Programmer's Guide: X.400 MTA*
15	*ISODE Volume 15: Programmer's Guide: X.400 UA*
16	*ISODE Volume 16: Programmer's Guide: Transport Services*
17	*ISODE Volume 17: Programmer's Guide: ASN.1 Library*
18	*ISODE Volume 18: Programmer's Guide: ASN.1 Tools*

IC R2 is released as a number of separate modules. Each module has its own update number which is incremented independently from other modules whenever an update is made to it. The version numbers of the various components may therefore differ from one another.

As of March 1994, the latest versions of the modules and their associated documentation are as follows:

Module	Version	Volume	Last Document Update
IC Base	R2.0	01	June 1993
		02	March 1994
		03	March 1994
		09	February 1994
		10	February 1994
		13	February 1994
		16	February 1994
		17	February 1993
		18	February 1993
IC DSA	R2.0	05	March 1994
IC DUA	R2.0	08	February 1993
		12	March 1994
IC MTA	R2.0	04	March 1994
	R2.0	07	February 1994
		14	February 1993
		15	February 1993
IC SNMP	R2.0	03	March 1994
Contributed	R2.0	-	-

The ISODE Consortium

The ISODE Consortium is a not-for-profit cooperative enterprise. Its mission is to promote and evolve the ISODE implementation of OSI. This is a major package of OSI applications, including source code and documentation, with a worldwide record of operational success.

All individuals and organisations handling ISODE source code are encouraged to join the ISODE Consortium. For more information about the ISODE Consortium, contact:

Headquarters

Address:	ISODE Consortium The Dome The Square Richmond TW9 1DT UK	Phone: Fax:	+44 81 332 9091 +44 81 332 9019

Email

Internet (RFC822)ic-info@isode.com

MHS (X.400): S=ic-info; O=ISODE Consortium; P=ISODE; A=MAILNET; C=FI;

If this is not possible, you may also reach the ISODE Consortium at:

US Office

Address:	ISODE Consortium PO Box 200195 Austin TX 78720 USA	Phone: Fax:	+1 (512) 338 3340 +1 (512) 338 3600

Availability

IC R2 is a source release, and is available in the following ways:

1. To organisational members of the ISODE Consortium as a benefit of membership.
2. To educational and non-commercial research organisations, under zero-cost licence from the ISODE Consortium or under licence from a Research Consortium member of the ISODE Consortium.
3. To any organisation as a 30 day evaluation copy from the ISODE Consortium.

Acquisition and use of IC R2 and related materials for any purpose requires a written licence agreement from ISODE Consortium Limited, or a written licence from an organisation licenced by ISODE Consortium Limited to grant such a licence.

Binary products based on IC R2 are available on a wide range of platforms from ISODE Consortium members. Information on these products and where to obtain them is available from the ISODE Consortium.

Bug Reporting

Bug reports on software releases made by the ISODE Consortium are welcomed. These may be sent to the Consortium by any means, but electronic mail to one of the bug reporting addresses is preferred. Please send proposed fixes with the reports if possible. For non-members of the ISODE Consortium, any reports will be acknowledged, but no further action is guaranteed. Any changes resulting from bug reports may be included in future releases.

The bug reporting addresses in Internet (RFC 822) format are:

- `bug-isode@isode.com` for general ISODE reports.
- `bug-quipu@isode.com` for reports on QUIPU (Directory).
- `bug-pp@isode.com` for reports on PP (MHS).
- `bug-snmp@isode.com` for reports on SNMP.
- `bug-doc@isode.com` for reports on the documentation.

The MHS (X.400) equivalents of these addresses are:

- `S=bug-isode; O=ISODE Consortium; P=ISODE; A=MAILNET; C=FI`
- `S=bug-quipu; O=ISODE Consortium; P=ISODE; A=MAILNET; C=FI`
- `S=bug-pp; O=ISODE Consortium; P=ISODE; A=MAILNET; C=FI`
- `S=bug-snmp; O=ISODE Consortium; P=ISODE; A=MAILNET; C=FI`
- `S=bug-doc; O=ISODE Consortium; P=ISODE; A=MAILNET; C=FI`

Support

Basic support of ISODE Consortium releases is available to all consortium members as a benefit of membership. This includes upgrades and access to a database of bug reports and fixes. The ISODE Consortium also offers some additional support to companies building products specifically based on ISODE Consortium releases.

A range of ISODE-related support services are offered by ISODE Consortium members. A list of such services will also be available from the ISODE Consortium.

Chapter 1

Introductory Information

1.1 Overview

Before compiling, installing, and configuring PP, there are some general issues which it is useful to consider. If you are planning to install PP on more than one machine, you should read the latter parts of this chapter carefully.

System issues such as supported hardware and software platforms, disk space requirements, and communications requirements are discussed in *ISODE Volume 2: Administrator's Guide: Installation*.

1.2 Related Software

1.2.1 X Window System

The system has a very useful monitoring tool based on the X Window System[1] which is used for managing the MTA as a whole. It is possible to run an MTA without the console, but not so easily. The console depends on X11 release 4 (or greater) and the Athena Widget set. It will not work under earlier releases of X11.

There are also present some small programs to do X-based alert of new mail. These programs have the same X requirements.

All binaries should work against X11 release 3 (or greater) servers.

1.2.2 QUIPU

This release has no requirements on the QUIPU X.500 directory service agent. There is a list channel which can be configured to utilise the directory. The X based mail alert utility has the ability to fetch

[1] The X Window System is a trademark of the Massachusetts Institute of Technology

an originator's photograph from the directory (this feature can be disabled).

Future releases of PP will require a QUIPU DSA.

1.3 Specific Channel Requirements

Some of the channels in PP have external dependencies.

SMTP: The Simple Mail Transfer Protocol channel requires the use of TCP/IP. If the domain name server is to be used, then the BIND[1] resolver library must be provided at installation.

JNT Mail: This is integrated with the UNIX-NIFTP package, available from Cambridge (UK) at the following address:

```
Piete Brooks (pb@cl.cam.ac.uk)
Cambridge University Computer Laboratory,
Cambridge
ENGLAND
```

X.25 access is usually required for this channel.

X.400: This uses the ISODE lower layers (both 1984 and 1988 variants). For most services X.25, CONS, or CLNP is required to run this channel, although it can be run over TCP/IP using RFC-1006[8].

UUCP: This channel requires a working UUCP installed on the given machine.

DECNET: This requires the Sunlink DNI (version 7.0 or later) package.

1.4 User Agents

This version of PP does not supply any sophisticated user agents. It provides a mail-sending interface of the same approximate functionality of */bin/mail*. It is strongly recommended that the MH[6] user agent be considered for use with PP. MH should be configured with the SMTP transfer option which will connect to PP through the SMTP protocol (configuration option `mts send/smtp`). This also allows PP to be accessed from hosts not running PP directly but with SMTP connectivity.

Warning: MH should be compiled without the `BERK` option.

Commercially available User Agents can be configured to interact with PP using either the 822-local or P3file channels.

1.5 Compatibility with Older MTAs

PP is designed as a replacement for *sendmail*(8) and MMDF. As such it is capable of the majority of functions of both (and a great deal more). However, there are some differences.

1.5.1 MMDF

PP does not support the MMDF syntax that allows users to give an address as `user=string@host.domain`.

1.5.2 Sendmail

PP does not deliver to sendmail style mailboxes by default. This can be changed however (see Section 4.11).

Some programs which send mail have compiled into them the *sendmail* program. To deal with these, a fake sendmail is provided that emulates most of the common sendmail options.

As PP does not have sendmail's general rewriting rules, it can not in general be programmed to "fix up" strange addresses. This applies particularly to address formats such as DECnet and Xerox-style addresses. To handle this sort of address an inbound channel should be written which can parse these addresses and convert them to RFC-822 addresses.

1.6 MTA Layout

Usually PP will support sites with many hosts. It is possible, but not desirable, to run a copy of PP on each host. An MTA is defined by the existence of a queue (spool area). Each queue should be managed (in human terms) in order to guarantee good service.

For this reason, the number of MTAs should be kept small, and other mechanisms used for delivery to local hosts. Possibilities include:

- NFS. This simply consists of NFS-mounting (possible through use of an automounter) the necessary files systems for delivery.
- Remote access protocols. Use of services such as POP[7] may be an alternative.
- Delivery to a central place which is NFS-mounted by clients.

For these reasons, some of the PP utility programs may be accessed remotely using *rsh*(1).

A site with 1000 users and 100 hosts might consider installing between one and four MTAs. Reasons for choosing MTAs might include:

- External communication links.
- Large host with many users.
- Fileservers used substantially for mail.

- Separate administration domains.

1.7 Performance

PP is running on a Sun 4/330 at the nsfnet-relay.ac.uk mail hub, moving around 15,000 messages a day between the UK and the US. Experience from this and other sites has been fed back into the development process, with the result that the latest version of PP performs better than its predecessors. However, as is the case with any application, PP could still benefit from more fine tuning.

1.8 ISODE Runtime Tailoring

1.8.1 Queue Manager Tailoring

Information relating to the MTA, its QMGR, and their addresses are normally obtained from the file *isoentities* and take the general form:

```
vs2 "pp qmgr" 1.17.6.2.1 #1001/Internet=vs2+18000
```

The first field is the designator of the machine that the QMGR is running on. It is usually the name of the machine. The second is the QMGR service; this should not be changed. The value listed for the third field should also not be changed. The last field specifies the transport selector that the QMGR listens on (1001 in this case) and the NSAP address. In this case it is in internet form, with hostname `vs2` and TCP port number `18000`. The QMGR uses this address to listen on, *submit* uses the address to contact the QMGR as do mta console programs such as *MTAconsole*.

This information may also be obtained from the directory if the directory nameserver is being used. A suitable entry for addition to quipu might be:

```
objectClass= top & applicationEntity & quipuObject
cn= pp qmgr
description= Submit to Qmgr
presentationAddress= \
   '1001'H/LOCAL-ETHER=lancaster.nexor.co.uk+18000
supportedApplicationContext= 0.9.2342.60200172.201.1
acl=
```

The presentation address should be amended to be correct for your site, the `supportedApplicationContext` must not be altered however.

1.8.2 X.400 Tailoring

If the X.400 is going to be used, then a definition of this must be placed in the file */etc/isoservices*. A suitable definition might look like this:

```
"tsap/p1"  "591"   /usr/lib/pp/chans/x400in84
```

Whilst for the X.400(88) channel it might look like

```
"tsap/p1"  "592"   /usr/lib/pp/chans/x400in88 -n
```

This shows the service name `tsap/p1`, the transport selector `"591"` and indicates the location of the program (and any arguments). You should use a numeric IA5 form of TSEL for X.400(84) compliance.

Alternatively, an entry can be added to the directory if the *iaed*(8) is being used. Such an entry might look like the following:

```
objectClass= top & applicationEntity &\
      quipuObject & iSODEApplicationEntity
cn= x40084
description= X.400 service (1984)
presentationAddress= "591"/LOCAL-ETHER=128.243.9.1|\
  Janet=00002100102999|\
  IXI=20433450210399
supportedApplicationContext= 2.6.0.1.6
execVector= /usr/lib//pp/cmds/chans/x400in84
```

For more details of the formats of these files, see *The ISO Development Environment: Users Manual*.

Chapter 2

Configuring PP Channels

This chapter is principally concerned with the *tailor* file and how it is configured. The *tailor* file contains the runtime MTA configuration information. This chapter describes how to construct this information.

The information in the *tailor* file may duplicate that in the *Make.defs* in some cases. In these cases, the *tailor* file settings will override the *Make.defs* settings.

There are four basic types of entries in the *tailor* file:

- Variable entries.
- Logging entries.
- Table entries.
- Channel entries.

The following sections describe how to construct these entries. Appendix B contains a complete example *tailor* file.

Since the construction of the information is complex, it is *strongly* recommended to use *ckconfig* (8). *ckconfig* applies several sanity checks to the information contained in the *tailor* file and generally ensures that the information represents a logically correct configuration. *ckconfig* is described in Section 7.5.2.

2.1 The General Syntax of Tailor File Entries

A comment starts with an unescaped hash (`#`) and finishes at the end of the line. Other lines are split into components separated by white space or commas (or both). A line may be extended onto the next line by starting that next line with some white space. A field may be enclosed by quotation marks (`"<field>"` or `'<field>'`); this disables special characters such as field delimiters.

2.2 Variable Tailor File Entries

Variable entries take the form of a keyword and its value, separated by white space. Each key/value pair should be on a separate line and start in the first column. For example:

```
cmddir   /usr/pp/cmds
tbldir   /usr/pp/tables
```

There are a finite number of tailoring variables that can be referenced in the *tailor* file. These tailoring variables are divided into two types, mandatory tailoring variables and optional tailoring variables. They are listed and described in the sections below.

2.2.1 Mandatory Tailoring Variables

The following tailoring variables *must* be included in the *tailor* file. They are listed in alphabetical order for ease of reference:

`bodypart`: This is a list of bodyparts that are recognised by the system. This line may occur several times in the tailor file, each time appending its arguments to the list of body parts.

`headertype`: This is a list of headers that are recognised by the system. This line like the above line may occur several times each time appending to the list. Note that the common `hdr.` prefix is omitted from these values.

`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: `sheriff.cs.nott.ac.uk`. This is not recognised as a local address by default.

`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: `cs.nott.ac.uk`. This variable should usually be present in the domain table and marked as local in the *domain* table to ensure that messages routed to this name are regarded as suitable for local delivery.

`loc_or`: This is the local O/R 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, that must be marked in the *or* table. For example:

```
"/OU=CS/O=NOTT/PRMD=UK.AC/ADMD= /C=GB/"
```

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

`postmaster`: This is the RFC 822 address of the local mail system administrator. Typically of the form: `Postmaster <postmaster@cs.nott.ac.uk>`.

`pptsapd_addr`: This is the address that the *pptsapd* listens on for incoming connections. It is also used by the *qmgr* to call up the *pptsapd* to start a channel. The default value is: `Localhost=20001`.

This indicates that processes can communicate with the *pptsapd* daemon via a TCP/IP connection to the local machine on Port 20001. The address could just as easily be a remote machine name specifying an X25 address or a real NSAP (e.g., `Janet=0000210010297` for

X.25 and NSAP=388261100123456789 for NSAP). However, only Network services where the Transport service is implemented in user space by the ISODE TSAP library are suitable [1].

`qmgrhost`: This is the name of the machine on which the QMGR is running. For example, you could give the output of *hostname* (1) here. It is used as an index either into the ISODE *isoentities* file, or to search the X.500 directory for the application context. It is used by *submit* to contact the qmgr. *Submit* caches the result of the lookup in the file *qmgr-pa.cache* in the queue directory to speed up the resolution process. If this variable is changed or the details in the *isoentities* file or the X.500 directory are altered, this file should be removed.

2.2.2 Optional Tailoring Variables

The tailoring variables in this section are less critical, having defaults that are normally suitable. They are also listed in alphabetical order.

`administration_assigned_alternate_recipient`: This variable allows a undeliverable local address to be redirected to the user given here, provided the originator has allowed this. 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 *in place* of the normal message report indicating delivery failure.

`chandir`: This is the directory where the executable channel programs are found. It defaults to a directory called *chans* within the *cmddir*. When defined, if it is a relative name, (i.e., not starting with a forward slash) then it will be assumed to be a subdirectory of the *cmddir*.

`cmddir`: This is a directory and is the default place to find all the main PP commands. It should be a fully qualified pathname, i.e., starting with a forward slash.

`dap_passwd`: A password to be used in conjunction with the `dap_user` variable in binding to the directory. It is not necessary to set this variable normally, unless the directory entries require an authenticated bind.

`dap_user`: This is an X.500 directory distinguished name used when PP binds to the directory. This should be used to distinguish PP use of X.500 from other users. There is no default for this—if this variable is empty, all directory binds will be anonymous.

`dbm`: This is the name of PP database. It is assumed to be relative to the *tbldir* directory unless a full pathname is given. The default is *ppdbm*.

`default_lookup_policy`: This value is used when configuring PP with X.400 (see appendix A). This specifies the default lookup policy to use. It can take any combination of the following

[1]This refers to the types of provision of Transport Service by the ISODE TSAP library. Basically, there are two types of service: (a) those provided by in-kernel Network services (or below) such as RFC1006/TCP/IP or X.25; and (b)those provided by in-kernel Transport services, such as SunNet OSI or via the TLI or XTI. Class (a) services use a user-space implementation of TP0, and only addresses of this kind are suitable for use as *pptsapd_addr*

values:

Value	Effect
table[=prefix]	Use tables for lookup.
ds	Use X.500 for lookup.
queue	Queue the message on temporary nameserver failure.
nssoft	Continue with the lookup policies even if the previous nameserver lookup fails.

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.

default_lookup_timeout: The default time in seconds to spend on non table lookups. This is taken as a hint rather than as an exact time.

diskuse: This parameter is used to limit the disk space consumed by PP. There are two numbers, the first being the number of blocks that should be free, the second being the percentage of the disk space that should be free. Either of these can be effectively disabled by setting them to 0. The disk that is checked is the partition holding the queue (quedir).

dsaptailor: This indicates a different dsaptailor file to use in place of the default.

formdir: This is the directory where the simple programs run by the *fcontrol* are found. Not usually tailored, it defaults to a directory called *format* within the *cmddir*. When it is defined, it is taken as relative to the *cmddir* unless a full pathname is given.

fsync: This is a boolean variable which controls the use of the *fsync* (2) call. Fsync is normally used when critical files are written to before passing back a handshake. However, it can be switched off by a value of no if you consider that it is too expensive.

isode: This variable allows the tailoring of any ISODE parameter specified in the *isotailor* (5) file. However, the ISODE logging levels are better handled by the isodelogs tailor line.

loc_mtadnname: This is the distinguished name of the mta for use with X.500 routing.

lockdir: If the "file" style of locking is to be used, this is the directory to create the files in. It defaults to */tmp* and should be cleaned out at reboot time. It is ignored otherwise.

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:

Value	Meaning
flock	Use the *flock* (2) system call.
fcntl	Use the *fcntl* (2) system call.
file	Use lock files that are created and removed.
lockf	Use the *lockf* (3) library call.

The default style 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.

`logdir`: This is the directory where log files will be written in by default.

`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.

`maxloops`: This is a number indicating the number of times a message may pass thorough this mta before being rejected. The default is 5.

`nwarnings`: This is the maximum number of warnings to send.

The default is two warnings.

`pplogin`: This is the user id that most of the PP programs will run under. It defaults to *pp* and should be a username found in the */etc/passwd* file.

`qadaddr`: This is the address that the qmgr assistance daemon listens on.

`quedir`: This is the base directory for the PP queue. It should be a fully qualified pathname.

`queuestruct`: This is a variable 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 (say > 50,000.)

`returntime`: This is the time in hours after which to expire an undelivered message. By default, this time is doubled for low priority messages and halved for high priority messages. However, these other values can be overridden by specifying two other times in hours after the normal return time, the first of these being the return time for high priority messages and the second for low priority messages.

The default normal time out is 72 hours, giving about a day for high priority messages and just less than a week for low priority messages. Messages that pass through the list channel are set to low priority.

If you are expecting problems in delivery (e.g., around Christmas) then it may be worth increasing this to some larger value temporarily.

The CCITT F.400 recommendations for these values are: high priority — 4 hours; normal priority — 24 hours; low priority — 36 hours.

`set`: This allows setting of PP's own environment variables. The syntax is `key"="value`.

`setenv`: This allows setting of UNIX[2] environment variables. The syntax is `key"="value`.

`submit_addr`: This is a list of comma-separated submit addresses. By default a user interface program or channel will directly invoke submit, but with this option it will call a submit daemon. The specification is

```
<hostname> [ ":"<port> ]
```

[2] UNIX is a trademark of AT&T Bell Laboratories.

The hostname is a TCP hostname, the port reference is either a number or a port reference found in the services database. In this mode, *submit* is usually run under *inetd*(8). See Section 2.7.1 on page 23 for more details. The port defaults to `4001` if not given, for no very good reason. If it fails to contact *submit* it will attempt to do a fork/exec of it as usual.

Note that if there is more than one address specified, the list must be contained within quotes, i.e

```
submit_addr ``host1:port1,host2:port2''
```

`tbldir`: This is the directory where all the table files are located and should be a fully qualified pathname; it is used by `dbmbuild` to generate the hashed database and for address lookups when tables are defined in `linear` mode.

`unsetenv`: This allows unsetting of UNIX environment variables. The syntax is `key`.

`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.

`wrndfldir`: This is a directory to hold warning message files which can be sent automatically to sender or recipients when authorisation fails or a message is delayed in delivery. It may be a fully qualified pathname starting with `"/"`, or else is taken relative to the table directory *tbldir*.

The default is `warnings`.

`x400mta`: This is the name of this MTA in the X.400 world. It defaults to the same as `loc_dom_mta`. It may be tailored differently if your MTA goes by a different name in the X.400 world. It is only used for information, no use is made for any routing purposes. It should therefore be a string that users will understand.

2.3 Logging Tailor File Entries.

This section describes how to tailor the logging produced by PP.

There are four main groups of logging; covering normal events, unusual operational events, authorisation, and ISODE. By default, each group will describe in a single file, the appropriate activities of all the programs that provide logs. However, the *tailor* file may be used to separate out logging on a per-program or per-channel basis (see Section 2.3.1 for more details). Thus, for example, all normal activities may be logged in one file except *submit* which may be logged in a different file, and at a different level. Messages may also be logged via the *syslog* (8).

All logging entries are comprised of a standard set of key/value pairs, described below.

`file=xxx`: Set the log file to `xxx`. This is either a fully qualified filename, or more usually just a relative pathname which is taken relative to the *logdir*. It defaults to one of `norm`, `stat` or `oper` depending on the log stream.

`level=value`: This adds the given value to the PP logging levels. Legal values are

`all`: Enable all logging (this fills disks very quickly!).

`fatal`: Enable reporting of fatal errors.

`exceptions`: Report exceptional happenings.

`notice`: Report interesting happenings.

`trace`: Trace the programs flow (very detailed).

`debug`: A full debugging trace.

`pdus`: Show protocol data units.

The last three levels are only available if PP was compiled with `PP_DEBUG` defined to a suitable level. The value `all` is special, in that it sets all values. The other levels only set the individual values (e.g., `debug` does not imply settings of any other values).

The default level is to enable `notice`, `exceptions` and `fatal`.

`slevel=value`: Same as `level=value`.

`dlevel=value`: This unsets a PP logging level. Legal values of this are the same as the above.

`sslevel=value`: This sets the levels for which messages will be sent to *syslog* (8).

The default level is to enable `fatal` for each stream (*oper*, *stat*, and *norm*).

`dslevel=value`: This unsets a logging level for `syslog` logging.

`sflags=value`: This sets the logging flags to a given value; the value being one of:

`close`: Close the log file after each entry is made.

`create`: Create the log file if it doesn't already exist.

`zero`: Zero the log file when it gets too big.

`tty`: Log to the standard error as well as to the file. This should be used only when debugging.

The default is to `create` new files.

`dflags`: This unsets one of the above levels for the log.

`size`: Set the maximum size in Kilobytes to which the log should be allowed to grow. The default is not to limit the log size. What happens when the logfile reaches the specified maximum is governed by the flags. If `zero` is not in force, logging will just stop. If `zero` is in force, then the log will be truncated and restarted.

There are currently four logging variables that can be specified although there are several logging structures.

The logging variables are:

`normlog`: This is the log where the bulk of the logging is done.

`operlog`: This is where urgent messages are logged. In the normal course of events no entries should be made in this log unless something serious has happened.

`authlog`: This indicates where authorisation tracing will be logged.

`isodelog`: This line allows the tailoring of specific ISODE tracing levels. The first argument is an ISODE logging level. The rest of the line is interpreted as a normal log tailoring line. Valid ISODE logging levels are:

`x25level`: X.25 level (if used).

- `compatlevel`: Compatibility level.
- `addrlevel`: Addressing level.
- `tsaplevel`: Transport level.
- `ssaplevel`: Session level.
- `psaplevel`: Presentation structures.
- `psap2level`: Presentation level.
- `acsaplevel`: Association level.
- `rtsaplevel`: Reliable transfer level.
- `rosaplevel`: Remote operations level.

An example of logging tailoring is shown in Figure 2.1.

```
authlog         level=notice,   size=400
operlog         level=all,      size=400
normlog         level=notice,   size=400

submit          normlog file=submit      level=notice, size=400
qmgr            normlog file=qmgr        level=notice, size=400
x400in84        normlog file=x400in84    level=notice, size=400
x400out84       normlog file=x400out84   level=pdus,   size=400

isodelog compatlevel    level=pdus
isodelog tsaplevel      level=pdus
isodelog ssaplevel      level=pdus
isodelog rtsaplevel     level=pdus
```

Figure 2.1: Example of Log Tailoring

2.3.1 Diverting Logging Streams

By default, all the ISODE logging is directed at the normal log.

Notice that for a given program or channel, the logging information can be diverted. If a logging entry starts with the name of the program (e.g., `argv[0]` normally) then that entry is specific to that program. For example:

```
submit normlog file=submit
```

makes a *submit*-specific `normlog` tailoring entry. Hence all logging from *submit* directed at the `normlog` will go to the file *submit* instead of the file associated with `normlog`. The net result is that all the normal logging messages from *submit* end up in the file *submit* in the log directory.

This mechanism, that of being able to precede a tailor file line with the name of the program to which it should apply, is actually a general one—any tailor file line may be preceeded by a program or channel name (provided that this name is not a valid tailor file key) and the resulting line will apply only to that program. While used most commonly to divert log streams (as above), it may also be useful in other cases.

For example, to enable X.25 reverse charges on a particular channel:

```
x400out84    isode    reverse_charge    1
```

It is not always clear which of the channel name or the executable program filename should be used.

2.3.2 Logging to Syslog

If your system supports the *syslog* (8) facility, then PP can output desired log messages directly to the system syslog. This is particularly useful in an operational environment. The *oper* log stream contains important messages and is a good candidate to log to the syslog. For example, the *tailor* line

```
operlog    syslog=all
```

will cause all messages that are sent to the *oper* stream to also be sent to syslog. They will be logged with a facility code of `user` [3]. The following mapping is made between ISODE logging levels and *syslog* levels:

ISODE	Syslog
FATAL	ERR
EXCEPTIONS	WARNING
NOTICE	INFO
PDUS, TRACE, DEBUG	DEBUG
others	NOTICE

2.4 Table Tailor File Entries

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

[3] These *syslog* entries should be logged with a facility code of `main`, but they are not!

`override`: This allows values in the table to be overridden, or indeed if the table is small the whole table to be specified in the tailor file. Each occurence 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 in Figure 2.2.

tbl	aliases	**show**=`"Aliases:  mapping -> local id"`, **flags**=dbm,**file**=local−aliases
tbl	**domain**	**show**=`"Mapping domain key ->full domain/MTA"`, lags=dbm,override=`"nexor.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**=`"Qmgr authorisation"`,**flags**=linear

Figure 2.2: Example of Table Tailoring

2.5 Channel Tailor File Entries

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. Finally there are a few miscellaneous channels that do other jobs such as message deletion.

The *channel tailor* file entries each consist of the keyword `chan` followed by a value which, by default, names the channel and the program associated with it; then comes a list of key/value pairs which provides 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`.

The key/value pairs consist of the following:

`name=value`: The name of the channel (overrides the default).

`prog=value`: The program associated with this channel, i.e., the actual binary to run (overrides the default).

`key=values`: 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 2.6 on channel pairing on page 22. 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 entier value should enclosed in quotes.

`show="value"`: A descriptive string used in pretty printing. If this string starts with the keywords `with` or `via` then this value is included in `Received` lines generated for RFC 822 messages.

`type=value`: This parameter is *mandatory* and indicates the type of channel this is. This *must* be set to one of the values shown below:

Value	Type of Channel
`in`	An incoming channel.
`out`	An outgoing channel.
`both`	Both incoming and outgoing.
`flat`	A flattener channel.
`unflat`	An unflattener channel.
`shaper`	A reformatter channel.
`warn`	A warning channel.
`delete`	A message deletion channel.
`qmgrload`	A queue manager loading channel.
`debris`	A debris collection channel.
`timeout`	A timeout channel.
`delay`	A delayed submission channel.

`drchan=value`: This value must be set for outbound channels that cannot handle real delivery reports - that is channels of type `out` or type `both`.. At present, it is only the X.400 channels and the X.400-based delivery channels that can handle delivery reports. Other channels should set this to a suitable channel that can transform DRs into suitable messages. Currently this should always *dr2rfc*.

`content-out=list_of_values`: Content out tailoring only applies to specific channel types.

If the channel is an outbound channel, `content-out` lists the content types the channel can transfer.

If the channel is a flattener channel, `content-out` must have one and only one element and this element is the content type the channel produces.

No other types of channels should have `content-out` set.

The content types PP currently knows about are

`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).

`none`: no specified content. i.e., uses unstructured or exploded form rather than a single content.

The default `content-out` is `none`.

`content-in=value`: If the channel is an unflattener channel, `content-in` lists the content types the channel can explode into the constituent bodyparts.

`cost=value`: The cost of running this channel represented as a non-negative integer. This is used in the reformatting calculations to decide the best conversion path. The lowest cost route will be chosen, all other things being equal. Costs are relative; a cost of 0, the default, means that this channel essentially makes no changes to the message. Higher costs make the channel less favourable.

As a rule of thumb, any channel that makes a real change to a message should have a positive cost. Thus *rfc822norm* channels should have a small cost (e.g., 1) as they make small changes to the message. A channel such as *P2toRFC* should have a larger cost (e.g., 10) as this conversion should not be done unless needed. A channel that removes a content type by *removebp* should be given a large cost (e.g., 100) as this should only be run if there is no alternative. However, a channel such as *p2flatten* should have a zero cost as it makes no change to the message content, just combining it into one file.

`sort=value`: This value is a series of keys that are used for sorting purposes by the queue manager. The first key is the primary sort key and must be either `mta`, `firstmta`, `user` or `none`. The value `none` should only be used as a primary sort key for delivery channels. The remaining keys are secondary sort keys and are taken in order. The values for this field are any of those shown below:

Value	Meaning
`mta`	Sort by destination MTA. This is the most common primary sort key.
`firstmta`	Sort by MTA of first recipient. This is useful for reformatting channels.
`user`	Sort by username. This is useful for local deliveries.
`priority`	Sort by the priority of the message.
`time`	Sort by queued time.
`size`	Sort by the size of the message.
`none`	Don't attempt to sort

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.

`outinfo=value`/`ininfo=value`: This contains channel-specific information that can be used for a variety of purposes. For example, the `outinfo` field is used by the simple formatters to give the real program to run. Typically a simple formatter is a shell script or similar, with *fcontrol* indicated as the program. In such cases the *fcontrol* program is then responsible for queue manager interaction and it runs the program in the `outinfo` field as a subprocess.

`inadr=value`/`outadr=value`: The type of addressing that is used by this channel. The value is one of those below:

Value	Meaning
`x400`	X.400 addressing
`822`	rfc822 addressing

`adr=type`: This assigns the given address format to both the `inadr` and `outadr`. It is used for backwards compatibility and as a short cut.

`insubadr=value`/`outsubadr=value`: The type of subaddressing used by a channel, if appropriate. Currently this is only relevant in conjunction with a definition of `inadr=822` or `outadr=822`. The value is one of those below:

Value	Meaning
`real.rfc822`	normal rfc822 style addressing
`real.rfc733`	rfc733 style addressing
`jnt`	JNT style addressing

For inbound RFC 822 based channels, if `insubadr` is set to either `real.rfc733` or `jnt` then *submit* will recognise percentages as characters with routing properties. If `insubadr` is not set to either of these, *submit* views percentages as characters with no special properties.

`subadr=type`: This variable sets both the `insubadr` and `outsubadr` variables for a channel.

`adr-order=value`: The ordering of addresses on this inbound channel. This is one of the following:

Value	Meaning
`usa`	Internet style addressing is insisted upon.
`uk`	JNT style addressing is insisted upon.
`usapref`	Internet style addressing is preferred but both types will be tried.
`ukpref`	JNT style is preferred but both types will be tried.

The default value is `usa`. Normally only UK sites will need to modify this variable. Note that this flag only affects the behaviour of *submit*.

`appcont=list_of_values`: This is a list of application contexts the channel can transfer. (This refers to using X.500 with PP. See Appendix A for more information.)

`inlookup=policies` The lookup policies to use for messages coming in by this channel. It can take any of combination of the values specified below:-

Value	Effect
`table[=prefix]`	Use tables for lookup.
`ds[=prefix]`	Use X.500 for lookup.
`queue`	Queue the message on temporary nameserver failure.
`nssoft`	Continue with the lookup policies even if the previous nameserver lookup fails.

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`.

`outlookup=policies`: The lookup policies to use for messages going out by this channel. The values are the same as those for `inlookup` described above.

`lookup=policies`: This sets the policies for both inbound and outbound messages.

`timeout=value`: The time in seconds to spend on non table lookups.

`bptin=value`: A list of body parts that the channel accepts as input. This must be one of the types defined in the `bodypart` tailoring item defined in Section 2.2.1 on page 7.

If the channel is an inbound channel and a bodypart submitted by that channel has a name which partially matches an element of `bptin`, then that bodypart name is replaced by the match in `bptin` i.e. if `bptin=ia5-asn` is set for an inbound channel and that inbound channel submits a bodypart named `ia5` then that bodypart is renamed `ia5-asn`.

`bptout=value`: A list of body parts that the channel will output. The possible values are the same as for those for bptin.

If the channel is an outbound channel and bptout is not specified then that channel can accept any bodypart for output.

`hdrin=value`/`hdrout=value`: These variables are much the same as the `bptin/bpout` variables but apply to the headers instead.

`outtable=value`: 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.

`intable=value`: A table associated with the inbound part of this channel.

`mtatable=value`: This is a table that is used in binding to the appropriate channel. See Section 2.6 on page 22 on channel pairing for use of this feature.

`mta=value`: 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.

`access=value`: The type of access this channel requires; only applicable to channels of type `in` and `both`. The value is either `mta`, in which case the channel must be run by a trusted user-id, or else `mts` in which case authentication of messages is enabled. The default is `mta`.

`domain-norm=value`: The amount of domain normalisation of MTS addresses on inbound channels. The value is one of those below (the default is `partial`):

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

With `partial` local domains are recognised and skipped, so the first non-local domain will be normalised.

`maxproc=value`: This is the maximum number of instances of the channel the QMGR is allowed to run at any time. A value of `0` indicates that there is no maximum to the number of instances the QMGR can run. The default is `0`. See Section 7.1.6 for some discussion on limiting channels in this way.

`auth-table=value`: This is a table which contains the authentication data for the channel if it is an `mts` submission channel. The format is described in Section 4.10 on page 66.

`conv=value`: This variable specifies what effect the channel has on a message. This is used to check on conversion restrictions; in particular the X.400 and RFC 1327 conversion semantics. It may take one of the following values:

Value	Meaning
`none`	channel does no conversion
`1327`	channel does RFC 1327/1148/987 header mappings
`1148`	channel does RFC 1327/1148/987 header mappings
`conv`	channel does conversion without loss
`loss`	channel does conversion with loss

The default value is `none`. Channels that do mapping of P2 to RFC 822 *headers* and the reverse should use the value `1327` to indicate that conversion can be carried out even in the presence of the conversion-prohibited flag of X.400. This will allow messages with header mappings and no body part conversions to go through an RFC 1327/1148/987 gateway. Channels that change formats without losing information (except RFC 1327/RFC 1148/RFC 987 channels) should be marked as `conv`. Channels which lose data (such as, say, G3Fax to IA5) should be marked as loss (they should also be given a high cost).

`probe=value`: This variable states whether this channel can support X.400 style probe messages. It has values `y` and `n`. The default is `n`. This should be set for X.400 channels. If submit determines the channel cannot support probes it will return a delivery report.

`trace=value`: This variable tells *submit* what sort of trace element to generate. This is only applicable to RFC 822-based protocols, and can be set to either `received` to generate Received lines or `via` for Via lines. The default is `received`.

`solo-proc=value`: If this variable is set to `yes`, *submit* ensures that messages processed through this channel only have one recipient. This may result in a message with multiple recipients being resubmitted several times with one recipient each time. This feature is useful for when the processing must be specific to one recipient i.e. the *hdr2fax* filter (see Section 6.1.3). By default, `solo-proc` is set to `no`.

`bad-sender-policy=value`: This can be set to one of the following keys to control the policy for unroutable or unreplyable sender addresses.

Value	Meaning
`strict`	Generate delivery report (default).
`sloppy`	Artificially route on failure.

It is STRONGLY recommended that the default policy 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 `sloppy` mode is in effect, PP will attempt to route failure messages for unroutable senders either via the inbound MTA or if that fails to `postmaster`.

`check=value`: This variable sets the checking mode of the channel. Normally this should be left as the default, which is strict checking. However, for unusual cases it may be required to relax some of the constraints normally imposed. Setting `check` to `sloppy` 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.

An example of a channel tailoring is shown in Figure 2.3 and will help to make things a little clearer. The descriptions above can be used to determine exactly what each channel does.

chan	822−local	**prog**=local, **show**="Local 822 Delivery channel",**type**=both, **sort**="user time",**adr**=822,**adr**−order=ukpref, hdrout=822, **bptout**=ia5, **content**−**out**=822, **outtable**=local,**access**=mts, **drchan**=dr2rfc
chan	x40084	**prog**=x400out84,**show**="X400 1984 channel", **type**=both, **adr**=x400, **adr**−order=ukpref, **intable**=x400in84, **outtable**=x400out84
chan	qmgr−load	**prog**=qmgr−load,**show**="Loading the QMGR", **type**=qmgrload
chan	p2flatten	**prog**=p2flatten,**show**="Structured body -> P2", **type**=shaper,**content**−**out**=p2
chan	g3faxtotxt	**prog**=fcontrol,**type**=shaper,**bptin**=g3fax, **bptout**=ia5,outinfo=f−fax2txt, **show**="Fax -> Text mapper"

Figure 2.3: Example of Channel Tailor Entry

2.5.1 Other tailoring

There are some other variables that can be set with `set` which can alter the behaviour of some of the programs in PP. Those of general interest are listed below:

`passwd`: If this variable is set it specifies an alternate password file to use when checking user ID's and names. The main password file is always checked first. The file must be owned by root and not be writable by others.

`list_location`: This variable indicates where in the X.500 directory tree, distribution lists are stored.

2.6 Channel Pairing

Many of the channels within PP naturally fall into pairs. For instance, it is usual to have an inbound X.400 channel and a corresponding outbound X.400 channel.

As this is a common occurence, one channel definition in the *tailor* 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 QMGR knows about paired channels, it can make some optimisations. 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 (this feature is planned for PP).

A side effect of channel pairing can help with authorisation. An incoming channel usually has a fixed name. However, if a number of channels are defined in the tailor 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. Authorisation 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 that 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 RHS 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 authorise 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 authorisation done on that basis. Otherwise it is assumed to be `smtp-internet` and potentially different authorisation is applied.

On a smaller site, this same feature can be used to map many channels onto one. For instance, the *greyin* channel selects one of `gb-janet` `gb-pss` or `ipss` depending on which network the call originated. If this distinction is not important, all three can be mapped to one channel by tailoring:

```
chan gb-janet key="gb-pss,ipss",...
```

2.7 Tailoring of Common Channels

Channels perform operations on messages. Intermediate channels for reformatting and other manipulation are considered later. The following terms apply to endpoint channels:

outbound: A channel which delivers messages out of PP.

inbound: A channel which passes messages into PP (through *submit*).

active: A channel which is initiated by the QMGR.

passive: A channel which is not initiated by the QMGR (typically by a user or server).

The most common types of channel are passive inbound and active outbound.

2.7.1 Submit

The *submit* program can run in a variety of modes. The original and default mode is for incoming channels and user agents to directly execute submit and communicate using *pipe*(2) interprocess communication.

As an alternative, if the `submit_addr` variable is used, submit can be run either from *inetd* or as a standalone daemon. For example the following *tailor* file entry

```
submit_addr lancaster.nexor.co.uk:pp-submit
```

might be combined with the following *inetd.conf* entry

```
pp-submit stream tcp nowait pp
    /usr/lib/pp/cmds/submit submit
```

Alternatively, submit with the `-s` switch will run in standalone server mode. A similar tailor line to above is required. In this mode, submit will read the tailor file and contact the *qmgr* before answering connections, thus being more efficient in some cases.

2.7.2 Protocol Channels

Protocol channels are channels that talk to the outside world. There are a number of these supplied with PP. Those that have special requirements or are optional are described here.

SMTP channel

The *smtp* channel comes in two parts, the outbound channel and the inbound daemon. The inbound daemon can have a number of arguments to it, and can be run either under the control of *inetd* (8) or under the standalone server *smtpd*. To configure it to run under *inetd* the following entry in the file */etc/inetd.conf* might be appropriate (but see *inetd.conf* (5) for more details).

```
smtp stream tcp nowait pp \
    /usr/lib/pp/cmds/chans/smtpsrvr smtpsrvr smtp
```

If, however, you prefer to run a standalone smtp server, the command is something like this (usually included in *pp.start*):

```
/usr/lib/pp/cmds/smtpd /usr/lib/pp/cmds/smtpsrvr smtp
```

In this case, the *smtpd* daemon will run as an independent process, and invoke the *smtpsrvr* when incoming connections are received.

Arguments to the smtp daemon

The following arguments are supported by the *smtpd* program.

`-p port`: Specify a different TCP port to listen on when running standalone.

`-i maxcon`: Set the maximum number of simultaneous smtp connections that are supported. Only obeyed when running standalone.

`-t timeout`: Set a default timeout for use with the nameserver. This is only used if the server is compiled with nameserver support included.

The two mandatory arguments for *smtpd* are the name of a server program to interpret the smtp protocol, and the name of a channel that incoming messages will arrive on.

Arguments to the smtp server

The smtp server can take a number of flags to alter its behaviour. The flags `-r`, `-c`, and `-8` pertain to how incoming messages which contain 8-bit (non-ASCII) characters are handled. According to the SMTP standard, the presence of eight bit data is a protocol violation.

`-r`: (the default) If any eight bit data is received in an incoming message, the message will be rejected due to a protocol violation.

`-c`: Configures the smtpsrvr to accept incoming messages with eight bit data. The incoming data is not modified, and the message is processed normally. (The server ignores the protocol violation).

`-8`: Strip out eight bit data by removing the top bit. The modified message will be processed normally.

`-t secs`: Set the nameserver timeout to the given time.

Finally, the smtp server makes use of one additional tailoring variable in the channel specification. The `ininfo` element of the channel tailoring entry can contain a set of key/value pairs.

`sloppy`: If this key has the value true associated with it then any connection will be allowed. If this is not the case, connections are only supported from hosts that can be reversed translated (e.g., the IP number can be converted back to a host name). For backwards compatibility just a key of `sloppy` is taken to be equivalent to `sloppy=true`. Some people consider that setting `sloppy=false` is a violation of RFC1123 (Internet host requirements).

`amms`: This parameter takes a five or greater digit string. It is used to set the maximum acceptable message size (in bytes) for remote submission. It is used in the SMTP dialog with hosts supporting the SMTP extensions.

As an example, a channel might have something like:

```
ininfo="sloppy=true,amms=100000"
```

Arguments to the smtp outbound channel

The smtp outbound channel has several options. It can be given a flag `-p port` to tell it to connect to a different tcp port other than the default. This can be set up by setting the tailor entry to something like the following:

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

It can be given a `-l` flag which 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.

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

It can be given a `-r` flag in which case the data transfer is a raw transfer of the data. In the absence of this flag, or with a `-8` flag the 8th bit is stripped from all characters.

X.400 inbound channel

The X.400(84) inbound channel (*x400in84*) takes note of the `ininfo` part of the tailoring information. If this is set to the string `sloppy` then no checking of MTA name and password is done for any connection incoming on that channel. Also, if the NSAP cannot be found in the inbound table for this channel, the connection is still allowed.

X.400 inbound also has an optional flag; this is as follows:

`-c channelname`: Claim to be the given channel name on input. You may have a number of X.400 inbound channels, selected initially by T-Selector or network address. If you wish to split up traffic in this way, possibly for authorisation reasons, you should set the channel name in this way in the *isoservices* file. By default the channel is assumed to be the name of the program itself. However, a better way to do this is through the channel pairing system.

In addition the X.400(88) channel allows the flags:

`-n`: Force use of normal mode transfers (the default).

`-o`: Use the old 1984 compatible stack.

UUCP Channel

The uucp channel interfaces to the UUCP system. It uses the `outinfo` string of the channel to tailor the interface. This string is a key value pair that can set various parameters. The key/value pairs are separated by a comma. The possible keys are:

`uux`: Set the `uux` pathname and arguments. This parameter must be present.

`host`: When constructing the uucp from line, use this value as the host name.

An example configuration might be:

```
chan   uucp  prog=uucp-out,key="uucp-out,rmail,uucp-in",
             show="UUCP channel",type=both,
             adr=822,adr-order=U.S.,hdrout=822-us,
             bptout=ia5,outtable=uucp,
             out-info="uux=/usr/bin/uux - -r -gN,host=chx400",
             drchan=dr2rfc
```

The format of the UUCP outbound table is given in Section 4.16.

P3file Channel

The P3file channel implements the X.400(84) P3 protocol. This protocol was designed to allow a User Agent on the same host or another host to receive messages from and submit messages to an MTA (like PP).

In the tailor file, the P3file channel uses the `in-info` string of the channel to tailor the interface. The key set in the in-info string is:

`submit_queue`: Directory in which incoming P3 messages are placed by the UA. This is the directory which will be scanned by the p3filetopp daemon for incoming mail. This parameter must be present.

A sample tailor file entry might be

```
chan    p3file  prog=p3filetopp,show="P3-file channel",
                type=both,intable=p3file,adr=x400,access=mts
                in-info="submit_queue=/usr/spool/pp/sq"
                outtable=p3file,content-out=p2
                hdrout=p2,bptout=ia5
```

The table for the P3file channel must also be defined in the tailor file. A sample entry might be:

```
tbl     p3file  file="ch.p3file" show="EM users and delivery queue"
                flags=dbm
```

There is also a P3file daemon that runs as an independent process. It is called *p3filetopp*. The P3file daemon is used to submit messages into PP. The daemon could be started in the *pp.start* file as:

```
p3filetopp >dev/null 2>&1 &
```

The format of the P3file inbound/outbound table is given in Section 4.17.

The current version of the P3file channel is beta software.

DECnet[4] MAIL11 channel

The *decnet* channel implements the DECnet mail protocol. It is currently only available for the Sun implementation of DECnet (Sunlink DNI, V7.0 onward). The channel comes in two parts, the outbound channel and the inbound server. The inbound server is run under the control of the DECnet spawner (the DECnet equivalent of *inetd*) and will require an entry of the form

```
27    MAIL   /usr/lib/pp/cmds/chans/decnetsrvr
```

in the *dniserver.reg* configuration file. It is not possible to pass any arguments into the server due to limitations of the *dniserver*, so the server takes its tailoring from a channel with the hardwired name of `decnet-in`.

The *decnet* outbound channel makes use of a number of tailoring options. The `outinfo` element of the channel tailoring entry may be set to any combination of the following options. If more than one option is selected then the options are separated by commas and the whole string must be enclosed in quotes.

`strip`: If this option is selected, all headers except for To:, From: and Subject: will be stripped off and thrown away, otherwise they will be moved into the message body. This is for the benefit of VMS users who tend to be panicked by what they see as UNIX style headers.

`mrgate`: Select this option if the MTAs contacted by this channel are running DECs mail router software. This makes a small but significant change to the addresses presented to such sites.

`map_space`: This option enables the mapping of space to underbar in the incoming channel and vice versa in the outgoing channel. This caters for sites running All-in-One and/or mail router which have chosen to have mail addresses with spaces in them. This can be specified in the `ininfo` and `outinfo` variable.

Note that incoming connections are only allowed from hosts for which a reverse mapping exists in the DECnet database. There is no equivalent of the *smtp* `sloppy` option.

2.7.3 Filter Control and Common Filters

Filters can be called via a special channel called *fcontrol*. This channel can be used in a variety of guises to run various filters on messages and, in particular, on the body parts within the messages. It should only be used for filters where there is a one-to-one mapping between incoming body parts and outgoing body parts; i.e., the contents of the body parts, and possibly their names, may be altered by a filter, but the structure of the message will remain the same.

A filter reads an incoming body part from the standard input, does the required filtering, and writes the outgoing body part to the standard output. Any error or logging messages produced by the filter should be written onto the standard error. *fcontrol* logs these messages at the `LLOG_NOTICE` level. The filter should exit with a value of `0` to indicate success. Any other exit value is taken as a temporary failure.

To create a channel which runs a given filter, an entry has to be made in the *tailor* file. The entry should be constructed in the normal way with the following provisos:

`prog`: This should contain the filename of the filter controller, *fcontrol*.

[4] DECnet is a trademark of Digital Equipment Corporation.

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 (x400 format)
`recip`	the address of the recipient of the message (original format)
`recipDN`	the distinguished name the recipient
`822recip`	the address of the recipient (rfc822 format)
`400recip`	the address of the recipient (x400 format)
`qid`	the queue id of the message (e.g., msg.a000000)
`ua-id`	the user agent id of the message
`p1-id`	the P1 id of the message
`outtable`	the outbound table associated with this channel (if any)
`intable`	the inbound table associated with this channel (if any)
`outmta`	the destination mta
`confirm`	if confirmation of deliver is requested, this value is set to "yes" else "no".

Table 2.1: Filter Control Expansion Macros

`outinfo`: The pathname of the filter's executable program should be placed in the `outinfo` field, followed by any arguments that are to be passed to the filter. If the given name is not a fully qualified pathname, the filter controller assumes the executable to be under the *formdir* directory. Arguments of the form `$(key)` will be expanded to the entities described in Table 2.1. The key is case independent.

`bptin`/`bptout` Filters should not change the structure of the message. `bptin` should only contain one type of bodypart. This type is the bodypart type that the filter converts. `bptout` should also only contain one type of bodypart. This bodypart type is the result of the filter's conversion. i.e., the filter converts bodypart `bptin` into bodypart `bptout`.

`hdrin`/`hdrout` This fulfills the same type of function as `bptin/bptout` but for the headers of the messages. The header must be one of those defined in `headertype`.

`type` The type should be set to `shaper`.

Example *tailor* file entries of filters running in conjunction with *fcontrol* are shown in Figure 2.4.

Trivial filters such as voice to ia5 mappings can be done by a simple shell script of the type shown in Figure 2.5.

Here are descriptions of some common filters run in conjunction with *fcontrol*:

RFC 822 Filter

The filter *rfc822norm* is used in conjunction with the general filter controller *fcontrol*, for dealing with RFC 822 and related headers. It takes a generic RFC 822 header as input, and outputs a "normalised"

```
chan 822touk     prog=fcontrol,type=shaper,
                 hdrin=822,hdrout=822−jntmail,
                 outinfo="rfc822norm -adr-order ukpref -jnt -jntsender $(sender) -fold 50"

chan g3faxtotxt  prog=fcontrol, type=shaper, bptin=g3fax,
                 bptout=ia5, outinfo='removebp "Group 3 Fax"',
                 show="Fax -> Text Mapper",cost=100,conv=loss

chan dmd2odif    prog=fcontrol, type=shaper, bptin=dmd,
                 bptout=odif, outinfo=f−dmd2odif,
                 show="Diamond -> Odif",cost=5,conv=conv
```

Figure 2.4: Example of Filter Control Tailoring

```
#! /bin/sh

#
# Voice −> IA5 conversion filter
#

echo "********************"
echo "There was a voice body part here, which was"
echo "removed in the conversion process"
echo "********************"
exit 0
```

Figure 2.5: Simple Shell Script Filter

form. A normalised header is one in which all addresses are fully qualified and are in the correct order. For this reason, the `hdrin` and `hdrout` fields in any associated *tailor* file entry should be some extended version of `822`, e.g `822-us`, `822-uk`, etc.

The filter affects the following fields of the RFC 822 header:

- All of the addressing fields in RFC 822.
- The Acknowledge-To field (JNT Mail).
- Any invalid trace fields are replaced.

The filter's behaviour may be altered by specifying various options. These options are passed through *fcontrol* to the filter by use of the `outinfo` element of the channel structure. One possible `outinfo` string is as follows:

```
rfc822norm -822 -striptrace -jntsender $(822sender) -adr-order ukpref
```

The *rfc822norm* filter can be used in a variety of guises. For example, in appendix B the combination of *fcontrol* and *rfc822norm* appears twice, so providing two different channels, *822touk* and *822tous*.

The filter takes the following options:

`-adr-order <order>`: On input, recognise domain ordering as determined by the given `<order>`. `<order>` can be one of `usa`, `uk`, `usapref`, `ukpref`. These values have the same meaning as their corresponding values used in tailoring channels. The default is `usa`.

`-822`: On output, format source routes according to RFC 822. (default).

`-733`: On output, format source routes with a "%" notation.

`-bigend`: On output, format domains big–endian.

`-littleend`: On ouput, format domains little–endian (default).

`-jnt`: A combination of the `-bigend` option and the `-733` option.

`-striproutes`: On output, remove any source routes (733 or 822) from the header. This removal starts after the first valid domain in each address.

`-stripdomain <domain>`: Like -striproutes, except that only the specified domain is stripped.

`-striptrace`: Remove all trace fields from the header.

`-jntsender <sender>`: This applies the rules of mailgroup note 15, to ensure that the JNT Mail return path correctly identifies the P1 originator. Typically, this will involve adding a Sender Field to the message, and possibly modifying existing trace fields.

`-msgid`: This flag ensures the presence of a message id in the message. If there is no message id, the RFC 822 filter will create one and append it to the header.

`-full`: This flag indicates that all domains in addresses should be normalised. By default, only the next hop of an address is normalised.

`-percent`: This flag indicates that the `%` character should be treated as a routing character when parsing addresses on input. If this witch is not in force, `%` will be treated as any other character.

`-hidelocal <pattern>`: Apply a host-hiding heuristic to "clean-up" mail sent from other hosts on site which is going off site. Pattern can be a simple domain or a domain with wildcard subdomains, for example:

```
  *.icl.stc.co.uk
and
  stl.stc.co.uk
```

but not domains of the form

```
frodo.*.co.uk
```

Warning: this flag should be used with care and should not normally be required.

`-changedomain <from> <to>`: Change occurences of domain `from` to domain `to`. This flag is useful when running PP as a gateway between two or more networks and a specific domain is known by different names on different networks. Note that this replacement may only work on normalised domains. So to normalise all domains and guarantee the replacement, the `-percent` should also be used.

`-fold <number>`: This indicates the length for displaying RFC 822 header fields. The default value is 79; alternately with a value of -1 there will be no folding. Whatever the value, folding will only occur at higher level breaks, and not within an address.

`-stripack`: Strip `Acknowledge-To` lines from the header.

`-acks`: If no `Acknowledge-To` lines are present, generates Acknowledge-To the the sender.

`-exorcise`: Changes domains not found in the table specified by `-valid_domains` to be routed via the domain specified by `-exorcise_domain` (defaulting as appropriate).

`-exorcise_domain domain`: The domain to be added. This defaults to `loc_dom_site`. This also enables the `-exorcise` switch.

`-valid_domains table`: This is a table containing a list of known domain references. It defaults to the *domain* table. This also enables the `-exorcise` switch.

`-external`: When normalising a local address, follow *alias* table entries marked with the qualifier `external` (the default).

`-internal`: Don't follow *alias* table entries marked with `external` when normalising local addresses.

P2norm Filter

The filter *p2norm* can be used to

- normalise O/R addresses in X.400 P2 headers
- and/or downgrade X.400 (88) P2 headers to X.400 (84) P2 headers.

The normalisation of O/R addresses consists of the replacing of synonyms in the *or* table (e.g replacing `/PRMD=X-Tel Ltd/ADMD= /C=GB/` with `/PRMD=NeXor/ADMD= /C=GB/`) and the replacing of local synonyms in the *aliases* table (e.g replacing `/S=pac/` with `/I=P/S=Cowen/`).

The downgrading from X.400 (88) to X.400 (84) is done according to the RFC 1328[3].

The *p2norm* filter's behaviour may be altered by specifying various command line options. The possible options are as follows:

`-nonorm`: don't normalise the O/R addresses in P2 headers. By default, *p2norm* will normalise any addresses in the headers.

`-downgrade`: downgrade X.400 (88) headers to X.400 (84) headers. By default, *p2norm* will not downgrade the headers.

`-internal`: don't follow *aliases* table entries marked with the `external` qualifier.

`-external`: follow *aliases* table entries marked with the `external` qualifier. This is the default behaviour.

Body Part Deleting Filters

In the *formdir* directory there is a general shell script that can be run as a filter to remove body parts. It ignores its input and writes out a short message indicating that the body part has been removed. This can be used as a trivial filter in say, `fax` to `ia5`, to indicate that a fax body part has been deleted. The `outinfo` string for such a channel might be:

```
outinfo='removebp "Group 3 Fax"'
```

An example *tailor* file entry for such a filter is shown in Figure 2.4.

ODIF filters

There exist some filters that will convert between BBN Slate format and ODIF. These filters are not publicly available as they were developed under an ESPRIT PODA project. However, anyone interested in these filters should contact Prof. Peter Kirstein (P.Kirstein@cs.ucl.ac.uk).

2.7.4 The List Channel

This channel should be tailored in a content independent fashion. This means that the following fields should `NOT` be set:

```
content-in
bptin
bptout
hdrin
hdrout
```

And `content-out` should be set to include all content types known to your system including `none`.

Note that it is not possible to mix different forms of addresses on lists. This is because of the different meanings of symbols between the different forms of addressing. For example, with an X.400 address `(a)` is a printable string encoding of the `@` symbol but with an RFC 822 address it is just a comment. This means that lists with X.400 forms of addresses `must` only contain X.400 addresses and must be expanded by a list channel that is tailored with `outadr=x400`. Similarly lists with RFC 822 forms of addresses `must` only contain RFC 822 addresses and must be expanded by a list channel that is tailored with `outadr=822`.

It may be necessary to set the field `adr-order` in the tailoring for a list channel. The default for this field is `usaonly`. If you have some UK ordering of domains in your lists you should set `adr-order=ukpref` or `adr-order=usapref`.

A valid content independent entry for the list channel is as follows:

```
chan list prog=list,show="List channel",type=both,
          table=list,outadr=822,
  content-out="p22,p2,none"
```

The value of the `outinfo` field can alter the behaviour of the list channel. The values that can be set in the `outinfo` field are:

`dosublists`: This enables the expansion of sublists contained within a list. This will improve performance if there are a number of nested lists *BUT* this behaviour may cause problems with the sender field. If a sublist is expanded, it will be submitted with the sender relating to the main list and not relating to the sublist. By default, the expansion of sublists is disabled.

`linked`: (see `notlinked`)

`notlinked`: These two values indicate the mechanism used for resubmitting the body of a message. If the `linked` value is set, the body of the message is resubmitted using the UNIX link facility. This is more efficient in terms of speed and space. If the `notlinked` value is set, the body of the message is resubmitted by copying. The default behaviour is `linked`.

The list channel's basic function is to expand one recipient to many by resubmission. The one recipient is the distribution list, the many are the members of that list. On resubmission, the list channel changes the originator of the message from the original address to the address of the maintainer of the list. The address of the maintainer is constructed using the de facto standard of `listname-request`. As such this address must be a valid address – usually an `alias` entry in the *aliases* table. If no such address is found, `postmaster` is used as the new originator.

2.7.5 The directory list channel

Most of the things that apply to the list channel apply to this channel to. However there are a number of additional parameters that can be set in the `out-info` variable. These are

`dn`**:** The Distinguished name to bind to the directory as.

`passwd`**:** The password to use to bind to the directory.

`localdit`**:** The position in the directory to search below for distribution lists. The default is the value in the *dsaptailor* file.

`search`**:** The search scope. This can be either `subtree` to search the entire subtree from the `localdit` or else `single` for a single level search. The default is `single`.

2.7.6 The RFC 1327 Channels

The *P2toRFC* and *RFCtoP2* channels perform RFC 1327 conversions on the headers of messages. The tailoring of these channels is as with other channels with the proviso that the `conv` entry is set to `1327`. Note also that the `hdrin` field should be set to have `ipn` as well as either `p2` or `p22`. This enables the *P2toRFC* channel to recognise and map IPNs.

Note that *P2toRFC* needs to compare OIDs and for this needs QUIPU's OID tables to be present and correct.

The output of the 1327 channels maybe be altered by use of the `outinfo` tailor entry.

With the *P2toRFC*, if the `outinfo` entry contains the string `uk`, all rfc822 addresses in the produced header will be output in the uk order of domains.

With the *RFCtoP2*, if the `outinfo` entry contains the string `rfc987`, the conversion from rfc822 to P2 will adhere to the RFC 987 mapping rather than the RFC 1327/RFC 1148. Note that these mappings are very similar anyway.

2.7.7 The MIME-MHS Channels

The optional MIME-MHS channels provide conversion between MIME contents and MHS bodyparts according to RFC 1494 and RFC 1495. The simple MHS body part to MIME content conversions are summarized in Table 2.2. Note that we refer to *MHS bodyparts*, *MIME contents*, and *PP body types*—the body types represent the middle ground between MHS bodyparts and MIME contents. To support MIME-MHS conversion, flattener and exploder channels for MIME and filter channels for the RFC 1494-defined conversions need to be specified.

MHS body parts		MIME contents
Basic body parts	*Tag number*	
ia5-text	0	text/plain
g3fax	3	image/g3fax
message	8	message/rfc822
bilateral	14	application/octet-stream
other basic body part	*n*	application/x400-bp; bptype=*n*
Extended body parts	*Data part OID*	
GeneralText with a single ISO-8859 character set	2.6.1.4.11	text/plain; charset=iso-8859-*n*
ODA	2.8.1.0.1	application/oda
mime-postscript-body	1.3.6.1.7.1.2.1.2	application/postscript
mime-jpeg-body	1.3.6.1.7.1.2.1.3	image/jpeg
mime-gif-body	1.3.6.1.7.1.2.1.4	image/gif
other external body part	*oid*	application/x400-bp; bptype=*oid*
mime-body-part	1.3.6.1.7.1.2.1.1	*other MIME type*

Table 2.2: Summary of MIME-MHS Conversions

The MIME-MHS channels require the following body types:

`mime-image-g3fax`, `g3fax`: The MIME content `image/g3fax` is converted without loss to the MHS bodypart `g3fax` by the MIME-MHS filters.

`mime-application-octet-stream`, `bilateral`: The MIME content `application/octet-stream` and the MHS bodypart `bilateral` are "catch-all" formats for arbitrary binary data and are converted directly, without loss.

`mime-unknown`, `external`: The `mime-unknown` body type is used when the tailor variable `pp_fullmime` is set in the *tailor* file (e.g. with the line `set pp_fullmime=true`); it is used to represent any MIME content for which there is no explicitly defined body type. The `mime-unknown` body type is converted to the MHS bodypart `external` according to the conventions described in RFC 1494.

In general, the MIME content `type/subtype` is mapped on the body type `mime-type-subtype` if it exists, and `mime-unknown` otherwise. So, for example, it would be possible to introduce a `mime-application-postscript` body type and an appropriate channel to convert this type to `external`. If the tailor variable `pp_fullmime` is *not* set, then MIME content types that do not have an explicit *PP* body type are converted to `ia5` instead of `mime-unknown`. The ability to fine-tune the channels for MIME-MHS conversion by the specification of new body types can be used for local, non-standard conversions where required.

MIME Shaper Channels

The *mimexplode* channel should be specified like this:

```
chan mimexplode prog=mimexplode,
                show="MIME -> Structured Message",type=unflat,
                content-in=822,hdrin=822,hdrout=822
```

Note that although the MIME exploder doesn't change the message header, it has to read the header to determine the MIME content type of the body. Submit will only invoke the MIME exploder if there is an RFC 822 exploder defined; the `MIME-version` and `Content-type` fields are present in the message header; and the MIME content is something other than `text/plain; charset=us-ascii`.

The exploder will recalculate the formatter path for the message when it has finished in order to specify the required filters for any body types it has generated.

The `mimeflatten` channel is a replacement for the `822flatten` channel that uses MIME formatting instead of RFC 934 formatting. The channel is specified like this:

```
chan mimeflatten prog=mimeflatten,
                 show="Structure Message -> MIME",type=flat,
                 content-out=822,hdrout=822-us,
                 bptin="ia5,mime-image-g3fax,
                           mime-application-octet-stream,
                           mime-unknown",
```

Note that although the `mimemhs` channel only produces the `mime-unknown` body type, we need the other body types to avoid unnecessary conversions for messages that are not destined for X.400 recipients.

Note that all RFC 822 channels (at least `822-local`, `smtp`, and `shell`) will need to accept the various `mime-` body types; the corresponding programs need take no special action. This is because the recipients on the other side of the RFC 822 channels are able to accept the MIME EITs so the MTA does not need to arrange to have these converted to, say, IA5, before the flattener collates the entire message.

MIME Filter Channels

A minimum of five filter channels need to be provided for full MIME-MHS support; three provide conversion for MIME contents to `g3fax`, `bilateral` and `external`, a fourth provides a fall-back for X.400 '84, and the fifth channel converts from any incoming non-IA5 MHS bodypart type to the corresponding MIME content.

The first three channels are specified like this:

```
chan mimetog3fax prog=fcontrol,show="MIME image/g3fax -> X.400/g3fax",
                 type=shaper,conv=conv,cost=5,
                 bptin=mime-image-g3fax,
                 bptout=g3fax,
                 out-info="mimemhs -88"

chan mimetobilat prog=fcontrol,show="MIME app/octet-stream -> X.400/bilateral",
```

```
                    type=shaper,conv=conv,cost=5,
                    bptin=mime-application-octet-stream,
                    bptout=bilateral,
                    out-info="mimemhs -88"

chan mimetoexternal prog=fcontrol,show="MIME -> X.400/external",
                       type=shaper,conv=conv,cost=5,
                       bptin=mime-unknown,
                       bptout=external,
                       out-info="mimemhs -88"
```

The MIME to `external` and `bilateral` filters need not be specified in an exclusively X.400 ’84 environment. The fall back conversion for X.400 ’84 channels is specified as

```
chan mimetomhs84 prog=fcontrol,show="MIME -> X.400 ’84",
                    type=shaper,conv=conv,cost=20,
                    bptin="mime-unknown,mime-application-octet-stream",
                    bptout=ia5,
                    out-info="mimemhs -84"
```

Note that this channel is assigned a higher cost than the other three channels to avoid it being used for X.400 ’88 where a “better” conversion is available. If the `bilateral` body type is used for the X.400 ’84 channels (for example, in a MOTIS ’86 environment), then the `mime-application-octet-stream` body type can be omitted from the input body types for this channel.

The final filter channel provides conversion from all incoming X.400 bodyparts to MIME contents and is specified as

```
chan mhstomime prog=fcontrol,show="X.400 bodyparts -> MIME ",
                  type=shaper,conv=conv,cost=5,
                  bptin="g3fax,external,bilateral,
                         voice,ttx,encrypted,national,videotex",
                  bptout=mime-unknown,
                  outinfo="mhsmime"
```

Summary of MIME-MHS Tailoring

Adding MIME-MHS support to an existing *tailor* is quite straightforward; this section provides a check-list for the changes that need to be made.

- Declare the MHS and MIME body types, for example:

  ```
  bodypart ia5
  bodypart g3fax bilateral external
  bodypart voice videotex encrypted national ttx
  set pp_fullmime=true
  bodypart mime-image-g3fax mime-application-octet-stream mime-unknown
  ```

  An initial implementation should define `pp_fullmime`, it can be removed later if needed (in, for example, an X.400 ’84 dominated environment).

- Define the MIME exploder and flattener channels; make sure that the `822flatten` channel is commented out or removed.

- Add the various `mime-` body types to the set of accepted body types for all RFC 822 channels. For example, the `822-local` channel might look like this:

```
chan 822-local prog=local,
               show="Local Delivery channel",type=both,
               sort="user time",adr=822,adr-order=usapref,
               hdrin=822,hdrout=822-us,
               bptout="ia5,mime-image-g3fax,
                           mime-application-octet-stream,
                           mime-unknown",
               content-out=822,outtable=local,access=mts,drchan=dr2rfc
```

- Ensure that the X.400 channels, the P2 flattener and the P22 flattener can accept the body types that MIME to MHS conversion can generate. In particular, make sure that the relevant channels can accept the `external` body type.
- Add the five MIME-MHS filter channels to the tailor file.

2.7.8 Shaping Channels

Shaping channels alter the structure of the message. This section describes tailoring requirements of the common shaping channels.

RFC 934 Channel

The *rfc934* channel flattens an RFC 822 style message as specified by RFC 934[5]. The tailor file entry for such a channel should be constructed with the following provisos:

`type`: This field should be set to `flat` as the channel flattens message.

`content-out`: This field should contain the value `822` to indicate that it converts contents to RFC 822 format.

Note that the *rfc934* channel only recognises three types of bodyparts: RFC 822 headers, which it recognises as starting with the string `hdr.822`; text bodyparts, which it recognises as being of the form `n.ia5` where `n` is a number; and forwarded messages, which it recognises as being subdirectories. If there is a bodypart in the message that it does not recognise, the *rfc934* channel will fail. The key strings `hdr.822` and `ia5` are hardwired into PP. They can only be changed by editing the relevant entries in the file *Lib/pp/static.c* within the source tree.

If the `outinfo` contains the string `link-hdr`, the *rfc934* channel will link the header across rather than copying.

P2flatten Channel

The *p2flatten* channel combines all the separate components of a message into one file of `p2` or `p22`, depending on how the channel is tailored. The tailor file entry for such a channel should be constructed with the following provisos:

`type`: This field should be set to `flat`, as the channel flattens the message.

`content-out`: This field should contain the content that the channel outputs. It should be either `p2` or `p22`. If it is set to `p2`, *p2flatten* will output a file of `p2` (i.e., a message suitable for transfer via X.400(84)). If it is set to `p22`, *p2flatten* will output a file of `p22` (i.e., a message suitable for transfer via X.400(88)).

P2explode Channel

The *p2explode* channel explodes a `p2` or `p22` message into its separate component bodyparts. The tailor file entry for such a channel should be constructed with the following provisos:

`type`: This field should be set to `unflat` as the channel unflattens the message.

`content-in`: This field should contain the inbound content, either `p2` or `p22`. If both are set the channel will be used for exploding both formats.

p2explode can produce two forms of ia5 text.

- the X.400 encoding of the ia5 bodypart. This is the binary encoding of the ASN and it contains both carriage return and linefeed characters to represent the end of a line. The bodypart will be called `n.ia5-asn`.
- or the UNIX encoding of ia5. This is the plain text file (as carried by RFC-822 based protocols). Its only contains linefeed as its representation of end of lines. The bodypart will be `n.ia5`.

The form of ia5 text required from *p2explode* is tailored by an entry in the `out-info` tailor field. This entry has the form `ia5=value` where value is either `x400` for the X.400 encoding or `822` for the UNIX encoding.

By default, *p2explode* will produce the UNIX encoding.

2.7.9 Miscellaneous Channels

Miscellaneous channels groups together the “special” channels required by PP. This section describes tailoring requirements of these “special” channels.

Qmgrload Channel

The *qmgrload* channel is used to refresh the QMGR model of the queue by reading the model stored on disc. The only proviso for the tailor file entry of this channel is that the `type` is set to `qmgrload`. This channel takes note of the `outinfo` field in the *tailor* file. This, if present, should be a number indicating how many messages to load into the QMGR at one time. It defaults to 50 which is a sensible value for most sites.

Msg-Clean Channel

The *msg-clean* channel is used by the QMGR to free the storage associated with a message once all activity to do with that message has finished. For the tailor file entry of this channel, `type` is set to `delete`.

Debris Channel

The *debris* channel is used by the QMGR to free any storage which no longer is of interest to the mail system. The *tailor* file entry of this channel should have `type` set to `debris`.

The *debris* channel only removes files and directories which it deems suitable for deletion and whose lasted modified times are suitable ancient. The time interval after which a file is suitable ancient may be specified in the `outinfo` field. 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

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 (x400 format)
`recips`	the address of the recipient of the message (original format)
`recipsdn`	the distinguished names of the recipients
`822recips`	the address of the recipient (rfc822 format)
`400recips`	the address of the recipient (x400 format)
`qid`	the queue id of the message (e.g., msg.a000000)
`ua-id`	the user agent id of the message
`p1-id`	the P1 id of the message
`locmta`	the local mta (`loc_dom_mta`)
`locsite`	the local site (`loc_doc_site`)
`hours-left`	the number of hours left until the message will be returned.

Table 2.3: Warning expansion macros

Timeout Channel

The *timeout* channel is used to time messages out once they have been around for a given time. The tailor file entry of this channel should have `type` set to `timeout`.

Delay Channel

The *delay* channel is used for delayed submission. This occurs when *submit* did not have enough time to complete the submission checks for a message. The tailor file entry of this channel should have `type` set to `delay`.

Warning Channel

The *warning* channel is used to send a message back to the originator of a message, warning him or her that the message is delayed and still awaiting delivery. The tailor file entry for this channel should have `type` set to `warn`. If the `outinfo` field has an entry of value `copy-to=address` copies of any warning messages sent will also be sent to the address specified.

The *warning* channel sends back a message found in the *warndir* directory. The file it sends back is of the either `warning.N` where `N` is the number of the warning message (usually either 1 or 2). If this file is not present it will send the file `warning` from that directory.

The text of the file is subject to macro expansion. Strings of the form `$(key)` are expanded with value given in Table 2.3.

Dr2rfc Channel

The *dr2rfc* channel constructs receipt notifications or error messages in the form of RFC 822 messages. Once constructed these messages are then submitted to the QMGR for return to the sender. The channel's behaviour can be altered by various key/value pairs which may appear in the `outinfo` entry of the channel's tailor entry. These pairs are as follows:

`order=value`: If the `value` equals `uk`, *dr2rfc* produces UK-ordered domains in its output. By default, *dr2rfc* produces US-ordered domains.

`submit=value`: This alters the method used to submit any return of contents. `value` can have one of two values - `linked` or `copy`. If the value is `linked`, *dr2rfc* will submit the return of contents via the use of links wherever possible. If the value is `copy`, *dr2rfc* will copy the return of contents on *submit*. The default value is `linked`.

`admininfo=value`: This determines whether verbose administration information is included in positive delivery reports. If the value is `true`, administration information is included with positive delivery reports. By default, the administration information is omitted for positive delivery reports.

`return=value`: This key/value pair tailors the amount of the original message that will be submitted as return of contents. If the value is `all`, the whole of the original message is returned. By default, the return of contents is subject to the constraints described below. Otherwise the value must have the syntax of a number followed by a letter (e.g., 10k or 2000l). This provides an upperbound on certain aspects of the return of contents. The letter must either be an `l` or a `k` indicating which aspect is constrained. If both quantities are tailored, the minimum value of either is taken. `l` constrains the number of lines the return of contents can contain. `k` constrains the size of the return of contents (note that the number before the letter is multiplied by 1000 to get the actual constraint). The default values for these constraints are 5k for the size and 10 lines. Note that return of contents is subject to availability and if available the header of the original message is always returned irrespective of any return of contents constraints.

The `return` key may have multiple occurences in the `outinfo` field, typically to constrain both the number of lines and the size of the return of contents i.e.
`outinfo="return=10k return=4l"`.

Note that *dr2rfc* uses QUIPU's OID tables to map to "user friendly" representations of OIDs.

2.8 How Channels Work

The majority of channels work in the way described below. This description may clarify the view one gets from examining the logging produced by the *qmgr* and the channels.

A channel is called by the QMGR. The QMGR first calls up the *pptsapd* program and tells it what channel is required. The *pptsapd* then fork's and exec's the appropriate channel passing over the communication channel from the Qmgr.

The QMGR then initialises the channel which informs the channel what name it is (at this point the channel usually checks that this is reasonable). Then the QMGR tells the channel to process a given message and a set of recipients. All of the identified recipients will be on the same MTA, or require the same conversion. The channel will then attempt to do the work. After this it will update the queue in the following way for each recipient:

- If a conversion succeeded, it will increment the reformat counter.
- If a delivery succeeded, it will set the status to DONE or DR, dependent on whether a positive Delivery Report is needed.
- If the operation failed permanently, it will set the status to DR.
- If the operation failed for some temporary reason, no change is made to the message.

If the status is set to DR, the DR should be generated before updating the queue status. When the queue is updated, the QMGR should be informed of the final status of each recipient. The returned value is one of the following:

success: In this case the QMGR will schedule the next designated channel to process the message. The current channel will be given other messages to process if they are available.

messageFailure: This is a soft failure associated with the message. The QMGR will try the message again sometime in the future. Other messages on this channel will be tried immediately.

mtaFailure: This is a soft failure associated with the destination mta. No more messages to this destination will be tried, but messages to other destination may be tried on this channel.

messageAndMtaFailure: This is a combination of both the above. Other messages to this destination will not be tried, and in addition, when they are retried this message will not be tried first (e.g., it is suspected that this message may have caused the mta failure!)

positiveDR: A positive delivery report was generated. The QMGR will schedule the appropriate channels to return the report.

successSharedDR: A positive delivery report was generated. A previous recipient has already reorted a positive delivery was generated. Only scheduling will be done once for the previous recipient.

negativeDR: The QMGR treats this exactly the same as `positiveDR`.

failureSharedDR: The QMGR treats this exactly the same as `successSharedDR`.

successChanged: The processing was successful but message information has been changed. In this case the QMGR will read the new information and act on that.

Once the response is received, the QMGR will give the channel more work, or close it down. Connections to MTAs should be kept open, the QMGR will attempt to sort the queue by MTA.

Chapter 3

Navigating PP Routing

3.1 Which Came First. . . ?

There's an inherent chicken–and–egg problem in describing PP routing: either the routing tables or the routing algorithm must be described first in detail. The detailed description of the routing tables can be bewildering without an understanding of what one is trying to accomplish with the tables. Therefore, the routing algorithm will be presented first. The five fundamental tables used for routing are introduced very generally below. The reader is encouraged to refer to Chapter 4 for a detailed description of the fundamental routing tables. Most likely, understanding the routing algorithm and routing table roles and capabilities will require an iterative process between this chapter and the following chapter.

3.2 Fundamental Routing Tables

The *aliases* table governs the handling of aliases for user names. It can be used to map non-users to usernames, redirect users who have moved, rewrite users addresses, map aliases, and identify ambiguous users.

The *users* table determines the local delivery channel (or channels) for a particular local user.

The *domain* table is used to derive a fully-qualified domain from any aliases or short-form names, and to obtain a pointer to the routing information for that domain.

The *or* table provides routing and normalisation for X.400 address in the same way the *domain* table does for RFC 822 addresses.

The *channel* table is used to discover the possible channels, and in some cases relaying MTA-name, that may be used to send a message on towards a remote host. This remote host is the one identified by the entries in either the *or* table or the *domain* table.

3.3 Basic Routing Process

When a message arrives, *submit* uses the *domain*, *or*, *users*, *aliases* and *channel* tables to identify delivery routes for the addresses in the message. The *or2rfc*, *rfc2or* and *rfc1148gate* tables are used to convert between X.400 style addresses and RFC-822 style addresses and vice versa.

The parsing of an address proceeds as follows:

- An address arrives in one particular form – either an X.400 style address or an RFC-822 style address.
- The address is normalised via the appropriate table – the *or* table for X.400 style addresses, the *domain* table for RFC-822 style addresses.
- There are three possible outcomes of the normalisation process.
  1. The address is unrecognised
  2. The address is recognised as a remote address.
  3. The address is recognised as a local address.
- If the address is recognised as a remote address, routing information to the remote site is accessed via the *channel* table.
- If the address is recognised as local, the local address is first looked up in the *aliases* table.

  If an entry is found in this table, the parsing process recurses with the value found in the *aliases* table.

  If no entry is found in the *aliases* table, routing information for the local address is accessed via the *users* table.
- Having done the normalisation of the address and accessed the routing information for that address, the address is converted into the other style of address (i.e. X.400 style addresses are converted to RFC-822 style and vice versa). This conversion is done according to RFC-1327 and uses the *or2rfc*, *rfc2or*, and *rfc1148gate* tables.

  Note both styles of addresses are kept by PP. Also this conversion between styles is always possible.
- At this stage PP has parsed one style of address. If this parsing has resulting in routing information for that address, the address is valid. However the address still has to pass the authorisation stage before *submit* accepts it.

  If the parsing does not result in routing information, the address is not valid in that form. PP then switches to the other style of address and attempts to parse that form.

  There is a failure point. This occurs when PP decides all avenues and styles of addresses have been followed.

3.4 Routing in Depth

Figure 3.1 attempts to represent the interaction of the routing algorithm with the various tables. It is important to note that the figure is not designed to stand alone—rather, it can only be properly understood in light of the clarifying text in this section.

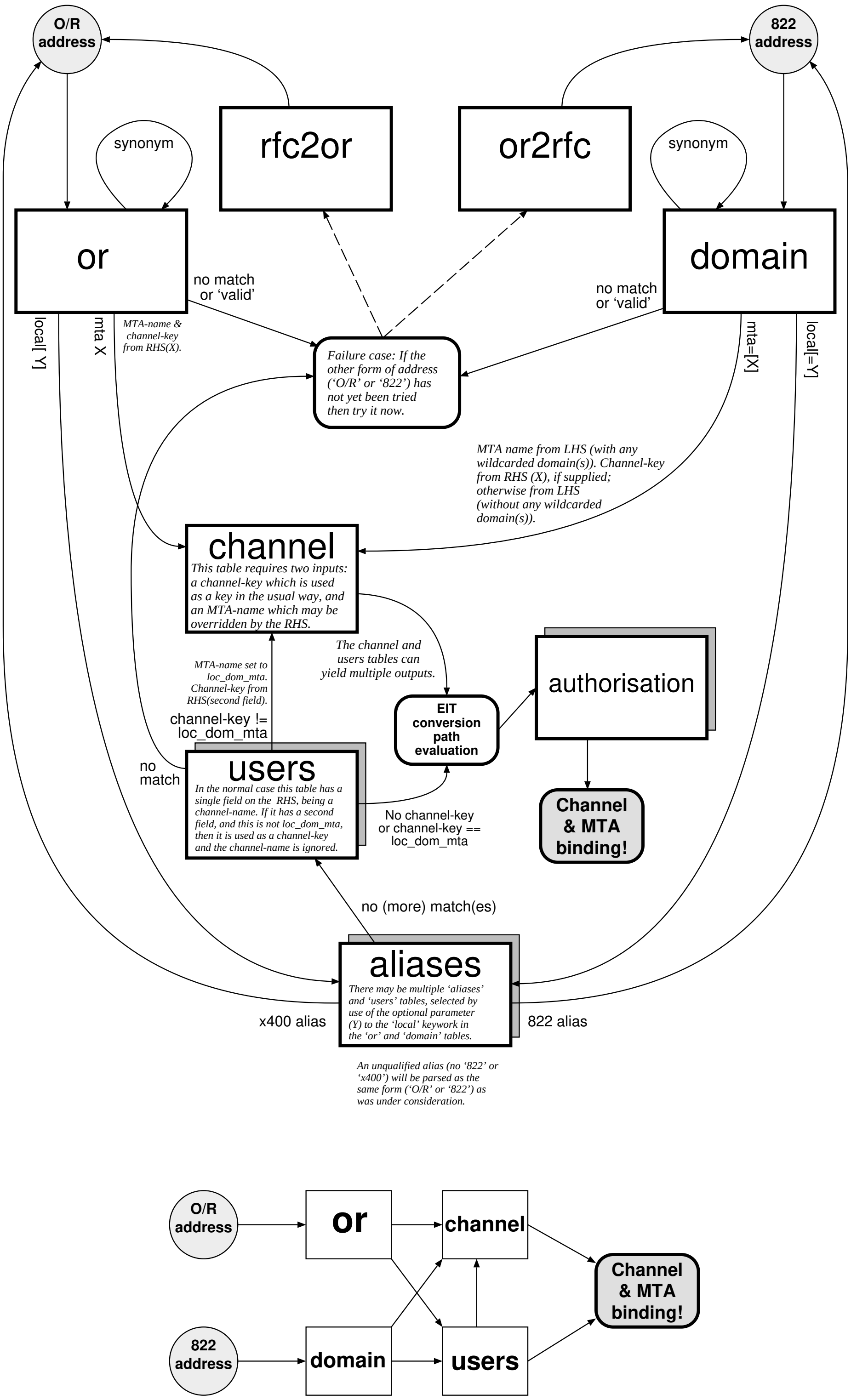

Figure 3.1: Overall Use of PP Tables

3.4.1 Detailed Table Outputs

The basic routing case is as follows:

1. An address enters the system and the associated routing table is consulted (X.400 by the *or* table, RFC822 by the *domain* table).
2. Assume the address is not a local address. The routing table lookup produces an MTA-name and a Channel-key value as input to the *channel* table.
3. Based on the defined routes in the *channel* table and the means of reaching them, the *channel* table produces a (possibly new) MTA-name′ value and the channel which should be used to reach MTA-name′.

The key to understanding things is that there are three important variables and that these get manipulated in various (and slightly inconsistent) ways:

1. MTA-name - the name of the MTA-name to which to deliver - to some extent this may be defined from some purely local namespace (but this is not the case when SMTP is in use).
2. Channel-key - some local name used simply as a key into the channel table.
3. The name of the channel with which to deliver the message.

Upon entry to the *channel* table, an address has associated with it a Channel-key and an MTA-name. The Channel-key is used as the index into the table and the RHS of the entry yields the delivery channel name. The RHS may override the MTA-name, yielding MTA-name′, or it may leave it alone. The use of MTA-name′ depends upon the output channel: in most cases this is just used as a key into that channel's outbound table, but for the SMTP channel it is looked up in the DNS or the local host table (usually /etc/hosts). The fact that these are the two data items associated with an address on entry to the *channel* table is critical, and where these values come from depending on the preceding table is crucial.

or **table:** when the table entry is of the form "mta MtaKey", Channel-key and MTA-name are both set to MtaKey.

domain **table:** when the table entry is of the form mta=*value*

1. If *value* is present
   - Channel-key is set to *value*
   - MTA-name is set to whatever matched the LHS of the table (the key that was used to look up in the *domain* table)
2. If value is not present
   - Channel-key is set to the LHS value minus a "*." if present
   - MTA-name is set as in case 1.

users **table:** when the RHS value includes an MTA-name value and that value is not equal to `loc_dom_mta` from the *tailor* file,

- Channel-key is set to this mta value [thus it's description as an MTA-name value is, at best, confusing]

- MTA-name takes the value of `loc_dom_mta` from the tailor file and is thus probably useless and should almost certainly be overridden by the value of the RHS of the relevant channel table value.

  Note: this Channel-key from the *users* table is used as a direct key into the *channel* table, and does not go via the usual address lookup of the *domain* or *or* tables.

Although this whole behavior is rather difficult to describe, it is powerful and can be most useful.

3.4.2 Choice of Outgoing Channel

It should be clearly pointed out that the choice between X.400-type and RFC822-type outbound channels is entirely independent of the format (RFC822 or X.400) in which a message comes in *and* of which table (*or* or *domain*) in which the routing is done.

It is quite possible to support both RFC822 and X.400 outbound channels using only one of the *or* or *domain* tables; but the different lookup characteristics of the tables and/or the potential absence of relevant mappings in the *or2rfc* and *rfc2or* tables may mean that not all the different types of routing policies can be exercised in this case.

The choice of which outbound channel to use, when more than one is available, is based upon authorisation restrictions and weight calculations. This weight is the sum of the costs of the individual channels that the message would have to pass through to go out on a given channel, and is usually specified in terms of reformatting (processing) cost rather than monetary cost.

Note that the cost calculation only plays a role in routing decisions when the MTA-name and Channel-key values input to the *channel* table could be routed via multiple paths.

3.5 Routing Tools

The tool *ckadr* can be used to check the action of parsing on various address. This and other tools are described in Chapter 7.

Chapter 4

PP Tables

This chapter describes the purpose and syntax of the various tables used by PP. As discussed in the previous chapter, the The *aliases* table, the *users* table, the *domain* table, the *channel* table and the *or* table are the primary tables used for routing and message delivery. There are also a set of tables containing authorisation information and a family of tables that provide mappings between X.400 O/R names and RFC 822 domain names. Specific channels may also have their own tables containing the information they require.

4.1 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 LHS must be preceded by a backslash "\" to escape it. All other characters are copied verbatim. However, some tables undergo further processing and may need other escape sequences.

The purposes and details of the various tables now follow.

4.2 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 usernames. (e.g., "postmaster" to some "user").
- Redirecting users who have moved (e.g., "fred" to "fred@foo.edu").
- Rewriting users addresses (e.g., "jpo" to "j.onions").
- Mapping aliases. (e.g., "list-request" to "j.onions").

- Identifying ambiguous users.

Entries in this table are constructed as follows:

```
<key> ":" <type>  <value> <qualifier>
```

The `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.

`x400`: a remote X400 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. This qualifier is explained by example in Section 4.7

The format is best illustrated by a sample extract of the table as shown in Figure 4.1.

The table includes an entry for a local alias `mailgroup-request`. The value `Irene.Hassell` 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 authorisation will be done on the basis of `mailgroup-request` rather than `Irene.Hassell`.

The next entries indicate that `a.dacruz` and `alina` are both synonyms for `Alina.DaCruz`. Here, the user's address will be converted to Alina.DaCruz, and this value will be used in future table searches.

An entry is given for a remote RFC 822 user, `jpo`. The value of `jpo@nexor.co.uk` is parsed as an RFC 822 address.

Then follows an entry for a remote X400 user, `pc`. The value

```
# sample aliases
#
mailgroup-request:alias Irene.Hassell
postmaster:alias Irene.Hassell
pp:alias postmaster
#
a.dacruz:synonym Alina.DaCruz
alina:synonym Alina.DaCruz
#
jpo:alias jpo@nexor.co.uk 822
#
pc:alias "/I=P/S=Cowen/O=nexor/PRMD=NeXor/ADMD= /c=gb/" x400
#
f.bloggs:synonym f.bloggs@foo.bar 822 external
```

Figure 4.1: Example of Alias Table

```
"/I=P/S=Cowen/O=NeXor/PRMD=NeXor/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.bloggs` is an external synonym. The address `f.bloggs` will be replaced by `f.bloggs@admin.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. This is explained in greater detail in Section 4.7.

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

4.3 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> ] ...]
```

`Channel`: specifies the name of a local delivery channel on `mta`.

`Mta`: is the local machine on which the mailbox resides. If no MTA is indicated, the channel may be used for delivery by any MTA using these tables. 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 one domain.

A user can have several channels for local delivery. For example, in Figure 4.2 `John.Taylor` can have his mail delivered either by the *822-local* channel or by the *slocal* channel. Which of these two channels is actually used for delivery depends on the format of the message being delivered. This allows a user to have, for instance, X.400 format mail delivered by one channel, and RFC 822 mail by another channel.

```
#################### extract of 'users' table ####################
#
Alina.DaCruz:822-local vs2.cs.ucl.ac.uk
Steve.Kille:822-local pyr1.cs.ucl.ac.uk
John.Taylor:822-local vs2.cs.ucl.ac.uk, slocal vs2.cs.ucl.ac.uk
Postie.Pat:822-local vs2.cs.ucl.ac.uk
x400-users:list
info-server:shell
#
```

Figure 4.2: Example of Users Table

Figure 4.2 illustrates how the various local users will receive mail and on which machines their mailboxes reside. 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.

4.4 The Domain Table

The *domain* table is used by *submit*

- to derive a fully-qualified domain from any aliases or short-form names;
- and to obtain a pointer to the routing information for that domain.

See Section 5.1 on page 78 for a description of how to go about constructing this table.

Entries in the domain table are encoded in the form:

```
<LHS> ":" <qualified-entry> {"|" <qualified-entry>}
<qualified-entry> ::= <key> ["=" <value>] {[<key>["="<value>]]}
```

Left Hand Side (LHS) Constructions

`LHS` has two possible constructions:

`dmn` This construct is used for exact matches. No subdomains are allowed in this entry. This effectively sets `min=0` and `max=0` and cannot be altered.

`*.dmn` This construct is used to specify partial domain matches. Unless altered by the `min` and `max` keys described later, constructs of this form will match domains with at least one subdomain (i.e., `min=1` and `max=*`).

Note that an entry of `*` with no domain components specified at all, represents the `default domain`. If the `default domain` is present in the *domain* table, then any top level domains which are not matched in the table, are matched by this `default domain` entry.

Qualified Entries

Each qualified entry may have several `key=value` components. The keys have two uses: to provide the information associated with the entry; and secondly to aid in matching the LHS. The possible types of `key` and what their values represent are as follows:

`max=<number>`: This specifies the maximum number of subdomains for the entry. A value of `*` indicates that the maximum number of subdomains should not be used when testing for a match, effectively setting the maximum number of subdomains to infinity. This is the default value for `max` on non-exact match entries.

`min=<number>`: This specifies the minimum number of subdomains for the entry. As with the `max` value above, a value of `*` indicates that the minimum restriction should not be used when testing for a match. Effectively `min=0`. The default value for `min` is `1`.

`norm=<dmn-name>`: This identifies the fully-qualified domain name. If `<dmn-name>` isn't specified, the fully-qualified domain name is taken as `LHS` (without the `*.` wildcard prefix if present).

`mta=<chan-ref>`: This provides a pointer to where the routing information for this domain is stored in the *channel* table. If `<chan-ref>` isn't specified, the pointer is taken as `LHS` (without the `*.` wildcard prefix).

`norm+mta=<dmn-name>`: This key is a combination of the `mta` and `norm` keys described above (i.e. `norm+mta=<dmn-name>` is the same as `norm=<dmn-name> mta=<dmn-name>`).

`mta+norm=<dmn-name>`: This key is identical to the `norm+mta` key described above.

`synonym=<dmn-name>`: This identifies a synonym. The `LHS` key is replaced by `<dmn-name>` and parsing continues by looking for the new `LHS` in the domain table.

If the `<dmn-name>` is prefixed with the wildcard symbol `*.`, the new `LHS` is constructed from `<dmn-name>` and the subdomains that a wildcard represents in the original `LHS`. For example a domain of `foo.bar.zz` matching an entry of `*.zz:synonym=yy` would result in a table lookup of `foo.bar.yy`. Note that this is only applicable to partial `LHS` entries.

`local=<subtbl-name>`: This indicates that `LHS` is local. Any host that needs to be recognised as local should have this keyword - this includes `loc_dom_site`. With no `<subtbl-name>`, this indicates that `LHS` is a local reference whose information is contained in the default tables *aliases* and *users*. With a `<subtbl-name>` specified, this indicates separate local alias and user tables. This feature is used when one MTA is supporting multiple local departments, each with its own user and alias tables. This is described more fully in Section 4.7.

`valid`: This is shorthand for the entry `norm=LHS` e.g `*.fr:valid` is the same as `*.fr:norm=fr`. It implies that the domain is known but not routable directly. This may be used to indicate the domain is valid and may be used for RFC 1327 mappings. It may also be used to stop use of the default route. If a top level domain has an entry in the *domain* table, it will not be routed via the default route.

Domain table lookup

The right hand side of a *domain* table entry is a sequence of qualified entries. The qualification is based upon the number of subdomains below the key being looked up (i.e. the number of domain components the wildcard prefix `*.` represents).

For example, take the domain “1.2.3.nexor.co.uk”. When attempting to look this domain up in the table, several different keys may be used in the following order:

- “1.2.3.nexor.co.uk” the exact match,
- “*.1.2.3.nexor.co.uk” where the number of subdomains is zero,
- “*.2.3.nexor.co.uk” where the number of subdomains is one (i.e. `*` represents “1”)
- “*.3.nexor.co.uk” where the number of subdomains is two (`*` represents “1.2”)
- and so on.

PP will take the qualified entries in turn and choose the first one whose restrictions on the number of subdomain components is met by the number of subdomain components actually present. The restrictions for an entry are set by the `min` and `max` keys.

Note that all occurences of domain references in the *domain* table must be specified in US order (littleendian). If the appropriate flag is set, the routines accessing the *domain* table will “do the right thing” for UK ordering.

Domain lookup results

As mentioned above, the *domain* table is used to provide two items:

1. The fully-qualified domain name.

   If a match is found in the table then this is always provided. If it is not explicitly provided by the `norm` key or a combination containing that key, then it is implicitly provided as the left hand side of the entry matched (i.e. the key used to index the *domain* table) For example with the entry

   ```
   nexor.co.uk:local
   ```

   The fully-qualified name is `nexor.co.uk`. If the left hand side is a partial domain construct, the prefix `*.` is omitted from the fully-qualified name.

   If any subdomains exist they are prepended to the fully-qualified name to reconstruct the full domain. For example with the entry

   ```
   *.gb:norm=uk
   ```

   and the address `nexor.co.gb`, the fully-qualified name produced from the table would be `uk`. After the re-addition of the subdomains, this would become `nexor.co.uk`.

2. The pointer to the routing information.

   The pointer into the *channel* table may be omitted. In which case the fully-qualified domain name has been derived but no routing information associated with it. This is useful when you wish to normalise the domain in message headers but you do not wish to deliver mail to that domain. It is also of use when the route to that domain is specifically an x400 route and as such the pointer to the routing information is specified in the *or* table described in Section 4.5.

```
*.edu:mta=nameserver
cs.nott.ac.uk:mta=cs.nott.ac.uk
computer-science.nott.ac.uk:norm+mta=cs.nott.ac.uk
*.fr:valid
*.co.uk:min=0 max=0 mta=co.uk|mta=server.co.uk
*:mta=nameserver
```

Figure 4.3: Example of Domain Table

Examples

Figure 4.3 contains several examples entries.

The entries in this figure imply the following actions:

- The `*.edu` entry will match all domain names with the most significant component of `edu` and at least one subdomain. Thus, the hosts `ucbarpa.berkeley.edu`, `prep.ai.mit.edu` and `foo.edu` would all be matched by this entry. However an address of `user@edu` would not be matched.

  The fully-qualified domain name is constructed with the `edu` component as the most significant component.

  The key `nameserver` will be used in the lookup of the channel table to find routing information for this domain.

- The `cs.nott.ac.uk` entry will only match the exact name `cs.nott.ac.uk`. It will not match `crg.cs.nott.ac.uk` or `admin.cs.nott.ac.uk`.

  The fully-qualified domain name is `cs.nott.ac.uk`.

  The key `cs.nott.ac.uk` will be used to access the routing information in the *channel* table.

- The `computer-science.nott.ac.uk` entry will only match the exact name `computer-science.nott.ac.uk`. The fully-qualified domain name is `cs.nott.ac.uk`. This string is also used to access the routing information in the *channel* table.

- The `*.fr` entry will match all domains with the most significant component of `fr` and at least one subdomain. So it will match `sorbonne.fr` or `nice.fr` but it will not match `fr` by itself.

  The fully-qualified domain name is constructed with the `fr` component as the most significant component.

  No pointer to the routing information is specified. So when parsing addresses of the form `nice.fr`, PP will match this entry which indicates the correct domain ordering. However with no routing information for this form of the address, PP will convert the address to X.400 as described in RFC 1327 and then attempt to find routing information from the X.400 equivalent of the *domain* table – the *or* table.

  There is a similar mechanism in the *or* table. Using these two mechanisms in tandem, you can keep pointers to X.400 routing information in the *or* table and pointers to RFC-822 routing information in the *domain* table.

  As a side effect, because the top level domain `fr` has been matched, the default route will not be considered as a possibility.

- The `*.co.uk` entry has two qualified entries.

  The first entry will only match the exact domain `co.uk` and nothing else.

  The second entry will match all other domains with `co.uk` as the two most significant components i.e `nexor.co.uk`, `bnr.co.uk` etc.

  The fully qualified domain name in both cases is constructed with the `co.uk` as the two most significant components. In the first case, there are no subdomain components to add so the domain name is `co.uk`.

  The pointer to the routing information differs for the two cases. In the first case it is `co.uk`. In the second it is `server.co.uk`. This enables you to route differently depending on the number of subdomain components.

- The `*` entry is an example use of the `default domain`. It will match with anything that does not match any other rule. The fully qualified domain name will be the same as on input. The pointer to the routing information will be `nameserver`.

  In the extreme case where you wish every domain to be routed via domain name server technology, this can be the only entry in the domain table.

Note: The default route should be used with care. It is tempting to use this as a catch all. This makes routing less efficient as every domain that uses this will be sent to one host, effectively passing the buck. It also means that users making small typo's in the top level domain may have their mail routed a large distance before the mistake is found.

Note: It is important to put the relevant entries for the local domains in the *domain* table, especially `loc_dom_site` and `loc_dom_mta`. When tables are being shared between multiple local MTAs, it may be best to effect this latter entry via an `override` parameter to the table definition in the tailor file.

Warning: If local domains are not defined as local in the tailor file, local addresses will not be recognised as local.

4.5 The O/R Address Table

The *or* table provides routing and normalisation for X.400 address in the same way the *domain* table does for RFC 822 addresses. Lines are encoded in the form:

```
<key> ":" <type> <value> ...
```

In the <key> field, the characters ":" and "." must be preceded by a "\" character.

The <type> and <value> fields on the RHS are separated by white space, and so any space characters in those fields must similarly be preceded by either "\" or else the value should be encapsulated in double quotes.

The *or* table is used to evaluate O/R addresses. The <key> is encoded as dmn-orname as specified in RFC 1327 i.e. a sequence of <component>$<value> pairs separated by full stops.

The algorithm for O/R address analysis should be considered as a "tree-walking" algorithm. The O/R address components are sorted, and components are looked up: first level 1; then level 2 + level 1 and so on. So repeated lookups are carried out, using this table. The first failed lookup breaks the loop with the last "routable" match (mta or local matches) being chosen as the result of the analysis. If there is no "routable" match, the analysis has failed and the O/R address is invalid.

There is a wildcard facility that may be used when constructing the LHS, <key>. To use the wildcard facility, set the <value> to the wildcard symbol *. For example, consider the case where you wish to route all unknown ADMDs under one country to one mta. This can be achieved by the following entry.

```
ADMD$*.C$GB:mta admd.gb
```

Given the address /S=pac/O=NeXor/PRMD=NeXor/ADMD= /C=GB/, the lookup of the O/R table will proceed as this

- lookup C$GB, assume it succeeds
- lookup ADMD$ .C$GB, this fails
- lookup ADMD$*.C$GB, this matches the example entry
- continue looking up PRMD$NeXor$ADMD$*.C$GB and so on till lookup fails. When the lookup fails, the last successful mta or local entry is used for the appropriate routing.

There are four possible instances of the <type> field, similar in functionality and appearance to their counterparts in the *domain* table.

valid: This means that this part of the tree is OK. As lookup stops as soon as an entry is not found, it is necessary to mark entries which have no routing information as valid to allow lower entries to be found.

mta <mtaname>: This identifies the MTA <mtaname> as being associated with this vertex of the tree. This <mtaname> is also a key into the PP *channel* table. The search should not stop at this point, as further tree searching may determine a more optimal mta.

synonym <dmn-orname>: This identifies a synonym, and indicates that all O/R components matched so far should be replaced with those identified on the RHS, and the search should now resume from the root. The characters "\" and "." in the <component> and/or <value> parts of the dmn-orname construction must always be preceded by a "\".

For historical reasons, this encoding has a second equivalent form:
alias <dmn-orname>.

local [<subdom-name>]: This identifies a local O/R Address. The remainder of the O/R address should be used to determine a local mailbox. If the subdom-name is empty the normal *alias* and *user* tables are consulted. If the subdom-name is present, then as with the similar feature in the *domain* table, this feature is used when one MTA is supporting multiple local departments, each with their own user and alias tables. This is described in Section 4.7.

Figure 4.4 gives an example which shows the validity of some Country, ADMD, and PRMD values, and indicates which MTAs will deal with which PRMDs. The MTA value given here is internal to the PP lookup system, providing a key with which to reference the channel table.

```
#
# extract of 'or' table
#
#       -*** gb ***-
#
C$GB:valid
C$UK:synonym C$GB
ADMD$GOLD 400.C$GB:valid
PRMD$UK\.AC.ADMD$GOLD 400.C$GB:valid
OU$NoOrg.PRMD$UK\.AC.ADMD$GOLD 400.C$GB:mta NoOrg
#
#
#       -*** sites connecting via gold 400 ADMD ***-
#
PRMD$DIGITAL.ADMD$GOLD 400.C$GB:mta bt-gold
PRMD$TELECOM GOLD.ADMD$GOLD\ 400.C$GB:mta bt-gold
OU$CS.O$UCL.PRMD$UK\.AC.ADMD$GOLD\ 400.C$GB:local
#
#
#       -*** france ***-
#
C$FR:valid
ADMD$PTT.C$FR:mta emu-france
ADMD$ATLAS.C$FR:mta emu-france
PRMD$BULLMTS.ADMD$ATLAS.C$FR:mta MYBULL
PRMD$DERIEUX.ADMD$ATLAS.C$FR:mta persona
#
```

Figure 4.4: Example of Or Table

4.6 The Channel Table

Submit uses the *channel* table to discover the possible channels, and in some cases relaying MTAs, that may be used to send a message on towards a remote host. This remote host is the one identified by the `mta` entries in either the *or* table or the *domain* table.

See Section 5.1 on page 79 for a method of building this table. The format of an entry has the form:

```
<key> ":" [ <mtaname> ] "("<channel>")"
          ["," [ <mtaname> ] "("<channel>")" ...]
```

`key`: is the index name derived from the mta entry in the *domain* or *or* tables – the remote host.

`mtaname/channel`: the `mtaname` derived from this lookup will be used as a key in the appropriate outbound `channel`'s table – the relaying MTA. If there is no `mtaname` present, then the remote host is reached directly through that channel. This sort of action is appropriate if the channel uses external information (such as the DNS) to look up the domain.

Figure 4.5 illustrates an extract of this table.

There may be a number of possible channels for a given remote host. Everything else being equal, the order of channels in the table is the preferred method of delivery. The authorisation process will select one of the possible channels and this is the one used.

```
#
#
a.cs.uiuc.edu:a.cs.uiuc.edu(smtp)
#
bsec.abcy.uwist.ac.uk:bsec.abcy.uwist.ac.uk(gb-janet)
bt-gold:bt-gold(x400out84)
bte:bte(x400out84)
abcl.co.uk:ukc.ac.uk(gb-janet),ukc.ac.uk(gb-pss)
btvax.ulster.ac.uk:btvax.ulster.ac.uk(gb-janet)
bull:bull(x400out84)
bunny.ulcc.ac.uk:bunny.ulcc.ac.uk(gb-janet)
#
emu-france:emu-france(x400out84)
#
northern.co.uk:ukc.ac.uk(gb-janet),stl.stc.co.uk(gb-pss)
#
# These are resolved directly by the DNS - no relay
edu:(smtp)
gov:(smtp)
com:(smtp)
us:(smtp)
mil:(smtp)
org:(smtp)
```

Figure 4.5: Example of Channel Table

4.7 Supporting Multiple Institutions on One MTA

PP 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 *alias* tables,
- and the `-internal` and `-external` flags to the header normalisation filters, *p2norm* and *rfc822norm*.

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, `nexor.co.uk`, and a subdomain under that site, `admin.nexor.co.uk`.

1. **PP acting as two different, independent mail hubs.**

This is the situation where one PP system is acting as two distinct domains.

The main domain `nexor.co.uk` has the information relating to its local users in the tables named *users* and *aliases*, and the subdomain `admin.nexor.co.uk` has the information relating to its local users in the tables *admin-users* and *admin-aliases*. There is no intersection between the `nexor.co.uk` name space and the `admin.nexor.co.uk` name space so these tables can be independently maintained.

PP has to be informed that `admin.nexor.co.uk` is a local domain. This is done via the relevant entry in the *domain* table and/or the *or* table. e.g.

```
admin.nexor.co.uk:local=admin
OU$admin.O$NeXor.PRMD$NeXor.ADMD$ .C$GB:local admin
```

2. **PP acting as one name space divided over two realms.**

This is the situation where PP is acting as one complete name space composed of two separate realms, i.e. all users have mail addresses of the form `user@nexor.co.uk`.

As above the information about local users is divided into two separate sets of tables, *users* and *aliases*, and *admin-users* and *admin-aliases*. As above PP is informed that `admin.nexor.co.uk` is a local domain. But as these two domains are to be combined into one name space – `nexor.co.uk`, there has to be an intersection between the two sets of tables and in particular between the *aliases* tables.

The intersection only applies to addresses in the `admin.nexor.co.uk` realm. The delivery information for these users is stored in the *admin-users* table. On delivery, PP has to be able to access this table. Therefore it has to map from the address `user@nexor.co.uk` to `user@admin.nexor.co.uk`. This is done via an entry in the *aliases* table of the form:

```
user:synonym user@admin.nexor.co.uk 822
```

To ensure that the name space is maintained, one has also to ensure that the address `user@admin.nexor.co.uk` is mapped to `user@nexor.co.uk` for senders and in headers. This is so, because the user agents on the subdomain may stamp mail with their subdomain's name i.e. `admin.nexor.co.uk` To acheive this mapping, there should be an entry in the *admin-aliases* table of the form:

```
user:synonym user@nexor.co.uk 822 external
```

The `external` qualifier prevents this `synonym` being unwound if the responsibility for the address we are parsing is set (i.e we are trying to delivery to that address, it is for internal consumption). Unless the flags described below are used, the `synonym` is always unwound for addresses in headers.

WARNING: with the two entries in the two different *aliases* tables, there is a possiblility of circular aliases. The use of the `external` qualifier stops the circles, but it is adviseable to use the `ckadr` tool to check some example addresses before going on-line with such a configuration.

3. **PP acting as a gateway between the external world and the internal domains.**

The previous configuration suffices for a gateway between the outside world and the internal domains. However it may swamp the gateway machine with internal messages. All users are represented in headers as `user@nexor.co.uk` irrespective of which subdomain they are under. So any replies to messages will be directed to `user@nexor.co.uk` which is the gateway. It would be better if internal recipients of the message saw the internal addresses. Hence could bypass the gateway and send directly to the relevant domain.

This is an issue which doesn't have any relevance for the parsing of envelope addresses. It only effects the headers presented to the user (both x400 headers and RFC 822 headers). As such, one has to set up the PP tailor file and routing information so that messages going to internal recipients have their headers reformatted differently than messages going to external recipients.

To obtain the different reformatting, one should tailor the `-internal` and/or the `-external` flags for the *p2norm* and *rfc822norm* filters. By default, the header normalisation filters unwind all `synonym`'s including those marked with the `external` qualifier. The presence of the `-internal` flag stops the filters unwinding the `synonym`'s marked `external`. Thus leaving `user@admin.nexor.co.uk` in headers.

To go about this tailoring change:

- declare new `headertype`'s of the form 822-XXX, p2-XXX and p22-XXX. Whether you need all of these depends on which protocols you are using, RFC 822, X.400(84) or/and X.400(88).
- identify the channels which transfer to and receive from the `admin` domain. This may involve spliting an existing channel say `smtp` into two `smtp` and `smtp-admin` (see Section 2.6 on how to split a channel on inbound).
- on these channels, set `hdrin` and `hdrout` to 822-XXX, p2-XXX, or p22-XXX depending on the underlying protocol of the channel in question.
- create new reformatters to normalise standard headers into headers suitable for the `admin` domain. These reformatters should be run by the filter control channel and be either the filter `rfc822norm` for RFC 822 based headers or the `p2norm` filter for X.400 based headers and should have the `-internal` flag set. e.g

```
chan 822toXXX
prog=fcontrol,type=shaper,
hdrin=822,
hdrout=822-XXX,
out-info="rfc822norm -internal -adr-order ukpref -jnt -bigend"
```

4.8 RFC 1327 Mapping Tables

These mappings are used to convert between X.400 addressing and RFC 822 addressing. The mappings involved are not algorithmic, making necessary the use of such tables. RFC 1327 describes in detail their function. See Appendix E for how to obtains copies of the globally maintained tables in PP format.

4.8.1 The O/R to RFC 822 Table

The table *or2rfc*, exemplified in Figure 4.6, is for mapping OR names into RFC 822 addresses. The `<key>` is encoded as domain-syntax (see RFC 987 and RFC 1026) and `value` as dmn-orname. The longest possible match is sought.

4.8.2 The RFC 822 to O/R table

Performing exactly the reverse operation is *rfc2or*, which has the `<key>` encoded as dmn-orname and the `<value>` as domain-syntax. An example of the format is given in Figure 4.7.

```
#
#       -*** general GB ***-
#
PRMD$UK\.AC.ADMD$GOLD 400.C$GB:ac.uk
ADMD$GOLD 400.C$GB:gold-400.gb
#
#       -*** via gold 400 ADMD ***-
#
PRMD$TELECOM GOLD.ADMD$GOLD 400.C$GB:telecom-gold.gold-400.gb
PRMD$DIGITAL.ADMD$GOLD 400.C$GB:digital.gold-400.gb
#
#       -*** france ***-
#
ADMD$ptt.C$fr:ptt.fr
PRMD$aristote.ADMD$ptt.C$fr:aristote.fr
PRMD$BULLMTS.ADMD$ATLAS.C$FR:bullmts.fr
#
```

Figure 4.6: Example of or2rfc Table

4.8.3 The Known RFC 1327/RFC 1148 Gateway Table

The *rfc1148gate* table has the same syntax as the *rfc2or* table. Its entries represent gateways between X.400 and RFC 822 (via RFC 1327). This table is used in the cases where RFC 822 messages are relayed across an X.400 link. In such cases mapping to X.400 addresses may not always make sense as there will effectively be routing information encapsulated in the address. Normally this would cause the message to fail. However, if it is known that the destination is an RFC 1327/RFC 1148 gateway, then the mapping can proceed in the expectation that the other end can unwrap the message.

An example RFC 1148 gateway file is shown in Figure 4.8.

4.9 PP Authorisation

Authorisation is applied after address parsing, normalisation and validation, to establish whether the message is permitted to be transferred on a per recipient basis. Address lookup may establish several possible outbound channels for each recipient; authorisation selects one of these after applying tests based on entries in the three authorisation tables *auth.channel*, *auth.mta*, and *auth.user*.

Access policy for channel pairs (inbound + outbound) determines how the rights for the four entities: sender, recipient, sending MTA, and destination MTA, are to be interpreted. (“Rights” means one of the four values in/out/both/none and a fifth value “unset” to indicate that no table entry is present.)

The intention is that one of these four may be shown to be authorising the message (i.e., willing to pay for it).

It should be noted that authorisation is not applied to delivery report messages.

```
#
# extract of 'rfc2or' table
#
gold-400.gb:ADMD$GOLD 400.C$GB
#
#       -*** via gold 400 ADMD ***-
#
telecom-gold.gold-400.gb:PRMD$TELECOM GOLD.ADMD$GOLD 400.C$GB
digital.gold-400.gb:PRMD$DIGITAL.ADMD$GOLD 400.C$GB
#
#
#       -*** france ***-
#
fr:PRMD$fnet.ADMD$ptt.C$fr
ptt.fr:ADMD$PTT.C$Fr
aristote.fr:PRMD$aristote.ADMD$PTT.C$Fr
bullmts.fr:PRMD$BULLMTS.ADMD$ATLAS.C$FR
derieux.fr:PRMD$DERIEUX.ADMD$ATLAS.C$FR
#
```

Figure 4.7: Example of rfc2or Table

```
ca:O$ubc.PRMD$cdn.ADMD$telecom\.canada.C$ca
uk:OU$CS.O$UCL.PRMD$UK\.AC.ADMD$ .C$GB
```

Figure 4.8: Example of RFC 1148 Gateway Table

4.9.1 Access Policy

The access policy is one of the following:

`none`**:** Message may not be transferred.

`free`**:** No constraints on the four entities.

`block` **(i.e., "blockable"):** - at least one of the four entities must enable the message by specifying "in" or "both" for the inbound channel and "out" or "both" for the outbound channel. If none of the four enable it, the message may not be transferred. Therefore the default is to not let messages through. If a message fails in this mode, it effectively means that the message was not authorised.

`negative`**:** Any one of the four entities may stop transfer by disabling the message. This implies the default state is to let everything through, and special cases are stopped by explicit configuration. If a message fails in this mode, it effectively means that the message was prohibited.

The default access policy is determined by the entry `*->*` in the *auth.channel* table. If this entry is not present, the default access policy is `block`.

When determining rights for a proposed transfer the following algorithm is followed:

- Determine the inbound and outbound channels and look up this pair in the *auth.channel* file.

If there is no entry in this file, take the default from the tailoring variable as the access policy for the channel.

- If the access policy is free or none, then the message is allowed through or rejected respectively.
- If the access policy is negative, check the MTA and user tables, *auth.mta*, and *auth.user* to see if this route is explicitly disabled.
- If the access policy is block, check the user and MTA tables to see if the route is explicitly enabled.

4.9.2 Table Formats

There are three tables associated with authorisation, plus the tailor variables.

Channel Authorisation

The first of these tables is the *auth.channel* table. This contains the overall policies for channel to channel relaying. As the key, the table has the syntax:

```
<inchannel> "->" <outchannel>
```

If no value is found for the above, then as a fallback the values

```
<inchannel> "->" "*"
"*" "->" <outchannel>
"*" "->" "*"
```

are checked in that order.

The values that the table can have are as follows:

`<policy>`**:** Any of the above mentioned policies (`free`, `block` etc.).

`warnsender=<file>`**:** Send a warning message to the sender if the authorisation fails, with text taken from the given file (relative to the *wrndfldir*).

`warnrecipient=<file>`**:** Send a warning message to the recipient with text taken from the given file if the authorisation fails..

`sizelimit=<digits>`**:** Enforce the given size limit on messages. No messages larger than this size will be relayed. This is obeyed regardless of the overall policy of the channel. (I.e., A channel may be marked as free with a sizelimit, and the sizelimit will still be enforced.)

`test`**:** This setting modifies the checking procedure to always allow the message through. However, all the checking is carried out and the result is logged. This is useful to test out the results of a setting for a while before fully applying them. If either `warnsender` or `warnrecipient` is set, the warning message will be sent if the message would have normally failed the authorisation checks.

If warning messages are to be sent, the contents of the message are subject to macro expansion. The macros of the form `$(key)` are replaced with given values, as shown in Table 4.1.

An example of the *auth.channel* table format might look like the following:

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 (x400 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 (x400 format)
`ua-id`	the user agent id of the message
`p1-id`	the P1 id of the message
`locsite`	the local site name
`locmta`	the local mta name
`reason`	The reason for authorisation failure
`direction`	if the message is destined for the sender or the recipient

Table 4.1: Warning Message Macro Expansions

```
*->*:free
822-local->pss:free
822-local->local:block
822-local->slocal:free
822-local->smtp:free, warnsender=smtpwarnsender, test
822-local->janet:block, sizelimit=4000
x400in84->x400out84:none
x400in84->local:free,sizelimit=10000
822-local->x400out84:block,warnsender=restricted, test
smtp->smtp:block, test
```

MTA authorisation

The next table is the MTA policy table in file *auth.mta*. This governs the policy for a given MTA if the channel policy is not `free` or `none`. The keys used in this table are the source MTA and the destination MTA. Each of these are looked up separately to determine policies.

The allowed values for this field are any of the following in a comma-separated list:

`<channel>=<direction>`**:** This specifies for given channels what directions are allowed for this MTA. The possible directions are:

- `in`**:** Allow inbound traffic for this channel.
- `out`**:** Allow outbound traffic for this channel.
- `both`**:** Allow both inbound and outbound traffic for this channel.
- `none`**:** Allow no access for this channel.

`default=<direction>`**:** This specifies the default directions that traffic can flow in the absence of a particular channel being specified.

sender-requires=<pattern>: This MTA requires that the sender address being authorised matches the given pattern. This is a standard regular expression as specified in *regex* (3).

recip-requires=<pattern>: This MTA requires that the recipient address being authorised matches the given pattern.

requires=<pattern>: This sets both recip-requires and sender-requires.

sender-excludes=<pattern>: The MTA requires that the sender being authorised does not match the given pattern.

recip-excludes=<pattern>: The MTA requires that the recipient being authorised does not match the given pattern.

excludes=<pattern>: This sets both recip-excludes and sender-excludes.

bodypart-excludes=<bodyparts>: This specifies a list of bodyparts that the MTA will not accept. These bodyparts are taken from the list of standard bodyparts as described in Section 2.2.1 on page 7.

sizelimit=<digits>: The MTA will not relay messages larger than the given sizelimit.

An example of this table might look like the following:

```
vs2.cs.ucl.ac.uk:default=both, smtp=out, x400out84=out,
        822-local=in, excludes="A.DaCruz.*"
localhost:default=in, x400in84=out, smtp=both,
        822-local=none, sizelimit=654321
```

(Lines are folded for readability.)

User Authorisation

The user table is where both the recipient and originator are looked up to determine access rights. The key is either the full normalized name of the user, or if local, just the local part of the name. The values may contain any of the following:

<channel>=<direction>: This specifies, much as for the MTA, what directions are allowed for this user.

default=<direction>: This gives the default directions allowed for this user.

bodypart-excludes=<bodyparts>: A list of bodyparts that are not allowed to be sent to the user.

sizelimit=<digits>: Disallow messages larger than this sizelimit.

An example of the user authorisation table is:

```
j.taylor@cs.ucl.ac.uk:default=both
/I=J/S=Taylor/OU=cs/O=ucl/PRMD=uk.ac/ADMD=gold 400/C=gb/:\
  bodypart-excludes=g3fax|dmd, sizelimit=100000, 822-local=both
a.dacruz@cs.ucl.ac.uk:default=both
s.kille@cs.ucl.ac.uk:default=both
p.cowen@computer-science.nottingham.ac.uk:default=both, \
       822-local=in, x400out84=none
j.onions@computer-science.nottingham.ac.uk:default=both
```

4.10 Authentication

Components of PP must do appropriate checking that they are being invoked correctly. Note that authentication is complementary to authorisation. It is also a prerequisite.

4.10.1 Submit Authentication

Submit needs to verify the source of an inbound message. For messages from an inbound channel (protocol server) accessing the MTA Abstract Service, this means verification of the UID of the channel (typically to be “root” or “pp”). For the MTS Abstract Service, the identity of the P1 originator must be verified. These two verifications use the same mechanism. There are two cases:

1. Submission from a local process using pipes. In this case, the UID of the invoking process is available, and can be the basis for authentication.
2. For remote submission, the UID is not available, and so a password based mechanism is used. This may also be used locally.

A password and the corresponding name are carried in the initial protocol handshake. This may be overridden for later messages.

If one or more UIDs and a password are specified in the authentication table, just one of them must be matched for authentication to succeed.

Each (inbound) channel has an “auth-table” variable which identifies a file that contains the authentication table for that channel. Lines in the table have the following format:

```
<address> ":" [<login-name> ["," <login-name> ... ]] [ "|" <password>]
```

<address>: String-encoded email address, in the preferred format of the channel (either X.400 or RFC 822).

<login-name>: An authorised login name from the */etc/passwd* file. (Note that user login names are used, not numeric UIDs.) Zero or more login names can be specified, separated by commas.

<password>: Encrypted password (as in the */etc/passwd* file). Note that the password is optional.

Example files:

```
#
# Start of dummy 822 file
#
S.Kille@cs.ucl.ac.uk:steve|aiohpefcn

#
# Dummy X.400 file
#
/i=s/s=kille/ou=cs/o=ucl/prmd=uk.ac/admd= /c=gb/:steve,pp
/i=j/s=onions/o=nott/prmd=uk.ac/admd=gold 400/c=gb/:jonions
```

4.10.2 QMGR Authentication

Management tools like the *MTAconsole* can connect to the *qmgr* in two distinct modes. These are

`no authentication`: In this mode, no authentication is done for the user.

`weak authentication`: In this mode a name and password are passed across in the bind, so there is some limited authentication of the initiator.

When an incoming call is received by the qmgr, authentication information is checked in the *auth.qmgr* table. The format of this table is:

```
<name> ":" "passwd=" <password> "," "rights=" <rights>
```

These parameters are interpreted in the following way:

`name`: This is the name given in the bind parameters. If the connection was bound with no authentication, the name `anon` is used for the lookup. If the name is not found in the table, then if the user bound with authentication, the request is refused. If the user bound without authentication, and the name `anon` is not present in the table, then limited access is allowed.

`passwd`: This value takes an encrypted passwd, which can be generated with the PP utility *mkpasswd*. The password passed in the bind information is encrypted and compared against this value if present. If no `passwd` field is present, then a passwd is not required.

`rights`: This value is used to indicate the rights that the initiator will be allowed. It may be one of the following values:

Value	Meaning
`none`	The association is refused.
`limited`	The initiator is allowed to examine the queue but not to change any parameters.
`full`	The initiator is allowed to examine the queue and modify queue parameters

If it is missing, then limited access is assumed.

For example:

```
anon:rights=none
admin:passwd=YNKJHgaawczz,rights=limited
pp:passwd=YNKcyeOsT6Fqc,rights=full
```

In this case the following is true:

`anon`: Unauthenticated users are not allowed any access to the QMGR.

`admin`: The user admin with the correct password is allowed to view the queue.

`pp`: The user pp, with the correct password, is allowed full access to control the queue manager.

If no table is specified, limited access is allowed for any connection.

4.11 The 822 Local Table

The *ch.local* table 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 delivery mail to registered users. The LHS is the registered users mail address. The format is of the RHS is of the form "key=value." The following table of key/value pairs are allowed:

uid The numeric user id to deliver the message as.

gid The numeric group id to deliver the message as.

username A username 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 exec'd 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 or `unix` for sendmail compatible style (the default is `pp`).

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. This defaults to the global tailor entry. A keyword of `none` disables this facility.

searchpath The directory to look for binaries that the user can run in restricted mode.

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

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

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. (Actually 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 `USRBINDIR` defined in the *Make.defs* file).
- The processes executed by the `pipe` command must be executable by the system call *exec*(2). (E.g. 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:

```
Alina.DaCruz:username=alina home=/cs/users/vs1/alina
John.Taylor:username=jtaylor
Postie.Pat: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>]]
```

4.12 The X.400 (1984/1988) Outbound Table

The outbound X.400 channels use this table to map from an internal PP MtaKey to the connection information for the remote MTA (network address, MTA name(s), and password(s)). An MtaKey in this table is the value representing the MTA derived from the “mtaname” field of the *channel* table. The value is a set of `key=value` fields separated by tabs, spaces, and/or commas. For example:

```
MtaKey:rmta="MtaName",rpass="pass",rpsap='"591"/Janet=000005110000'
```

The `MtaKey` is the key obtained from the address lookup. This is the remote MTA to which the outgoing connection it to be made. The keys that are used for outbound connections are listed below.

These keys are used in both the outbound and inbound X.400 tables:

`lmta`: The name of the local mtaname to pass in the RTS connect parameters.

`lpass`: The passwd of the local site to pass in the RTS connect parameters.

`rmta`: If not defined, the MTA Name supplied by the remote MTA is not verified. If it is defined, the MTA name supplied by the remote MTA must match this value.

`rpass`: If not defined, the password supplied by the remote MTA is not verified. If it is defined, the password supplied by the remote MTA must match this value. This should not be set unless the `rmta` is also set.

`window`: This controls negotiation of the RTS checkpoint window. The value `none` is equivalent to the default. The outgoing channel proposes this value (default `3`) as the maximum number of outstanding minor synchronisation points that may be permitted within an activity. The incoming channel negotiates the minimum of that proposed by the initiator and this value. The minimum valid value is `1`.

`ckpoint`: This controls negotiation of the RTS checkpoint size. The value `none` is equivalent to the default. The outgoing channel proposes this value (default 63) as the maximum size RTS data unit (measured in units of 1024 bytes) between minor synchronisation points. If the initiator proposes `0` then the responder is free to choose any value.

The incoming channel negotiates the minimum of that proposed by the caller and this value, except that if the caller proposes `0` then the incoming channel will negotiate this value. The minimum valid value is `0`.

The following keys are only used for outbound connections:

`rpsap`: The remote presentation address. This is specified in the string encoded format. Note that session selectors and presentation selectors should always be absent in the 1984 RTS. This is the only mandatory key on the RHS of the outbound tables.

`lpsap`: The local presentation address. This is only necessary if you need to originate from a specific selector or network address. Normally this parameter is not specified.

`mode`: The mode of RTS connection. This is either `mon` for monologue (the default) or `twa` for two way alternate. This only controls the mode that is initially requested, as the remote MTA may only accept a one-way connection.

`type`: The type of RTS connection. This is only used with *x400out88*. This is set to one of the following:

Value	Meaning
`1984`	Use the 1984 protocol mode (default).
`1988-X410`	Use the 1988 X.410 mode protocol,
`1988-normal`	Use the 1988 normal mode protocol.
`1988`	Use the 1988 normal mode protocol (same as previous).

`trynext`: A key to another entry in the table. If a connection using the current information fails, the outbound channel will try to connect to the listed MtaKey. The syntax is `trynext="NextMtaKey"`. (*Caution*: if the connection to NextMtaKey succeeds, the *stat* log may show that the message was delivered to the *first* MtaKey specified, not the NextMtaKey.)

`tracing`: The style of tracing information to use. It may take one of the following values:

Value	Meaning
`admd`	Remove all but the first tracing component.
`nointernal`	Remove all internal tracing components.
`local-internal`	Keep only internal trace related to the current PRMD.
`all`	keep all tracing information

The default value is `all`.

The `admd` mode should only be used when talking to a particularly restrictive ADMD. Again, it depends on the capabilities of the channel how closely this information can be followed.

`fix-orig`: This applies an on-the-fly change to the originator address. The syntax is

```
"fix-orig=" <old-or> "->" <new-or>
```

and might be applied as

```
fix-orig="/ADMD= /C=GB/->/ADMD=Gold 400/C=GB/"
```

to change the ADMD from space, to `Gold 400` in the originator for outgoing messages to specific hosts.

The address fix is also done on the P1 MPDU ID.

`other`: Other information. Currently this is not used by the X.400 outbound channels, but may be in the future.

If an entry with a key `default` exists in the table, then this may be used to set default setting for all entries. The default values may be overridden in any entry by redefining the key in that entry. The use of a `default` entry is recommended. The `default` entry is commonly used to define the `lmta`, `lpass`, and `lpsap` values.

An example outbound table is shown in Figure 4.9.

WARNING: the X.400 outbound tables have one entry per line. Long entries cannot be broken across multiple lines. Lines are folded in this example for clarity. Do *not* break lines in your own tables.

```
default:lmta="us",lpass=" ",
germany-nixdorf:rmta="nme",rpsap='Int-X25(80)=26245300042203'
france-derieux:rmta="persona",rpsap='Int-X25(80)=208075200131'
france-reunir:rmta="sunir.reunir.fr",
              rpsap='Int-X25(80)=20803500034996'
pp-nottingham:rmta="pp.cs.nott.ac.uk",
              rpsap='"591"/Janet=000021000018+PID+03010100'
vs2.cs.ucl.ac.uk:rmta="vs2.cs.ucl.ac.uk",
              rpsap='"591"/Internet=vs2.cs.ucl.ac.uk'
```

Figure 4.9: Example of X.400 Outbound Channel Table

For detailed information on establishing X.400 connections, see section 5.3.

4.13 The X.400 (1984/1988) Inbound Table

The inbound X.400 channels use the table to map from the OSI calling address to the local PP MtaKey and associated connection parameters (MTA name(s) and password(s)). The syntax has the same general format as the X.400 outbound table.

However, the key on the LHS of the table is the calling address given in the session (1984) or presentation (1988) connection from the remote MTA. These will look much like the rpsap values in the X.400 outbound tables. However, no macro substitution may be included.

Without macro substitution, the keys look pretty horrible. It is typical to maintain these tables with keys in macro form and run them through the PP utility *nsapexpand* to expand the macros. For example, you might store the macro version of the *ch.x400in84* table in *ch.x400in84.mac*.

The primary security checking of incoming connections is achieved by comparing the incoming address with the entries in the table.

Like the X.400 outbound tables, the RHS of each entry is a sequence of `key=value` pairs separated by spaces, tabs, and/or commas. The `lmta`, `rmta`, `lpass`, and `rpass` keys have the same function described above in the X.400 outbound channel.

The following keys are only used in the inbound X.400 tables:

`rname`: The "real" name of the remote MTA. This is an entry that is suitable for lookup in the channel table. Typically, this is the same MtaKey value used to describe the remote MTA in the inbound X.400, outbound X.400, *channel*, and *or* tables.

`other`: Other information. This is a string and is interpreted by the channel to allow other types of operation to be specified. It is used by the X.400 inbound channels as follows:

If `other` is set to `sloppy`, the inbound channel will not check against any `rpass` specified in the inbound table. (Note that not verifying passwords can also be achieved by simply not specifying any `rpass` values in the table.)

If it contains `undefined=`*type* then submission of the Encoded Information Type (EIT) "undefined" will be mapped to *type*.

Again, a `default` entry in the table may be used to default many of the fields. Defaults may be overridden as in the X.400 outbound tables.

An example inbound table is shown in Figure 4.10

WARNING: the X.400 inbound tables have one entry per line. Long entries cannot be broken across multiple lines. Lines are folded in this example for clarity. Do *not* break lines in your own tables.

```
default:lmta="us",lpass="bar"
"3459"/Int-X25(80)=208006030203:rname="france-kwai",
                rmta="kwai.inria.fr"
"425"/Int-X25(80)=26245622190113:rname="germany-ibm",
                rmta="foo",rpass="foo"
Janet=000021000018:rname="pp-nottingham",rmta="cs.nott.ac.uk"
Janet=000005111680:rname="pp-vs2",rmta="cs.ucl.ac.uk"
```

Figure 4.10: Example of X.400 inbound Channel table

For detailed information on establishing X.400 connections, see section 5.3.

4.14 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 that 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* (see Volume III (The PP Manual: User's Guide)). Anyone with one of the specified uids 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 that 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 that 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 that 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*.

```
#
# extract of 'list' table
#
#Comment: -*- Example lists -*-
filelist:pac,file=/crg/pp/mainlist,a list with members in a file
strlist:pp|pac,p.cowen|j.onions|g.lunt,a list with members in line
mixedlist:jpo,j.onions|file=mixedlist,another list
#
```

Figure 4.11: Example of List Table

If you have a large sendmail alias 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 of PP. Note: it does *not* convert all alias table entries, only lists.

4.15 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 that 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.

`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 in Table 4.2.

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 (x400 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 (x400 format)
qid	the queue id of the message (e.g., msg.a000000)
ua-id	the user agent id of the message
p1-id	the P1 id of the message
userid	The username or 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

Table 4.2: Shell Channel Expansion Macros

When running, the *shell* channel pipes all the message body parts, in order, to the program's standard input. The *shell* channel understands an exit code of 0 as success and otherwise 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 in Figure 4.12.

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

Figure 4.12: Example of Shell Table

4.16 The UUCP Table

The *ch.uucp* table is used by the *uucp* outbound channel to convert from an RFC 822 style address to an UUCP style address. This conversion is done for all “envelope” addresses. Entries in this table have the format:

```
<822-domain> ":" <uucp-mapping>
```

`822-domain`: specifies the next hop 822 domain.

`uucp-mapping`: specifies the mapping of the 822 domain into a UUCP address. In this mapping the string `%s` is expanded to the `822-address` thus completing the conversion.

An example segment of a *uucp* table is shown in Figure 4.13.

```
#################### extract of `uucp' table ####################
#
icocae.uucp:iconet!icocae!%s
icodap.uucp:iconet!icodap!%s
trzfal.uucp:iconet!trzfal!%s
icogre.uucp:iconet!icogre!%s
barhor.uucp:iconet!barhor!%s
#
```

Figure 4.13: Example of UUCP Table

4.17 The P3file Table

The *ch.p3file* table is used by the *P3file* outbound channel for message delivery. For each each O/R Address Personal Name value in the table, the table table identifies which Unix userid and directory should be used to deliver the message. Entries in the table have the format:

```
<First.Last> ": username=" <login> ", delivery_queue=" <directory>
```

`<First.Last>`: is the local part of an O/R Name. The *or* table should resolve down to this. The conversion of local parts of O/R Names to strings is `<Given>.<Surname>` or `<Given>.<Initials>.<Surname>` (this is standard PP behaviour—see RFC1327). Note that this name should appear in the users table with p3file specified as the local delivery channel.

`<login>`: Unix username to deliver under

`<directory>`: Directory to place outgong P3 files under

The p3file table is also used by the p3filetopp daemon when messages are submitted.

Note—if Enterprise Mail is being used, the *em.ini* file should be configured to match the above values.

An example segment of a *P3file* table is shown in Figure 4.14.

```
#################### extract of `p3file' table ###################
#
Roger.James: username=roger, delivery_queue=/users/roger/dq
Wodger.Wabbit: username=ww, delivery_queue=/users/ww/dq
Polly.Programmer: username=polly, delivery_queue=/users/polly/dq
Dan.Different: username=dan, delivery_queue=/users/polly/mailq
#
```

Figure 4.14: Example of P3file Table

4.18 Statistical Logging

Each line in the statistics log ("stat") either logs the status of inbound messages, or delivery reports that are generated, or notes a message that is delivered. Amongst the parameters logged are:

- A success indicator (e.g., "ok" or "DR").
- Unique message id.
- A P1 message ID.
- Inbound channel→outbound channel.
- Sender.
- Sending MTA.
- Recipient.
- Destination MTA.
- Message size.
- A diagnostic message.

Fields are space-delimited and double quoted where necessary.

Duplicate recipients are eliminated (second and subsequent) if there is an exact match on the 822 or X.400 address (as appropriate), *not* the original entry.

Channel binding is performed after all other tests have been passed, and takes into account any body parts actually received but not specified in the message header.

Unsuccessful recipients are flagged to generate nondelivery reports.

The tool *statp*, found under the *Tools* directory, takes the statistics log and converts it to something suitable for processing via `awk` or other such processing tools. There are several example awk scripts under the source directory *Tools/statgen*.

Chapter 5

Configuring Your System

5.1 Managing Tables

As described in Section 4, there are a number of tables that PP requires. For a site of any size, the generation of these tables is likely to be substantially automated. Manual generation will probably only be used by very small sites. Local tables, such as *aliases*, *users*, *local channel*, *list channel* or *authorisation* files will come from local sources. The system manager will be able to generate these easily. In the longer term, support for local information by use of the OSI Directory is to be expected.

For now, the tables needed for X.400 routing (the *or* table and X.400 channel tables) must be manually generated. This is an indication of the immaturity of this technology.

RFC 1327 Mapping Tables

RFC 1327 mappings should be global. Therefore, it is desirable to use standard tables as far as possible. Two approaches are possible:

1. UCL (and later the UK Gateway service at ULCC) will maintain and make available RFC 1327 tables in PP format. FTAM will probably be the preferred mode of access.
2. There are scripts to take the RARE WG1 format tables and generate PP format tables from these. The ambiguous and dubious mappings are avoided by the simple expedient of ignoring all mappings which relate to a single domain component.

The first method is preferred, until the RARE tables are updated at a more frequent interval (preferably on a continuous basis, which can be pulled each day).

Domain and Channel Tables

There are some support tools to help generate the *channel* and *domain* tables, for domain-based information. Note that the *channel* table is used for both X.400 and domain bindings.

For *domain* and *channel* tables, it is important to understand the use of these when defining mechanisms to build them. The steps that are taken for resolving a domain address are as follows:

1. Look up the domain of the address in the *domain* table. Note:
   - Domains are in US order.
   - There is a qualification on whether a given domain matches a specific entry. This qualification is based on the number of subdomains permitted for that entry.
2. Look up the domain name derived from the domain table in channel table. The RHS of the channel table gives a list of host/MTA pairs which may be used. If the MTA is null, the originally specified MTA will be used.

Two basic strategies for generating these tables are described.

Internet: This assumes that SMTP is used, in the context of domain nameservers. The tables are used to identify a part of the domain tree which will be passed to the SMTP channel. Use of DNS is confined to this channel.

NRS: Here, UK NRS tables are used to point valid NRS domains at JNT Mail channels.

Simple sites will be able to use one of these strategies directly. More complex sites will need to add additional information.

Generation of the domain table

When building a domain table, it is useful to start by building a variant which only has entries for fully qualified domain names and the partial domain names that are known. The *abbrev* tool can be used to create the required entries for abbreviated forms of domain names.

The domain table is compiled from one or more sources:

Local Name: Your own domain name, and alternates are needed. For example:

```
cambridge.ac.uk:mta
cam.ac.uk:norm+mta=cambridge.ac.uk
```

NRS data: The NRS data can be converted directly into the PP domain format by use of the *c-nrs* program[1]. The file *pp.dmn* created by this program should then be concatenated with the other generated domain data before being processed by the *abbrev* tool.

Internet: A list of top level domains which you wish to be passed to the SMTP/DNS channels should be given. For example:

```
*.edu:mta=edu
*.gov:mta=gov
*.com:mta
*.mil:mta
```

Note that to match with partial entries prefixed by `*.`, unless altered by the `min` key, there must be at least one subdomain. Thus preventing `edu`, `gov`, `com` and `mil` being recognised as valid domains by themselves.

Other known domains: Other known domains (e.g., those associated with local UUCP links) should be added in. For example:

[1] To obtain a copy of this, contact Piete Brooks at Cambridge Computer Laboratory (pb@cl.cam.ac.uk)

```
iconet.uucp:mta
inset.uucp:mta
```

Once all the data has been collected, including all your local hosts and domains, it should be passed through the *abbrev* program. This arranges for all reasonable abbreviations of domains to be added. An example input might be the following:

```
cs.nott.ac.uk:mta
cs.ucl.ac.uk:mta
stc.co.uk:mta
widget.uucp:mta
```

The syntax of the *abbrev* command is `abbrev domainname...` and it reads from the standard input and writes to the standard output. One domain name should be the fully formed local domain. Other arguments can be used to generate abbreviations for other domains. When given the above as input and the command

```
abbrev cs.nott.ac.uk uucp
```

the output produced is

```
cs.nott.ac.uk:mta
cs.nott.ac:norm+mta=cs.nott.ac.uk
cs.nott:norm+mta=cs.nott.ac.uk
cs:norm+mta=cs.nott.ac.uk
cs.ucl.ac.uk:mta
cs.ucl.ac:norm+mta=cs.ucl.ac.uk
cs.ucl:norm+mta=cs.ucl.ac.uk
stc.co.uk:mta
stc.co:norm+mta=stc.co.uk
widget.uucp:mta
widget:norm+mta=widget.uucp
```

Generation of the Channel Table

The channel table is also generated in two stages. The input is as follows.

Local domain: This is not needed.

NRS data: The file *pp.chan* generated by the *c-nrs* program is used directly.

Internet: The list of valid domains is given, with the channel indicated. For example:

```
edu:(smtp)
gov:(smtp)
com:(smtp)
mil:(smtp)
```

Private links: These should be added manually. For example:

```
widget.uucp:widget.uucp(uucp)
```

Note that the key “widget.uucp” must be the same on each side of the table.

Application relays: Known domains, reached indirectly, can be added. For example:

```
foobar.uucp:widget.uucp()
```

This says that foobar.uucp is reached indirectly through widget.uucp.

In the second stage, the program *tjoin* is run to collate and reformat these input data into their final form. This stage is essential if the input contains application or top-level relays, and performs the following functions:

- "Unrolling" application relays, so that the installed tables specify the correct first hop.
- Where an MTA can be reached by multiple channels, sorting these into the preferred order. A direct connection will always be preferred to an application relay.

Tjoin input allows the following syntaxes:

mta:mta(channel): Specifies to contact `mta`, call `mta` using channel `channel`. Both `mta` parameters should be the same

mta:mta1(): To contact `mta`, call `mta1` using the channel associated with `mta1` which should be specified elsewhere in the *channel* table. This is the application relay format.

mta:(channel): To contact `mta` call `mta` using channel `channel`. This form is appropriate for domain nameservers.

An example of the functions that *tjoin* does is shown here. The input of

```
stc.co.uk:(smtp)
uucp:uucphost.stl.stc.co.uk()
ac.uk:psshost.stl.stc.co.uk()
win.icl.stc.co.uk:erbert.uucp()
erbert.uucp:erbert.uucp(uucp)
psshost.stl.stc.co.uk:psshost.stl.stc.co.uk(smtp)
uucphost.stl.stc.co.uk:uucphost.stl.stc.co.uk(smtp)
```

would give as final output the table:

```
ac.uk:psshost.stl.stc.co.uk(smtp)
erbert.uucp:erbert.uucp(uucp)
psshost.stl.stc.co.uk:psshost.stl.stc.co.uk(smtp)
stc.co.uk:(smtp)
uucp:uucphost.stl.stc.co.uk(smtp)
uucphost.stl.stc.co.uk:uucphost.stl.stc.co.uk(smtp)
win.icl.stc.co.uk:erbert.uucp(uucp)
```

5.2 Adding New Users

To add new users to the tables is best done in an automated way, generating the list from some master table perhaps or a system database.

To add a user, for example “J.Soap” with login id “joe” using the local channel, the following is required:

1. Add an entry to the *alias* table to map “joe” into “J.Soap.”

   ```
   joe:synonym J.Soap
   ```

   This step is not required if you do not want to refer to Joe other than by the UNIX login id.

2. Add an entry to the *users* table to specify which channel to deliver Joe’s mail by.

   ```
   J.Soap:822-local
   ```

   If you run a number of MTAs locally, then add the machine name after the channel.

3. Add an entry to the local channel table to give the delivery details.

   ```
   J.Soap:joe
   ```

   Optionally add the home directory and the mailbox if required.

4. To make the changes have immediate effect, run *dbmbuild*. It is not necessary to restart any of the daemon programs.

5.3 X.400 connections

This section discusses how to go about setting up X.400 connections with other sites using table based configuration. For configuration by use of the X.500 directory see Appendix A. This section describes how to set up the *or* routing tables, and the other tables needed for bilaterally agreed linkage.

To set up an X.400 connection, information about the remote site needs to be obtained and an entry needs to be inserted in the PP tables.

5.3.1 Remote Site Information

Obtain the information below prior to setting up an X.400 link.

	Technical Contact:	The person you contact if things go wrong.
	Telephone:	Telephone number of the technical contact.
*	O/R Address:	Address of a contact person.
*	MTA Name:	Name by which the remote site identifies itself.
*	MTA Password:	Password by which the remote site identifies itself.
*	Transport Selector:	A port identifier at the transport level.
	Network Address:	The X.25 DTE and/or IP address of the remote site.
*	X.25 Protocol ID:	A port identifier at the X.25 level.
*	ManagementDomains:	Any management domain/s (MD) that the remote site provides a connection to.

Any information prefixed with an “*” may be optional.

Examples of the format are shown in Tables 5.1 and 5.2.

Technical Contact:	Peter Yee
Telephone:	not known
O/R Address:	C=US; ADMD=TELEMAIL; PRMD=ARC; O=NASA; S=yee;
MTA Name:	titan.arc.nasa.gov
MTA Password:	" " (i.e a one character space)
Transport Selector:	591 (ascii)
TCP IP Address:	"128.102.128.5"
X.25 DTE Address:	not known
X.25 Protocol ID:	not known
ManagementDomains:	C=US;ADMD=TELEMAIL; PRMD=ARC;

Table 5.1: Example of X.400 Information for NASA

Technical Contact:	John Andrews
Telephone:	01-387-7050 ext 3691
O/R Address:	C=GB; ADMD=GOLD 400; PRMD=UK.AC; O=UCL; OU=UCL; S=Postmaster;
MTA Name:	UK.AC.UCL.CS
MTA Password:	" " (i.e a one character space)
Transport Selector:	591 (ascii)
TCP IP Address:	128.16.5.31
X.25 DTE Address:	23421920030080
X.25 Protocol ID:	not needed
ManagementDomains:	C=GB; ADMD=GOLD 400; PRMD=UK.AC C=GB; ADMD=BTMHS; PRMD=UK.AC C=GB; ADMD= ; PRMD=UK.AC

Table 5.2: Example of X.400 Information for UCL

Conventions

These are some of the conventions that PP uses for X.400 links. These need not be followed strictly but make a good starting point.

- ISODE uses a transport selector. This should be ASCII-encoded and should be all numeric (to conform to X.400(84)). The value “591” is often used.
- The MTA name is usually an appropriate domain. In the UK this is in NRS (i.e., big endian domain) order.
- The MTA password is usually a single space.

5.3.2 Routing and Mapping tables

First, a number of mapping tables need to be installed. If you do not have a copy of them, you will need to get one. Copies should be obtained from University College London (see Appendix E for details).

`or`: The O/R Table (See Figure 5.1 below and Section 4.5).

`rfc2or`: The RFC 822 to O/R Table (See Figure 4.7) .

`or2rfc`: The O/R to RFC 822 Table (See Figure 4.6).

`rfc1148gate`: The RFC 1148 known gateways file.

The *rfc2or*, *or2rfc* and *rfc1148* are the RFC 1327 mapping tables which should be placed in the Table directory.

5.3.3 Editing the O/R Table

Replace the `_DEFAULT_MTA_` occurrences with appropriate remote MtaKeys. These are values used internally inside PP to identify the remote MTA. The choice of MtaKeys is up to the PP administrator, and is used for identification purposes only.

Local information should be prepended to this table, and will override later entries.

When observing the example O/R tables in Figures 5.1 and 5.2 please note that:

- The O/R skeleton table in Figure 5.1 on page 84 is a truncated example of the *retrieved* UCL O/R master table.
- The O/R skeleton table in Figure 5.2 on page 85, is a truncated example of an *edited* UCL O/R master table.

5.3.4 X.400 Outbound Procedures

A brief description of how the X.400 outbound table is used within PP follows:

- PP connects to a remote MTA using the address specified in the `rpsap` field. These address are tried in the order defined in the `ts_communities` ISODE variable. If all addresses fail, and the entry in the x.400 outbound table contains a `trynext` entry, that MTA will be tried too.
- When PP is sending the connect request, includes its local MTA name and password as identification.
- The remote site *either* accepts the connection and returns its own MTA name and password *or* it rejects the connection and gives its reason as “validation failure”.
- If the remote site rejects the connection, PP terminates the X.400 connection.
- If the remote site has accepted the connection, PP matches the returned MTA name and password against the values held for the remote site in the X.400 outbound table (i.e the `rmta` and `rpass` fields).
- If the remote site’s MTA name and password match, then the connection has been set up, and PP starts to transmit a message; otherwise PP terminates the X.400 connection.

```
# A Truncated example of the UCL O/R Master Table
#
#       -*** United Kingdom ***-
#
C$GB:valid
C$UK:alias C$GB
C$US:valid
ADMD$BTMHS.C$GB:valid
ADMD$ .C$GB:valid
ADMD$GOLD 400.C$GB:mta _DEFAULT_MTA_
ADMD$GOLD-400.C$GB:alias ADMD$GOLD\ 400.C$GB
ADMD$ATTMAIL.C$US:mta _DEFAULT_MTA_
PRMD$UK\.AC.ADMD$GOLD 400.C$GB:valid
PRMD$AC\.UK.ADMD$GOLD 400.C$GB:alias PRMD$UK\.AC.ADMD$GOLD\ 400.C$GB
PRMD$UK\.AC.ADMD$BTMHS.C$GB:alias PRMD$UK\.AC.ADMD$GOLD\ 400.C$GB
PRMD$UK\.CO.ADMD$GOLD 400.C$GB:valid
PRMD$BDAC.ADMD$ .C$GB:mta _DEFAULT_MTA_
PRMD$DTIIED.ADMD$GOLD 400.C$GB:mta _DEFAULT_MTA_
O$UCL.PRMD$UK\.AC.ADMD$GOLD 400.C$GB:mta _DEFAULT_MTA_
#
#       -*** australian sites ***-
C$AU:valid
ADMD$OTCDIA.C$AU:valid
ADMD$OTC.C$AU:valid
PRMD$EDU.ADMD$OTCDIA.C$AU:mta _DEFAULT_MTA_
PRMD$DITMELPP.ADMD$OTC.C$AU:mta _DEFAULT_MTA_
#
#       -*** french sites ***-
C$FR:valid
ADMD$PTT.C$FR:mta _DEFAULT_MTA_
ADMD$ATLAS.C$FR:mta _DEFAULT_MTA_
PRMD$REUNIR.ADMD$ATLAS.C$FR:mta _DEFAULT_MTA_
#
# -*** usa sites ***-
ADMD$UM.C$US:valid
ADMD$ATT.C$US:valid
PRMD$NREN.ADMD$UM.C$US:mta _DEFAULT_MTA_
PRMD$ARC.ADMD$TELEMAIL.C$US:mta _DEFAULT_MTA_
PRMD$WOLLONGONG.ADMD$ATT.C$US:mta _DEFAULT_MTA_
```

Figure 5.1: Example of an O/R Table to Edit

5.3.5 Editing the X.400 Outbound Table

Insert a default entry in the X.400 outbound table if it does not exist, where the key has the value "default" with the following fields set. These are the default values by which the the local PP identifies itself to a remote site when an outgoing X.400 connection is made. Refer to the X.400 outbound table in Figure 5.3 when doing this.

`lmta`: Local MTA name.

`lpass`: Local password.

`lpsap`: Local network address (same format as `rpsap`).

When the O/R table was previously edited, the remote site names that were chosen by the PP administrator should have been carefully recorded. These are the names that are now used to "key" into the X.400 outbound table. In addition to this, the information below is also required (refer to Figure 5.3.1).

```
#   A Truncated example of a
#   UCL O/R Master Table after being edited by
#   a PP Administrator
#
#       -*** United Kingdom ***-
#
C$GB:valid
C$UK:alias C$GB
C$US:valid
ADMD$BTMHS.C$GB:valid
ADMD$ .C$GB:valid
ADMD$GOLD 400.C$GB:mta gb-ucl
ADMD$GOLD-400.C$GB:alias ADMD$GOLD\ 400.C$GB
ADMD$ATTMAIL.C$US:mta gb-ucl
PRMD$UK\.AC.ADMD$GOLD 400.C$GB:valid
PRMD$AC\.UK.ADMD$GOLD 400.C$GB:alias PRMD$UK\.AC.ADMD$GOLD\ 400.C$GB
PRMD$UK\.AC.ADMD$BTMHS.C$GB:alias PRMD$UK\.AC.ADMD$GOLD\ 400.C$GB
PRMD$UK\.CO.ADMD$GOLD 400.C$GB:valid
PRMD$BDAC.ADMD$ .C$GB:mta gb-ucl
PRMD$DTIIED.ADMD$GOLD 400.C$GB:mta gb-ucl
O$UCL.PRMD$UK\.AC.ADMD$GOLD 400.C$GB:mta gb-ucl
#
#       -*** australian sites ***-
C$AU:valid
ADMD$OTCDIA.C$AU:valid
ADMD$OTC.C$AU:valid
PRMD$EDU.ADMD$OTCDIA.C$AU:mta au-goanna
PRMD$DITMELPP.ADMD$OTC.C$AU:mta au-manta
#
#       -*** french sites ***-
C$FR:valid
ADMD$PTT.C$FR:mta fr-kwai
ADMD$ATLAS.C$FR:mta fr-kwai
PRMD$REUNIR.ADMD$ATLAS.C$FR:mta fr-reunir
#
# -*** usa sites ***-
ADMD$UM.C$US:valid
ADMD$ATT.C$US:valid
PRMD$NREN.ADMD$UM.C$US:mta us-destroyer
PRMD$ARC.ADMD$TELEMAIL.C$US:mta us-nasa
PRMD$WOLLONGONG.ADMD$ATT.C$US:mta us-wollongong
```

Figure 5.2: O/R Table after Editing

- MTA name and password (both optional)
- Network address (mandatory)
- Transport selector (optional)
- Protocol ID or PID (optional).

Insert an entry in the X.400 outbound table, where the remote site information maps into the following fields. Refer to the X.400 outbound table example in Figure 5.3 when doing this.

- remote site name - `key`.
- MTA name - `rmta` (if not specified, the remote MTA name will always be accepted).
- Password - `rpass` (if not specified, the remote password will always be accepted).

- Network address, TSAP and PID - `rpsap`.

NOTE

- All keys in the X.400 outbound table must be unique.
- If the `rmta` and `rpass` are not specified, this means that the remote site is using *either* some other method of validation *or* no validation at all.
- The `rpsap` field has a number of different types of formats, some of which may be seen in Figure 5.3. The formats are described in [2].

The X.400 outbound table below gives the NASA and UCL entries using the remote site information specified in Figures 5.1 and 5.2. Some possible `rpsap` formats are also shown.

5.3.6 Edit the Channel Table

The channel table below gives the NASA and UCL entries using the remote site information specified in the tables in Figure 5.1.

- Note that the key to this table has the same value as that in the X.400 outbound table in Figure 5.3.
- Also, note that the name specified in the rounded brackets is the name of the X.400 outbound channel, and possesses the same cost value as that specified in the *tailor* file.

5.3.7 X.400 Inbound Procedures

A brief description of how the X.400 inbound table is used within PP.

- A remote MTA requests a connection with the local PP system, and within the request its remote network address, MTA name and password are specified.
- The local PP uses the remote network address as a key into the X.400 inbound table.
- If the key is invalid, then the local PP sends a rejection back to the remote MTA, giving “validation failure” as the reason (unless running with `ininfo=sloppy`, see below).
- If the key is valid, then the remote MTA’s MTA name and password are checked against its *rmta* and *rpass* values specified within the X.400 inbound table. (If the *rmta* and *rpass* values are not specified in the X.400 inbound table, they are always accepted.)
- If the *rmta* and *rpass* values are not specified in the X.400 inbound table, *or* if they are specified and they match the values included in the connection request, then the local PP accepts the request by returning its MTA name and password, and begins to get ready for remote message transmission.
- If the values do not match, then the local PP sends a rejection back to the remote MTA, giving “validation failure” as the reason.
- The remote MTA compares the MTA name and password that this system returned to the values it was expecting. If the values do not match, then it terminates the connection.

```
#-*************************************
#
#
# -*** UCL's default local entry ***-
# -*** this is different for every PP site ***-
# (lines folded for readability)
default:lpsap='"591"/Internet=bells',lmta="UK.AC.UCL.CS",lpass=" "
#
#
-*** NASA ***-
#
us-nasa:rmta="titan.arc.nasa.gov",
  rpsap='"591"/Internet=128.102.128.5'
#
# -*** UCL ***-
gb-ucl:rmta="UK.AC.UCL.CS",rpass=" ",
  rpsap='"591"/X121+23421920030080'
#
# -*** when pid is specified, the rpsap format below is used ***-
#
eg_site1:rmta="SiteName1",
  rpsap='"999"/Int-X25(80)=23421234005004+PID+03010100'
#
# -*** when more than one network address is specified ***-
# -*** note the network addresses are seperated by '|' ***-
#
eg_site2:rmta="SiteName2",
  rpsap='"400"/X121+23427351714799|X121+23428490013799'
#
# -*** when trynext is specified, the format below is used ***-
#
eg_site3:rmta="SiteName3",rpsap='"999"/X121+234212340050',
  trynext="eg_site2"
#
#
au-manta:rmta="manta.cng.dit.csiro.au",rpass=" ",
  rpsap='"591"/Int-X25(80)=50523343000180+PID+03010100|Internet=128.250.1.100'
#
#
#
#-*************************************
```

Figure 5.3: Example of X.400 Outbound Table

```
#-***************************
#
#       -*** nasa ***-
us-nasa:us-nasa(x400out)
#
#       -*** ucl ***-
gb-ucl:gb-ucl(x400out)
#
#-***************************
```

Figure 5.4: Example of Channel Table

NOTE: 2 additional fields may be set to facilitate X.400 inbound connection testing.

`ininfo=sloppy`**:** If this is set in the *tailor* file, allows the acceptance of all incoming connections. All validation checks are bypassed.

`other=sloppy`**:** set in the X.400 inbound table against a particular remote site entry bypasses the validation checks for that site's remote MTA name and password. (Note that this can also be accomplished by not specifying the *rmta* and *rpass* in the entry for that MTA.)

5.3.8 Editing the X.400 Inbound Table

Before editing this table the PP administrator needs to:

- Decide whether or not to be selective over the acceptance of remote site connections. If *all* remote site connections are to be accepted, then in the *tailor* file the X.400 inbound channel is tailored with `ininfo=sloppy`, and *no* entries need to be registered in this table. If entries are registered, they will be bypassed.
- Decide whether or not remote site MTA name and password validation checks are to be done. This decision should only be taken if `ininfo=sloppy` is not set in the *tailor* file for the X.400 inbound channel. If no MTA name and password validation checks are to be carried out, then for that remote entry in the X.400 inbound table, set `other=sloppy` or do not specify *rmta* and *rpass* in the entry.

Please note when taking these decisions:

- If `ininfo=sloppy` is set within the *tailor* file for the X.400 inbound channel, the network address of a remote site is still used as a key into the X.400 inbound table to obtain its remote site name (`rname`). If the key is invalid, then the remote site name is set to its network address. This will cause problems, if delivery notifications are requested by such a site, as the local PP would be unable to use the remote site name to key into the X.400 outbound table. Therefore `ininfo=sloppy` should be set for testing purposes only.
- That for each remote site an X.400 inbound and outbound entry should exist. Otherwise if `ininfo=sloppy` is set, and a remote site's network address keys successfully into the X.400 inbound table, unless there is a corresponding entry in the X.400 outbound table, no mail will be able to be sent to that site.

When the X.400 outbound table was previously edited, the following remote site information should have been obtained:

- remote site name (chosen by the PP administrator).
- MTA name and password (both optional).
- Network address (mandatory).
- Transport selector (optional) .

As the X.400 inbound table is very similar to the X.400 outbound table the same information is again required, only this time, instead of making a connection to a remote site, the concern is centered on a remote site requesting a local connection.

The fields in the X.400 inbound table (see Figure 5.5) are very similar to the ones in the X.400 outbound table (see Figure 5.3), and should be set accordingly, except for two main differences:

`rpsap`: acts as the table entry key.

`rname`: is an extra parameter which specifies the remote site MtaKey. This should have the same value as the remote site `key` in the X.400 outbound table.

The X.400 inbound table below gives the NASA and UCL entries using the remote site information specified in Figure 5.1.

```
#-**********************
#
#
#       -*** UCL's default local entry ***-
#       -*** this is different for every PP site   ***-
#
# (Folded for readabilty)
#
default:lpsap='"591"/Internet=bells',
     lmta="UK.AC.UCL.CS",lpass=" "
#
#       -*** NASA ***-
#
"591"/Internet=titan.arc.nasa.gov:rname="us-nasa",
    rmta="titan.arc.nasa.gov",rpass=" "
"591"/Internet=128.102.128.5:rname="us-nasa",
    rmta="titan.arc.nasa.gov",rpass=" "
#
# -*** UCL ***-
#
"591"/X121+23421920030080:rname="ucl",
   rmta="UK.AC.UCL.CS",rpass=" "
#
#
#-**********************
```

Figure 5.5: Example of X.400 Inbound Table

In Figure 5.5 above, it can be seen that:

- The PID field is *never* used in the `rpsap` key.
- If a remote site has a number of network addresses, all may be specified in the `rpsap` of the remote site's X.400 outbound entry (see Figure 5.3) However, there will have to be `rpsap` entries for each network address belonging to that remote site. All other details (`rname`, `rmta`, `rpass`) would be the same.

Detailed logs of *x400in84*, *x400out84*, *x400in88*, and *x400out88* connection attempts are included in Appendix G. The logs include detailed examples of a variety of failure modes.

5.3.9 Testing Outbound Connections

- Go to the PP log directory.
- Look at the X.400 incoming log. The log should look like those in Figure 5.6.

```
2/ 7 15:41:38 x400in   22798 (#52     )  Starting x400in (via X400(84) inbound)
2/ 7 15:41:41 x400in   22798 (#52     )  Connection from
key='X121+5052334300018'   mta='manta.cng.dit.csiro.au'  password=' '
2/ 7 15:41:41 x400in   22798 (#52     )  No information found in the
X.400 incoming tables
2/ 7 16:41:41 x400in   22799 (#52     )  x400in/main error: rts_decode_request
2/ 7 16:41:38 x400in   22799 (#52     )  Starting x400in (via X400(84) inbound)
2/ 7 16:41:41 x400in   22799 (#52     )  Connection from
key='"PHL"/X121+302049600717'   mta='foo'  password='foobar'
2/ 7 16:41:41 x400in   22799 (#52     )  No information found in the
X.400 incoming tables
2/ 7 16:41:41 x400in   22799 (#52     )  x400in/main error: rts_decode_request
```

Figure 5.6: Example of X.400 Incoming Logs

From this log, the following remote site information may be obtained, which can then be inserted as two entries in the X.400 inbound and outbound tables.

- The TSAP: None for the first entry and “PHL” for the second.
- The Network Address: 5052334300018 for the first entry and 302049600717 for the second.
- The MTA Name: “manta.cng.dit.csiro.au” for the first entry and “foo” for the second.
- The password - “ ” for the first and “foobar” for the second.

However, note that:

- A remote site does not have to be specified in the X.400 inbound table for a connection to be successful. If in the *tailor* file the X.400 Inbound Channel is tailored with `ininfo=sloppy`, then all remote connections are accepted.
- If on the other hand MTA name and password validation is to be ignored, then for that remote entry in the X.400 inbound table, `other=sloppy` is added (or do not define *rmta* and *rpass* in the entry).

Detailed logs of *x400in84*, *x400out84*, *x400in88*, and *x400out88* connection attempts are included in Appendix G. The logs include detailed examples of a variety of failure modes.

5.3.10 How to make a test X.400 connection to a remote site

- Go to the table directory.
- Put the remote site information in the appropriate tables.
- Use the channel in “ping” mode to make an RTS connection to the remote site. This is done by running the channel binary as follows:

```
x400out88 ping –m mta –c channel
```

The channel will most likely be x400out88 unless you are using a more complex local configuration. The MtaKey is the `MtaKey` value from the outbound X.400(88) file. For example,

```
x400out88 ping –m MtaKey –c x400out88
```

Similarly for *x400out84*.

- Check the PP logs. The failure/success will be recorded there.
- The following failure reasons may be given:
  - "Validation failure": The remote site has either got an erroneous PP entry in their tables, or they haven't recorded your entry at all.
  - "Unacceptable dialogue mode": Either the remote site does not support monologue or the RTS ASN.1 encoding is bust.
  - "Connect request refused on this network connection": remote site is either down or its network listener is not running.
  - "Busy": remote site is busy, try again later.
  - "Protocol Error": protocol problems.
  - "Remote system problem": remote site's process has terminated abruptly.
  - "Timer expired": remote system's process is looping.
- If the "ping" was successful, then try transmitting a proper message, using the 'mail' program. 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 in the PP *tailor* file. This can be done by adding the following 3 entries:

  isodelog rtsaplevel file=rts level=pdus

  isodelog ssaplevel file=session level=pdus

  isodelog tsaplevel file=transport level=pdus

5.4 Adding Filters, Shapers + EITs

These are advanced issues of PP configuration.

5.4.1 Adding A New Encoded Information Type

If you wish to add a new encoded information type, then some editing of the *tailor* file is necessary.

Suppose we wish to add a new body type "binary" which we want to be able to submit and have converted to `ia5` before transmission.

1. Add a new body type to the list of body types in the *tailor* file.

```
bodypart    binary
```

2. Now specify which outbound channels can handle such a body part by adding this type to the `bptout` list associated with the channel. If no outbound channel will be delivering this type then this step is unnecessary. This is the case for `binary`.

3. To allow *submit* to work out how to convert from `binary` to `ia5` it is necessary to add a new reformatting channel, such as the following:

   ```
   chan bin2ia5   prog=fcontrol, type=shaper, bptin=binary,
                  bptout=ia5, outinfo="uuencode binary",
                  show="Binary -> Text"
   ```

   This specifies that to convert input bodypart type `binary`, to output body type `ia5` the channel *fcontrol* should run the process `uuencode program` with standard input from the `binary` body part and standard output into the `ia5` body part.

   In practise, it might be better not to call *uuencode* directly, but wrap it up in a shell script; we could then print out some preliminary remarks instructing the recipient how to unwrap the binary part.

Now, when *submit* receives a message which has flagged that it contains a body part of `binary`, it will be able to convert that part into `ia5`, which most output channels can deal with.

5.4.2 Adding a New Content Type

Adding a new content type for PP to relay is similar in style to adding a new encoded information type. The steps that need to be taken are dependent on whether conversion is involved or not.

If no conversion is involved, then to add a new content type such as `p22-encrypted`, for instance, would simply consist of registering that content type with an outbound channel. This is done by adding a new channel with the `content-out` part set to `p22-encrypted`. In this case, *submit* will pass the message straight through, unaltered, providing the recipient is reachable via that channel.

If conversion is required, then two things are necessary. The first is to add a new unflattening channel which is capable of splitting the message into component parts. Secondly, the component parts must be able to be reformatted into whatever is required on the outbound channel.

For example, suppose we have a new message content called "whizbang". This consists of a "whizbang" specific header and a standard `ia5` body part. To allow the type of mail to be input, the following would be required.

- An exploder channel that can split up "whizbang" format mail into its components.

  ```
  chan WBexplode    prog=WBexplode,type=shaper,
                    content-in=whizbang,
  ```

  This channel is defined to split up the message into two components, a file `hdr.wb` and a body part `1.ia5`.

- The new encoded information type `hdr.wb` needs to be registered in the tailor file

  ```
  headertype wb
  ```

- The `ia5` body part is okay as it stands, but to relay the message to X.400 or RFC-822 the header will need converting, therefore a filter to convert `hdr.wb` into either `hdr.822` or `hdr.p2` will be necessary. Only one need be written, because if *submit* has a `hdr.822` it can convert it to a `hdr.p2` using the *822top2* channel. Such a channel definition might look like this:

```
chan wbhdr-2-822 prog=wbto822,type=shaper,hdrin=wb,
                 hdrout=822,
                 show="WhizBang to 822 Header converter"
```

Chapter 6

Additional Capabilities

This chapter describes various units that may be added to PP to provide specific functionality. These units tend to provide new protocol functionality.

6.1 Fax capability in PP

This section describes PP's G3Fax capability and how to add it to the configured system.

The G3Fax capability is divided into several components:

- Addressing
- Tables
- Conversion
- Transmission and Reception

These are described below.

6.1.1 Fax Addressing

To enable an address to contain the information required for fax delivery, PP recognises two specific X.400 domain defined attributes:

`FAX`: the value of this attribute is used to identify the remote fax machine. It can be the actual telephone number of the machine (e.g. `/FAX=+44 602 790278/`), in which case it is used directly. Or it can be a string which is looked up in the *fax* table to get the required information (e.g. `/FAX=NeXor/`)

Note that the `FAX` attribute is compulsory. The PP fax transmission channel will bounce the message if this attribute is not present.

Note also that the number may contain symbols like '+' and '`P`'. The meaning and processing of these symbols is specific to the driver being used (see Section 6.1.4).

`ATTN`: the value of this attribute is a descriptive string. This string is printed on the coverpage of the fax along with the key `For the attention of`.

The `FAX` attribute identifies the remote fax machine and the `ATTN` attribute gives an informative string for presentation on the coverpage. Note that the presence of one or both of these attributes is not sufficient to route a message through PP for delivery via the fax transmission channel. Addresses will still have to be routed to the fax transmission channel via the conventional PP routing tables (the *channel* table and/or the the *users* table).

6.1.2 The Fax Table

This section describes the format of the table used by the fax related channels and filters. As the table is shared by both the conversion filters and the fax transmission channel, it is appropriate to describe the format and contents of the table here.

The general format of the table is the same as that of all PP tables. That is:

```
key:value
```

The table contains a mixture of delivery information and formatting information. The main type of entry is of the form

```
key: number,description
```

Where

`key`: is the value of the domain defined attribute `FAX` described in Section 6.1.1.

`number`: is the telephone number of the remote fax machine.

Any special symbol processing and meaning given to special values is a specific to drivers and described in Section 6.1.4

`description`: is a string describing the remote site. This description, if given, is printed on the coverpage by *hdr2fax*. *hdr2fax* maps the sequence "`\n`" to a newline when printing the string.

The fax table may also contain several hardwired keys used by the conversion filters and the transmission channel. These are:

`keyfont`: the value is the name of a file containing the font used by *hdr2fax* when printing out keys on the coverpage.

`valuefont`: the value is the name of a file containing the font used by *hdr2fax* when printing out the values associated with the keys.

`textfont`: the value is the name of a file containing the font used by *ia52fax* for all text and by *hdr2fax* for any subsidiary information it puts on the coverpage.

`xstart`: (see `ystart`)

`ystart`: the values of these are used by *hdr2fax* to identify the top left hand corner of the coverpage. The values can be given in units, `cm` for centimetres or `in` for inches. If no unit is specified, the value is assumed to be in millimetres.

`tab`: the value is the minimum amount of space to leave between the start of the `key` and the start of the `value` on the coverpage. It can be specified with units in the same way as `xstart` and `ystart`.

`coverpage`: the value is the name of a file containing the bitmap of a design which is used by *hdr2fax* as a backdrop to the coverpage.

`localOrg`: the name and address of the local organisation. This is printed by *hdr2fax* on the coverpage. Note that `\n` is mapped to a newline.

`localNumber`: the telephone number of the local organisation. This is printed on the coverpage by *hdr2fax*.

`postscript`: supplementary information which is printed at the bottom of the coverpage by *hdr2fax*. Note that `\n` is mapped to a newline.

`int_prefix` the prefix required to phone international numbers.

Figure 6.1 shows an example fax table.

```
#format information
keyfont:        /fonts/bh-lucida-bold-r-normal-sans-26
valuefont:      /fonts/bh-lucida-medium-i-normal-sans-26
textfont:       /fonts/bh-lucida-medium-r-normal-sans-26
xstart:         1.5in
ystart:         2.5in
tab:            2.5in
coverpage:      /bitmaps/nexor.bm
int_prefix:     010
localOrg:       NeXor,\nUniversity of Nottingham,\nUniversity Park,\nNottingham,\nNG7 2RD
localNumber:    +44 602 790278
postscript:     This fax was automatically generated by PP software.
#recipient information
lancaster.nexor.co.uk:  0,A local print
nott:           783450,Computer Science,\nNottingham University
cripps:         588138,Cripps Computer Centre,\nNottingham University
ucl:            071 387 1397,UCL,\nGower Street,\nLondon WC1E 6BT
bobbies:        999,The Police Station
```

Figure 6.1: Example of Fax Table

6.1.3 Fax Conversion

This section describes the fax conversion facilities supplied with PP. There are two filters:

hdr2fax: which converts an RFC-822 header to a G3Fax bodypart containing a coverpage.

ia52fax: which converts an ia5 bodypart to a G3Fax bodypart with multiple pages.

Note that the size of the page produced is driven by the `FAXPAGESIZE` macro from *Make.defs*. If this macro is not defined, the filters will produce US size pages. That is 216 mm wide by 273 mm high. If `FAXPAGESIZE` equals `-DA4`, the filters will produce A4 size pages, 210mm wide by 297 mm high.

These filters use fonts and bitmaps encoded in PP's own format. There are two subsidiary tools which can be used to produce the required format of fonts and bitmaps:

`xfontconv`: this converts an X windows' font into PP's format. It is run as follows:

```
xfontconv -font fontName -file fileName
```

where `fontName` is the name of the required font which must be present in the local X server and `fileName` is the name of the file in which to place the PP encoded version of the font.

`ximageconv`: this converts an X windows' bitmap into PP's format. It is run as follows:

```
ximageconv -in xbitmapFile -out fileName
```

where `xbitmapFile` is the file containing the bitmap to convert and `fileName` is the file in which to place the PP encoded version of the bitmap.

The *hdr2fax* Filter

The *hdr2fax* filter is used in conjunction with the general filter controller *fcontrol*. It produces a coverpage for a message, based on the information it is passed via the tailored `outinfo` field and the information it can glean from the RFC-822 header field.

The coverpage is built up as a series of key/value pairs separated by a set horizontal distance (`tab` see Section 6.1.2). These key/value pairs include To/value, From/value etc.

The information passed via the `outinfo` field is standard and as such the `outinfo` field should be tailored thus

```
hdr2fax -table $(table) -from $(400sender) -to $(400recip) \
-subject $(ua-id) -outmta $(outmta)
```

Although the `outinfo` is standard, you may omit or alter components of the field if you wish. This will result in the corresponding key/value pair being omitted from the coverpage. The only component which is compulsory is the `-table` component. Note that the `$(...)` values will be expanded automatically by *fcontrol*.

The components of the `outinfo` field are:

`-table`: the name of the fax table (see Section 6.1.2)

`-from`: the originator of the message.

`-to`: the recipient of the message.

`-subject`: the subject of the message.

`-outmta`: the pseudo mta to which this message is to be delivered.

There is a further consideration concerning the tailoring of *hdr2fax*. *hdr2fax* produces a coverpage specific to one recipient of a message. Therefore to ensure that each recipient gets a coverpage specific to himself, the tailoring for the *hdr2fax* should have the variable `solo-proc` set to `yes`.

Figure 6.2 shows an example of how *hdr2fax* should be tailored.

```
chan    822tofax    prog=fcontrol,type=shaper,
                    hdrin=822−us,hdrout=g3fax,
                    outinfo="hdr2fax -table $(outtable) -from $(400sender)
                        -to $(400recip) -subject $(ua-id) -outmta $(outmta)",
                    outtable=fax, cost=20,solo−proc=yes,
                    show="hdr.822 -> g3fax mapper"

chan    ia5tofax    prog=fcontrol,type=shaper,
                    bptin=ia5,bptout=g3fax,
                    outinfo="ia52fax -table $(outtable)",outtable=fax,
                    show="ia5 -> g3fax mapper"

chan    fax         prog=G3Fax, show="Fax channel",type=both,
                    content−in=p22, maxproc=1,
                    hdrout=g3fax, bptout=g3fax, drchan=dr2rfc,
                    outtable=fax, sort="user time",
                    outinfo="out=/dev/faxout,confirm=always,softcar=used",
                    ininfo="in=/dev/faxin,softcar=used,master=faxmaster@nexor.co.uk"
```

Figure 6.2: Example Tailoring of Fax Channels

The *ia52fax* Filter

The *ia52fax* filter is used, in conjunction with the general filter controller *fcontrol*, to convert ia5 bodyparts into multiple page g3fax bodyparts.

The `outinfo` field needs to specify which font the filter should use. This can be

- either indirectly via the `textfont` specified in the fax table (see Section 6.1.2). In which case the `outinfo` field should contain the component `-table $(table)`.
- or directly by specifying the name of the file containing the font to be used. The `outinfo` field should contain the component `-font filename`

Figure 6.2 shows an example of how *ia52fax* should be tailored.

6.1.4 The Fax Transmission and Reception Channels

This section describes the fax transmission and reception facilities supplied with PP. This code is divided into PP skeletons for an outbound fax channel and an inbound fax daemon, and drivers that are written to use these skeletons. At present there are only two drivers written using these skeletons. These drivers are for the following systems:

- the Panasonic Systemfax 250
- the Fujitsu dexNet200
- Class 2 fax modems

At the time of writing the inbound daemon is still in the beta stage.

The inbound daemon receives an incoming fax, wraps it up into an x400 message and sends it to a specified address. The outbound channel transmits a message to the remote fax machine via the local fax machine.

With channel pairing, both these entities can be tailored as one channel. Figure 6.2 shows an example of how these could be tailored. This example is written for the Panasonic Systemfax 250 driver.

Before describing the details germane to the individual drivers, the details germane to the PP skeletons are described. The key data structure linking the skeletons and the drivers is the structure `FaxCtlr`. The elements of this structure are mostly function pointers which should be filled in by the driver code. The structure contains other elements which mostly relate to the inbound daemon. These include: the address to send inbound faxes to; the subject to attach to the message sent; and the channel to run as.

In terms of tailoring, there is only one PP specific tailor element. This is for the outbound channel and is the element `confirm`. If this element is set to `always`, a positive delivery report will be sent back to the originator of the message when the message has been successfully transmitted. If set to `never`, positive delivery reports aren't sent. The default action is not to send delivery reports.

6.1.5 The Panasonic Systemfax 250 Driver

The Outbound Transmission Channel

The Panasonic Systemfax's outbound channel's behaviour can be altered by several key/value pairs in the `outinfo` tailor field:

`out=device`: the outbound device on which to contact the local fax machine. By default it will try to use `/dev/faxout`

`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`)

6.1.6 The Fujitsu dexNet200 Driver

Details not currently available here.

6.1.7 Class2 Fax Modems Driver

At the time of writing this driver only implements the outbound channel. Its behavior can be altered by several key/value pairs in the `outinfo` tailor field:

`out=device`: the outbound device on which to contact the local fax machine. By default it will try to use `/dev/faxout`

`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.

`wBlk=numBytes`: the maximum dimensions for each data block that may be send to the modem in a write operation. The default is 1024 bytes.

`localId=strings`: the string to be used as local ID string for the class2 `FLID` command. The default is "".

`maxPages=numPages`: the maximum number of fax pages that may be send for each message. The default is 10.

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.

6.2 FTBP—File Transfer Body Parts

NOTE: The interfaces described herein will be substantially changed in a future release.

This section describes *PP*'s FTBP capability and how to configure it. The details of the user-level programs can be found in *ISODE Volume 7: User's Guide: Message Handling Services*.

The FTBP services offered by *PP* are presented through two user-level programs, *fsn* and *fex* (q.v.). These programs allow users to submit files to the mail system for transfer and to retrieve files from a spool area when they are delivered.

The *fsn* program constructs a message with a single body part, of type `oid.2.6.1.4.12`, that can be delivered directly to an X.400(1988) recipient, or, with suitable processing, to a local or SMTP recipient. Support for FTBP within *PP* is provided by three new body types, three channels, and a table registering local FTBP users.

6.2.1 FTBP body types

The three body types used by FTBP are defined in the tailor file like this:

```
bodypart oid.2.6.1.4.12
bodypart ia5.ftbp.smtp
bodypart ia5.ftbp.filed
```

The `oid.2.6.1.4.12` body type is used for X.400 '88 recipients. It represents the particular form of externally defined (type 15) body part used by FTBP. The body type `ia5.ftbp.smtp` is used for transmitting files to Internet users—the file is wrapped in a message according to the template file *tables/ffiltform*. The body type, `ia5.ftbp.filed`, is used when the file is transferred to a local spooling area—instructions for extracting the file from the spool area, using *fex*, are constructed according to the template file *tables/ftxtform*.

The local delivery channels need to be configured to accept both `ia5.ftbp.smtp` and `ia5.ftbp.filed` body types. For example, the `822-local` channel definition might look like this:

```
chan 822-local prog=local,
               show="Local Delivery channel",type=both,
               sort="user time",adr=822,adr-order=usapref,
               hdrin=822,hdrout=822-us,
               bptout="ia5,
                       mime-image-g3fax,
                       mime-application-octet-stream,
                       mime-unknown,
                       ia5.ftbp.filed,ia5.ftbp.smtp",
               content-out=822,outtable=local,access=mts,drchan=dr2rfc
```

The `ia5.ftbp.smtp` body type will not be used if all local users have a defined FTBP spool area. However, it does provide a reminder that files transferred via SMTP from another MTA will effectively arrive as `ia5.ftbp.smtp` body types.

The SMTP channel needs to be configured to accept the `ia5.ftbp.smtp` body type. Note that it should not be configured to accept the `ia5.ftbp.filed` body type since any local spool area will be inaccessible to remote users.

Finally, the RFC 822 flattener channel (typically `mimeflatten` or `822flatten`) needs to be configured to accept both `ia5.ftbp.filed` and `ia5.ftbp.smtp`.

6.2.2 The FTBP spool table

The FTBP spool table is defined in the tailor file as, for example:

```
tbl ftbpspool file="ch.ftbpspool",
              show="local id -> FTBP spool area",
              flags=dbm
```

This file has identical syntax to the local users table and, indeed, wherever `outtable=ftbpspool` appears in the FTBP channel definitions, it is possible to use `outtable=local` if all local users have an FTBP spooling directory. The name of the spooling directory is constructed by appending `:FTBP` to the mailbox name derived from the `ftbpspool` table. Note that, like the local users tables, the `ftbpspool` table must contain an entry for the pp user.

6.2.3 FTBP Channels

Three channels need to be defined in the tailor file: one for submission of FTBP messages, and two for conversion of `oid.2.6.1.4.12` bodies to `ia5.ftbp.smtp` or `ia5.ftbp.filed` body types.

The submission channel must be called `ftrans` and is defined thus:

```
chan ftrans show="File Transfer submission",
            type=in,access=mts,
            adr-order=usapref,adr=x400,
            hdrout=p22,
            bptout=oid.2.6.1.4.12
```

The `oid.2.6.1.4.12` to `ia5.ftbp.filed` channel is used for local delivery of FTBP body parts. The *ftbp2txt* program saves the file contained in the message into the user's FTBP spool area and generates a text message explaining how to extract the file. In a sense, the `ftbpspool` channel performs part of local delivery and so should never be invoked for non-local recipients.

The channel is defined like this:

```
chan ftbpspool prog=fcontrol,type=shaper,conv=conv,cost=5,
               show="FTBP delivery spooler",
               bptin=oid.2.6.1.4.12,
               bptout=ia5.ftbp.filed,
               outtable=ftbpspool,
               out-info="ftbp2txt $(822sender) $(822recip) $(qid)"
```

The `oid.2.6.1.4.12` to `ia5.ftbp.smtp` channel provides a mechanism for converting FTBP body parts to text suitable for transmission through SMTP.

The channel is defined like this:

```
chan ftbpalt prog=fcontrol,type=shaper,conv=conv,cost=10,
             show="FTBP -> text",
             bptin=oid.2.6.1.4.12,
             bptout=ia5.ftbp.smtp,
             out-info="ftbpalt $(822sender) $(822recip) $(qid)"
```

Note that this channel has a higher cost than the `ftbpspool` channel, to ensure that it is not invoked for local users with an FTBP spool area.

The channel must be called `ftbpalt` as the `ftbpspool` channel uses it to provide fallback conversion when a local user does not appear in the `ftbpspool` table.

6.3 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

6.3.1 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.

`-c`**:** handle conversion between single-byte and double-byte encodings for accented characters. This is useful when converting between teletex and character sets like ISO 8859-1.

The character sets supported at present are:

none: Refers to no character set.

ia5: This is unencoded ascii text.

generaltext: The ISO general text 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",
            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,
            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

6.3.2 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

Management Tools

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 *ISODE Volume 14: Programmer's Guide: X.400 MTA*.

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). It defaults to the pp internal variable `qmgr_maxchans` if set, else to `3`.

`-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 occurring 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 Graphics 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 interval, the increment added to the interval after each unsuccessful reconnection attempt and the maximum interval can all be set dynamically from the configuration menu. This menu can also disable the reconnection mechanism and/or the increasing duration.

7.2.3 The Effect of the Mouse Buttons and Keys

This section describes how a user can use the mouse buttons and the keyboard to operate the MTA console. The following tables show the effect of the command buttons that can be found on the MTA console.

Top row of buttons

Button	Effect
`Refresh/Reconnect`	Refresh the data or reconnect to the QMGR.
`quit`	Exit the console program.
`connect/disconnect`	Connect to or disconnect from the QMGR.
`control/monitor`	Change between the two modes.
`configuration`	Set various parameters.
`Qmgr control`	Perform global QMGR control (only if authorised).

Channel buttons

Button	Effect
`info`	Display information about the channel.
`++`	Select next channel
`--`	Select previous channel
	If connected in authorised mode
`disable`	Disable this channel.
`enable`	Enable this channel.
`clear delay`	Clear any delays on this channel.
`add delay`	Specify a delay. The channel will wait this time before starting again.
`downward force attempt`	Invoke this channel now if possible, clearing any delays and those messages and MTAs waiting on this channel.

MTA buttons

Button	Effect
`info`	Display information on this MTA.
`++`	Select the next MTA
`--`	Select the previous MTA
	If connected in authorised mode
`disable`	Disable this MTA.
`enable`	Enable this MTA.
`clear delay`	Clear any delays for this MTA.
`add delay`	Specify a delay before next retry.
`force attempt`	Clear any delays for this MTA, an similarly for the channel this is on.
`downward force attempt`	Clear delays for this MTA and those of any messages on this MTA.

Message Buttons

Button	Effect
`info`	Display information about this message .
`++`	Select next message
`--`	Select previous message
`show all`	By default only information on ten messages is shown. This button will show all messages on this MTA/channel.
	If connected in authorised mode
`freeze`	Stop this message being delivered.
`thaw`	Allow this message to be delivered (this is the opposite of freeze).
`clear delay`	Clear the next delivery attempt time.
`add delay`	Set a next delivery attempt time.
`force attempt`	Try and deliver the message now by clearing all delays associated with this message.

Here are the effects each button has when pressed within the MTA console's display. It is based on a three-button mouse. Different configurations are tailorable through the X defaults mechanism for other hardware.

`left hand button`**:** If this button is pressed while the cursor is over one of the many *command* buttons in the console, that command is invoked.

If the console is in monitor mode and the left-hand button is pressed while the cursor is over the area occupied by the display of an entity, the console toggles to control mode, focuses on that entity, and displays information about the entities directly below it. For example, if in monitor mode the left-hand button was pressed while the cursor was over the area occupied by the channel `x400out84`, then the console would toggle modes, make `x400out84` the current channel and display information about the MTAs currently on that channel.

If the console is in control mode and the left-hand button is pressed while the cursor is over the area occupied by the display of an entity, then the console focuses on that entity and displays information about the entities directly below it. For example, if in control mode the left-hand button was pressed while the cursor was over the area occupied by the channel `x40084`, then

Accelerator	Command	Prerequiste
Meta + R	Refresh	
Meta + Q	Quit	
Meta + D	Disconnect	
Meta + C	Connect	
Meta + M	Toggle Modes	
Meta + F	Configuration	
Return	Ok	A popup must be on display
Control + C	Notok	A popup must be on display
Meta + H	Toggle Heuristic	The configuration popup must be on display
Meta + A	Toggle Authorisation	The connect popup must be on display

Table 7.1: MTA Console Accelerators

the console would make `x40084` the current channel and display information about the MTAs currently on that channel.

`middle button`: If the console is in monitor mode and the middle button is pressed while the cursor is over the area occupied by the display of an entity, the console toggles to control mode and focuses on that entity.

If the console is in control mode and the middle button is pressed while the cursor is over the area occupied by the display of an entity, the console toggles the focus on that entity (i.e., if the focus at that level is not on the entity, the focus is set on that entity. However, if the focus is already on that entity, the focus is unset).

`right button`: This button has a similar effect in both modes. If it is pressed while the cursor is over the area occupied by the display of an entity then the information displayed relating to that entity is refreshed and redisplayed.

There are various “accelerators” installed in the MTA console. These are key presses that allow the user to invoke commands that would otherwise have to be invoked by moving the cursor to a specific area on the screen and then pressing one of the mouse buttons. The accelerators installed in the console all provide alternative ways to invoke some of the commands represented by *command* buttons. Some have prerequisites for use. Table 7.1 shows the accelerators, their prerequisites, and the commands they invoke. This table has two modifier keys that must be held down together with the other keys mentioned in the accelerator for that accelerator to work – the *Meta* key and the *Control* key.

Several of the popups have various text boxes into which a user can type. At any one instant only one of these boxes can have the keyboard focus (the box with the keyboard focus gets the text the user types in). This is indicated by the text box with the focus having a border around it. All other text boxes will have no border. There are accelerators installed in the MTA console to allow the user to change the keyboard focus. The up arrow key when pressed will change the focus to the box above the one currently with the focus. The down arrow key when pressed will change the focus to the box below the one currently with the focus. The tab key has the same effect as the down arrow key. If the left-hand button on the mouse is pressed when the cursor is over one of the text boxes, that text box gets the focus.

7.2.4 Badness

The *badness* of an entity (channel, MTA or message) is a value used to compare the state of one entity with that of another. This comparison is used throughout the MTA console to order the displayed lists of entities, to choose which colour to display that entity in, and so on.

The badness of an entity is calculated from the sum of up to four factors.

`number`: The number of messages on an entity.

`volume`: The volume of messages on an entity.

`age`: For channel and MTA entities, this is the oldest message on the entity. For message entities, this factor is the age of the message itself.

`last`: The last successful attempt on an entity.

Each factor has a specific upperbound and these upperbounds are tailorable (this is described below). In the badness summation, the factors are expressed in terms of percentages of their respective upperbounds. So if the upper bound on the number factor is represented as *ub_number* and similarly for the other factors then the badness summation would be calculated as follows: $(number/ub\_number) * 100 + (volume/ub\_volume) * 100 + (age/ub\_age) * 100 + (last/ub\_last) * 100$, or, put more concisely, $\sum^{factors}(factor/ub\_factor) * 100$.

Tailoring of Upperbounds

When the console connects to a queue, it builds a database of tailoring entries. It can get the information for the database from any of three sources. The file specified in the connect popup, the hardwired defaults and the X defaults. Entries in the file override equivalent entries in the defaults. The file specified in the connect popup should be relative either to the current directory, a full path name, or the home directory.

An entry in this database has the form:

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

The `key` is a tuple indicating which channel, or type of channel and which type of entity on that channel the tailoring information in the value relates to. The channel or channel type and the entity are separated a "-". The entity must be one of:

```
chan
mta
msg
```

If the tailoring information is related to a specific channel, then the channel's name must be before the -. For example, if "822local," "x400in84," and "p2explode" are all names of channels, then the following are valid keys that can be used in the database: "822local-chan," "x400in84-mta," and "p2explode-msg".

If the tailoring information is related to a type of channel then the type of the channel appears before the -. The type of a channel is constructed from the `type` of that channel given in the *tailor* file and the `access` of that channel also in the *tailor* file. Table 7.2 gives all the types of channels that can be used in keys and how they are constructed. Some valid keys of this form are "mtaboth-chan", "passive-msg", "internal-msg."

Tailor File Values		
type	access	Key
in	mts	mtsin
out	mts	mtsout
both	mts	mtsboth
in	mta	mtain
out	mta	mtaout
both	mta	mtaboth
in		mtsin
out		mtsout
both		mtsboth
shaper		internal
warn		internal
delete		internal
qmgrload		internal
debris		internal
timeout		internal

Table 7.2: Tailor File Entries to Console Keys.

When the console searches through the database, it searches first for keys with the specific channel name in them (i.e., "channelname-entity"). If this fails, it searches for keys with the channel type in them (i.e., "channeltype-entity"). This second search will never fail as there are default entries for all possible channel types.

The `value` is a comma-separated list of expressions. These expressions are of the form `keyword<upperbound`, where keyword is one of:

```
age
vol
num
last
```

and upperbound is a number and an optional letter representing a unit. The possible units are

`h`: Hour (number * 60).

`d`: Day (number * 24 * 60).

`k`: Kilo (number * 1000).

`M`: Mega (number * 1000000).

Note that if one of the four keywords is missing from a value, the associated factor is ignored in the badness summation.

Figure 7.1 shows an extract from the MTA console's hardwired defaults.

Mappings to RFC 822 are done according to RFC 1327.

7.3 MTA Line-oriented Console

```
#define default_mtsboth_chan \
        "mtsboth-chan:age<2h,vol<1M,num<500,last<2h"
#define default_mtsboth_mta \
        "mtsboth-mta:age<2h,vol<1M,num<500,last<2h"
#define default_mtsboth_msg \
        "mtsboth-msg:age<2h,vol<1M"

#define default_mtain_chan \
        "mtain-chan:last<2h"
#define default_mtain_mta \
        "mtain-mta:last<2h"
#define default_mtain_msg \
        "mtain-msg:last<2h"

#define default_passive_chan \
        "passive-chan:age<20,vol<1M,num<10,last<20"
#define default_passive_mta \
        "passive-mta:age<20,vol<1M,num<10,last<20"
#define default_passive_msg \
        "passive-msg:age<20,vol<1M"
```

Figure 7.1: Extract From the Console's Tailoring

The *lconsole* provides basic access to the qmgr. It allows all the operations that may be achieved with the X based console, but does not have such good facilities for displaying the data. It has a number of commands that map onto the qmgr protocol. It may be run in authorised or unauthorised mode.

There are a few command line flags:

`-Q name`: Contact the given QMGR.

`-u user`: Use the given name to authenticate. A password will be prompted for if not given.

`-p passwd`: Connect with the given password. Use with care!

`-b`: Run in batch mode, little prompting is given in this mode.

Additional arguments on the command line, are taken as commands to execute. This allows the the possibility of shell scripts top access the queue status.

All commands may be abbreviated to any unique string. The *lconsole* has the following commands:

`channel`: This command lists out all the channels with their number of messages and reports by default. Normal channels are printed as `channel:msgs+reports`. Active channels have the `:` replaced by a `*`. Channels that are disabled have the `:` replaced by `!`. If an argument is given, then this is taken as a regular expression to match a channel. Any channel matched has its full details printed.

`clear`: This command clears a delay. The second argument is one of the following:

    `channel`: The next argument is a channel from which to remove the delay.

`mta`: The next two arguments are a channel and an mta from which the delay is removed.

`message`: The next argument is a queue ID, followed by a list of recipients numbers. For each recipient of the message, the delay is removed.

`close`: Close the connection to the QMGR.

`decrease`: Decrease the number of runnable channels.

`delay`: Set a specified delay for a given channel. The second argument is one of:

`channel`: The next argument is a channel followed by a delay given in the same syntax as the `MTAconsole`.

`mta`: The next two arguments are a channel and an mta followed again by a delay specification.

`message`: The next argument is a queue ID, followed by a a delay time and then a list of recipients numbers. For each recipient of the message, the delay is set.

`disable`: This command disables a channel, mta or message. The syntax is identical to the `clear` command. Additionally though, two other keys are recognised. `disable all` will disable all channels. `disable submission` will disable submission requests.

`enable`: This command is identical in syntax to `disable`, but with opposite effect.

`help`: With no arguments, this lists out a list of valid commands. With an argument, a one line summary of the specified command is given. An argument of `?` gives a one line summary of all commands.

`increase`: Increase the number of runnable channels.

`msg`: Give details about a messages queued on a channel/mta pair. The first argument is the channel (which may be a regular expression), the second an MTA. An optional third argument will print out full details on message queue id's matching that regular expression.

`mta`: Display information about mtas associated with a given channel. An optional second argument is a regular expression which allows more detailed information about matching mta names.

`open`: Make a connection to a QMGR. The optional second argument is the user to bind as. A password will be prompted for if a user is given.

`quit`: Close any outstanding connections, and exit the process.

`rereadqueue`: Forces the QMGR to initiate a re-read of the queue on disk.

`restart`: Restarts the QMGR.

`show`: This allows messages matching certain criteria to be displayed. The expression is of type key=value. The key may be any unique abbreviation of the following:

Key	Value
`channel`	The channel the message is queued on.
`contenttype`	The content type of the message.
`earlierthan`	Messages older than this time.
`eits`	A list of encoded information types.
`maxsize`	The size of the message.
`morerecentthan`	Messages younger than this date.
`mpduidentifier`	The MPDU id of the message.
`mta`	The next hop MTA.
`originator`	The originator of the message.
`priority`	The priority of the message.
`queueid`	The queueid of the message.
`recipient`	The recipient of the message.
`uacontentid`	The UA content ID field.

`shutdown`: Shuts down the QMGR. If an argument of `abort` is given then the QMGR is aborted and shuts down immediately.

`source`: Read in a file containing a list of commands and execute them.

`status`: Show the current status of the QMGR.

Some of the commands will not be allowed if the user has not bound in authorised mode.

7.4 SNMP Monitoring of PP

A PP MTA may be monitored using SNMP and an appropriate management agent. See *osiappd* (8) for details of setting up the necessary SMUX agent. The Applications Monitoring MIB (which may be needed to configure the management agent) is discussed in Chapter 11 in *ISODE Volume 3: Administrator's Guide: General Services*.

7.5 Checkup Procedures

Additional procedures exist for querying the queue directly about the status of the system. However, their prime importance is as development tools; the consoles should be the main source of information for the system administrator.

7.5.1 ckchan

The *ckchan* tool prints out information on the state of currently active channels and is invoked as follows:

```
ckchan [-v] [-s] <channelname> ...
```

ckchan prints out information on the specified channels. If no channels are specified, *ckchan* prints out information on all the channels currently active.

The presence of the `-v` flag on the command line instructs *ckchan* to output a verbose form of information.

If given a `-s` flag, the program only shows a summary of the information.

Warning: on even a moderately-sized queue, this process runs fairly slowly.

7.5.2 ckconfig

The *ckconfig* tool performs various checks on the configuration of PP including confirmation that:

- All the required directories are in place and have appropriate permissions.
- All the channels and filters described in the *tailor* file reference programs in the appropriate directory (*chndfldir*).
- All the tables described in the *tailor* file reference files in the appropriate directory (*tbldir*).

By default *ckconfig* will prompt for correction of any configuration errors it identifies.

ckconfig's behaviour can be altered by the following command line flags:

`-f`: (*force*) *ckconfig* will attempt to automatically correct configuration errors.

`-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 file`: Use this file as the PP *tailor* file.

It is worth running this program at intervals and after every change you make to PP.

7.5.3 Checking of Addresses

It is sometimes useful to check out addresses with and without the whole system being invoked. To this end two utilities are provided.

ckadr

PP provides a tool, *ckadr*, to check whether an address is acceptable to the mail system. Note that this tool does not do any of the PP authorization checks, so an address may be acceptable to *ckadr*, but PP may still refuse it due to authorisation criteria.

If the address is acceptable, *ckadr* will display how the address was parsed and how PP will deliver to this address. For example:

```
pac
Address parsing succeeded
pac -> (rfc822) p.cowen@cs.nott.ac.uk
    -> (x400)
  /I=p/S=cowen/OU=CS/O=NOTT/PRMD=UK.AC/ADMD=GOLD 400/C=GB/
Delivered to sheriff.cs.nott.ac.uk by 822-local
```

If the address is not acceptable, *ckadr* will display how far it got in the parse of the address and why it failed to find the address acceptable. For example:

```
/S=foo/OU=CS/O=NOTT/PRMD=HOL/ADMD=GOLD 400/C=GB/
Address parsing failed
Reason : Unknown PRMD HOL
Parsing gave this:
/S=foo/OU=CS/O=NOTT/PRMD=HOL/ADMD=GOLD 400/C=GB/ ->
  (rfc822) foo@CS.NOTT.HOL.gold-400.gb
/S=foo/OU=CS/O=NOTT/PRMD=HOL/ADMD=GOLD 400/C=GB/ ->
  (x400) /S=foo/OU=CS/O=NOTT/PRMD=HOL/ADMD=GOLD 400/C=GB/
```

If *ckadr* is invoked with command line arguments (other than its own command line flags), it will attempt to parse these arguments as addresses. Note that if any of the addresses have whitespace in them, they should be quoted to avoid being split up into separate arguments.

If *ckadr* is invoked with no command line arguments, it will read from the standard input until the end of that input. Each line of input is parsed as though it were an address.

The behaviour of *ckadr* can be modified with the following switches:

`-i inboundChan` Sets all addressing characteristics based on the inbound channel specified.

`-r`: Treat all addresses as 822 format addresses. This is the default behaviour.

`-x`: Treat all addresses as X.400 format addresses.

Note in most cases, *ckadr* will correctly parse an X.400 address even if you say it is an 822 address and vice versa. However there are some cases where this is not so, therefore be careful when using *ckadr* and different formats of addresses.

`-n`: Check the address without expanding aliases (i.e. as for originator addresses).

`-p`: Check the address and allow parsing of percent symbols in the route.

`-a`: Normalise all domains.

`-l`: Normalise domains with the USA ordering (`l`ittlendian) preferred.

`-b`: Normalise domains with the UK ordering (`b`igendian) preferred.

`-0`: recognise USA domain ordering *only*.

`-1`: prefer USA domain ordering.

`-2`: recognise UK domain ordering *only*.

`-3`: prefer UK domain ordering.

The above values relate to the numerical values used internally to specify which domain ordering to use. By default, *ckadr* will recogise USA domain ordering ONLY.

Probes

PP provides a tool, *probe*, which checks whether or not a message would be acceptable either locally or to a remote mail system.

A message probe attempts to transit the message transfer system network to its destination. On arrival at its destination or at a place where it can no longer be delivered (either because of errors or because a gateway through which the network beyond does not support probes is reached) a delivery report will always be returned. No message is ever delivered to a recipient as a result of a message probe.

The returned delivery report informs of the future successful or unsuccessful delivery of a message, having the same parameters as those specified within the previously generated message probe.

Within PP, the delivery report can be generated at either the remote end or locally. The exact location of the generation will depend on how the channels are specified within the PP *tailor* file.

If an outbound channel supports message probes, then it should have

```
probe=y
```

specified as one of its parameters in the *tailor* file. If this parameter is not specified than the default

```
probe=n
```

is taken. If `probe=y` is set against any outbound channel, then the delivery reports are generated remotely for any message probes outbound through that channel. It is assumed that the remote sites, reached via that outbound channel, will reply to the message probes. If, however, the parameter is set to `probe=n`, then the delivery reports will be generated locally, and this fact is recorded within the delivery report.

Note that the information within a delivery report could sometimes prove erroneous. For example, a remote recipient name (e.g James.Hacker) in an X.400 address needs clarifying, but the outbound channel associated with that recipient is set to `probe=n`. If the X.400 address is parsed correctly, then a successful delivery report is generated locally, regardless of whether or not the name of the recipient (e.g James.Hacker) is a valid one at the remote end. Thus it is necessary that the probe parameters are set correctly in the *tailor* file for *probe* to work correctly.

Note also, that only the X.400 channels support message probes, so any probing done of a non-X.400 site would have the delivery report generated by a gateway. Thus an X.400 remote user probing a recipient at a non-X.400 site, reachable via the local PP mail system, would have a PP generated delivery report. Once again the information within the delivery report could prove erroneous, if the remote recipient's address is parsed correctly but the recipient name is invalid.

Therefore, **for message probes to work absolutely correctly, probing must be carried out between X.400 sites only.**

The *probe* message generation program can be called in 2 ways:

- With command line arguments. It will attempt to parse the specified arguments before generating a message probe.
- Without command arguments. It will prompt on standard output and read on standard input until all the information has been received, before generating a message probe.

With the following options:

-t: Interprets the remaining arguments up to the next switch as "To:" Recipients.

-s: Specifies the size (in bytes) of the future message.

-e: Specifies one or more body part types within the future message. The list of types should be separated by a ",".

-u: Specifies a user agent identifier. This should be a maximum of 16 characters.

-i: Requests implicit conversion.

-a: Allows alternate recipients.

7.6 Dealing with Errors

To identify reasons for failure, it is necessary first to figure out which program is failing. This is not always easy as there are many programs involved in PP and at any one time, several of them may be interacting.

For a message submission that fails, it is rarely the inbound channel that is the problem, though obviously this can be the case sometimes. Most likely is that there is a problem in *submit*, so examining the *submit* log is likely to be a good starting point. It is also rare (but not unknown) that the problem is a bug in *submit*. The most likely causes initially are that you have one of the following wrong:

- The *tailor* file has things specified incorrectly. Most of the common problems here are missing formatters, badly specified channels, and missing directories. The *ckconfig* will help solve some of these.
- Another very likely cause is that the tables are wrong. This can be checked to a degree by use of the *ckadr* program. This allows addresses to be entered and the tables interrogated to determine the actions for that address. If you alter the tables, do not forget to run *dbmbuild* or else change the relevant table to linear mode.
- Worth checking, although less likely, is whether *submit* is installed properly with the correct setuid bit on (it should be mode 4755 or similar).

On outgoing channels, it is more likely either an error in one of the channels, a queue file that is incorrect, or an inconsistency between the QMGR and the queue on disc.

If the QMGR is running but not delivering anything, check to ensure the following:

- That the channels in question are enabled (check this via the MTA *console* program).
- That the *pptsapd* is running.
- That the channel programs exist in the right place, with the right modes (*ckconfig* will check most of these).

Remember, if you add channels or tables to the *tailor* file, then you will need to restart both the QMGR and the *pptsapd* programs.

If you wish to monitor the progress of the PP system through the logs in a dynamic way, it may be worth using the *ntail* program supplied in the *tools* directory. This program is much like running the UNIX *tail* (1) command with the `-f` flag on several files at once.

7.7 Examining the PP Queue

Occasionally there may be problems encountered by PP that cannot be resolved by normal means, and it will be necessary to look at the queue directory to find out what is going wrong. This section is a brief guide to this.

The PP Queue is stored under one directory, the *quedir* directory. An item in the queue is contained within a subdirectory of *quedir* with a unique name. This subdirectory may contain the following:

addr: a text encoded file containing information corresponding approximately to P1 functionality.

base: a directory which contains the message as it arrived.

filter.number: other directories which contain transformed versions of the message content.

report.number: files containing text encoded delivery reports associated with the message.

Of these components only the *addr* file must be present.

Each message content is a (possibly degenerate) directory tree. This hierarchy represents the structure of a message, with each body part being represented by a single file. Subdirectories represent forwarded IP messages. This structure is to allow for straightforward transformation of messages. When a message is converted, the body parts not affected are linked, rather than being copied.

An example queue file is shown in Figure 7.2.

7.7.1 Delivery Reports

The basic operation of a channel is discussed in Section 2.8 on page 41. Delivery Reports are text encoded, using a format similar to that used for text delivery. Reports are stored as files in the message directory, with name `report.<n>`, where n is the PP number for the lowest recipient to which the DR refers. Where a DR refers to multiple recipients, all except the lowest are marked to done in the queue, and the first is marked to dliv.

An example text-encoded delivery report is shown in Figure 7.3.

7.7.2 Receipt Notifications

These are handled by the UA, and are thus beyond the scope of PP. Mappings to RFC 822 are done according to RFC 1327.

```
Start-of-MsgEnvPrm
Message-type User-Mpdu
Message-size 84
Number-warnings 000
Warning-interval 24
Return-interval 72
Encoded-info EncTypes="hdr.822,ia5"
alt-recip-allowed true
cont-return true
Ua-id Test
pp-cont-corr "Subject: Test, To: pac"
content-corr 04165375626a6563743a20546573742c20546f3a20706163 none
Inbound-channel smtp
Inbound-mta "nexor.co.uk"
MsgId string="lancaster..123:25.08.91.15.53.50" Country=GB Admin=" "
Prmd="NeXor"
Trace Country=GB Admin=" " Prmd="NeXor"
arrival="Wed, 25 Sep 1991 16:53:49 +0100" action=0 End-of-block
Trace mta="lancaster.nexor.co.uk" Country=GB Admin=" "
Prmd="NeXor"
arrival="Wed, 25 Sep 1991 15:53:50 +0000" action=0 End-of-block
Queued-time "Wed, 25 Sep 1991 15:53:51 +0000"
Start-of-MsgEnvAddr
Origs rno=000 status=done reform-done=000 reform-list="822tous|822tous.1"
outchan=dr2rfc outmta="lancaster.nexor.co.uk" ext-id=0 resp=false
mta-rreq=basic usr-rreq=basic type=822 eits="ia5,hdr.822-us"
content=822 orig="pp@nexor.co.uk"
x400="/I=P/S=Pat/O=NEXOR/PRMD=NeXor/ADMD= /C=GB/"
rfc="P.Pat@nexor.co.uk" end-of-addr
Recip rno=001 status=pend reform-done=001 reform-list="822tous|822tous.1"
outchan=822-local outmta="lancaster.nexor.co.uk" ext-id=1
resp=true mta-rreq=basic usr-rreq=basic type=822
eits="ia5,hdr.822-us" content=822 orig=pac
x400="/I=p/S=cowen/OU=x400/O=NEXOR/PRMD=NeXor/ADMD= /C=GB/"
rfc="p.cowen@x400.nexor.co.uk" end-of-addr
```

Figure 7.2: Example of Address File

```
Msgid string=DR020905-900816141523Z Country=gb Admin=" " Prmd=interspan
Trace mta="bells.cs.ucl.ac.uk" Country=gb Admin="gold 400" Prmd="uk.ac"
        arrival="Thu, 16 Aug 1990 14:15:32 +0000" action=0 End-of-block
Trace Country=gb Admin=" " Prmd=interspan
        arrival="Thu, 16 Aug 1990 14:15:23 +0000" action=0 End-of-block
Report-Hdr-End
Recip-No 1
DR-Failure Unable-To-Transfer Unrecognised-ORName
Arrival-Time "Thu, 16 Aug 1990 14:14:59 +0000"
Reported-Recipient-End
Report-End
```

Figure 7.3: Example Delivery Report Format

7.7.3 Deleting Messages Cleanly

Occasionally an undeliverable message has an originator who also becomes undeliverable while the message times out (due to a configuration change for instance). Rather than deleting the message from the queue directory, the administrator should edit the status field in the 'addr' file to mark the message as delivered. This will allow the qmgr to delete the entry for the message, and stop trying to deliver it forever afterward.

Chapter 8

Ongoing System Management

8.1 The Day–to–Day Running of PP

8.1.1 How to Start Up the PP System

Each time the UNIX machine is started up, the *rc.local* script in your UNIX system is executed. An entry for PP should be placed in *rc.local*. This entry should refer to a script which starts up various programs related to the running of PP. Typically this script is similar to *pp.start* (see Appendix C) which starts up:

- QMGR: The queue manager which runs as the PP userID.
- *pptsapd*: The PP TSAP Daemon which also runs as the PP userID.
- *smtpsrvr* (optional): The SMTP listener which runs as the superuser. This is only necessary if:
  1. You are using the SMTP protocol.
  2. You are not running SMTP from *inetd* (see Section 2.7.2 on page 23 for details of this).
- *tsapd* (optional): is necessary if you are expecting incoming X.400 messages. This is assumed to be part of the general ISODE configuration; see *isode-gen* (8) for details.

Normally, these programs will run without interruption unless something serious happens, such as the machine going down. But if necessary you can always stop them by using the UNIX *kill* command. You can restart them by running the script manually or by invoking the programs separately, then letting them run in the background.

8.1.2 What to Run Each Night

In addition to running *pp.start* whenever the machine starts up, there are other things you should run regularly, ideally each night.

For example, if you build the PP tables from other sources, such as the NRS database, local system databases or other external sources, then running *dbmbuild* each night is a useful thing to do.

All the nightly routines can be run from a shell script, conventionally named something along the lines of *pp.night* (see appendix D for an example).

The sequence of operations should be :

- Archive the day's logs and remove outdated logs (see Section 8.1.3 on page 129).
- Collect statistics.
- Import new table source data files if necessary.
- Build new tables.
- *dbmbuild*.

8.1.3 Saving Logs

Saving logs depends very much on the individual policies for each site. For many cases the logs can just be set to truncate themselves when a suitable size is reached. Alternatively, if you are more concerned about keeping the information you should backup the logs each night. Two important exceptions to these general rules above are the *oper* log and the *stat* log.

The *oper* log should under normal conditions be empty, or at least very small. For this reason it is worth checking this overnight and perhaps mailing it to the postmaster if it contains anything.

The *stat* log can be processed to show statistics. There is a program in the *tools* directory call *statp*. This program processes the *stat* log file into a more structured format that is suitable for processing with *awk* (1). A sample awk script is supplied which shows the sort of output that can be produced from this.

Appendix A

Configuration of MHS for use with the Directory Service

This chapter provides an overview of how the Messaging Administration/ PP Message Handling Service module may be configured to make use of the X.500 Directory Service and the Domain Name Server (DNS).

WARNING. Use of the X.500 to provide routing is still an experimental function in this version of PP.

A.1 Introduction

The Messaging Administration/ PP MHS can be configured to use the X.500 Directory service and/or the Domain Name Server to override, augment or replace PP's tables when parsing and normalising addresses. If configured, the X.500 Directory Service is used to perform the following tasks:

- Resolve Distinguished Name to O/R address
- Bind O/R address to MTA
- Map between X.400 and RFC-822 addresses

A.2 Description of Resources

The following resources are used to configure the PP MHS module to use the X.500 Directory Service and the Domain Name Server (DNS).

A.2.1 Parsing Algorithm

The parsing algorithm used by the Messaging Administration/ PP MHS module is as follows:

1. Parse the address from the given input format (either RFC-822 or X.400) into PP's internal structures
2. Normalise and validate the address
3. Convert the address into the required format according to RFC 1327 (i.e., RFC-822 converted to X.400 or vice-versa)
4. If step 2 was successful then end with success
5. Else if the address format produced by step 3 is equivalent to the address we have previously attempted to parse, then end with failure
6. Else proceed to step 2 with the address found from step 3

This process may loop an arbitrary number of times to deal with cases where addresses carry encoded forms of other address formats (e.g., an X.400 address carrying an encoded RFC-822 address which may in turn carry an X.400 address etc.).

Various repositories of information are referenced in steps 2 and 3. Which repositories are actually used depends on:

- The lookup policy that has been set
- The address format that the algorithm is currently evaluating

The lookup policy is an ordered list of the repositories. For example, a policy of `"dns ds table"` means use the DNS then the Directory and then PP's tables when evaluating steps 2 and 3 of the algorithm.

Some combinations of repository, algorithm step and address format are nonsensical and so are bypassed. For example, the DNS contains no information relating to the conversion between X.400 addresses and RFC 822 addresses and so bears no relevance to the algorithm step 3. Also the DNS contains no information relating to the routing of X.400 addresses. Hence the DNS is skipped when evaluating step 2 for X.400 addresses.

In summary, the main algorithm loops between the two addressing formats attempting to normalise and validate each format in turn. A secondary algorithm goes through the policies at each step of the main algorithm.

A.2.2 The DIT Name Spaces

When PP makes use of the X.500 Directory Service three name spaces are used. PP only ever reads from these name spaces, it never writes information to them. The name spaces are:

DN the regular DIT that is used for storing organisations and people.

OR the part of the DIT that stores routing information for X.400 addresses

RFC the part of the DIT that stores information about RFC-822 domains

These trees are used to convert and route addresses. The details are discussed later.

A.2.3 Use of Policy

The address policies are referenced whenever an address format conversion or a lookup of information is required. For example, when an address-to-route mapping is required, an attempt to resolve the mapping is made using the list of policies. Each policy is tried in turn to resolve the mapping until one of the following occurs:

- Success in which case there is a result
- Positive failure, which indicates that the information does not exist
- Transient failure (such as a transient directory problem) normally causes the lookup to fail temporarily and no further options are tried (until the next time around)
- No information indicates that the next policy in the list should be tried

The policies are tried each time a lookup is tried, always starting with the first policy.

A.2.4 X.400 O/R Address Routing

O/R address routing is performed as follows. If the policy indicates use of the X.500 Directory service at this stage, the O/R address is converted into a distinguished name, following the conversion rules given in Table 3 on page 7. For example the O/R address

```
/i=j/s=onions/o=NeXor/prmd=NeXor/ADMD= /
c=GB/
```

is converted into the distinguished name

```
c=GB@aDMDName= @pRMDName=NeXor@mHS-
o=NeXor@cn=j.onions
```

The best match of this in the directory that contains a suitable mTAInfo attribute is taken as the routing information.

The `mTAInfo` information points at an MTA somewhere else in the DIT, usually in the DN part.

A.2.5 Conversion of X.400 O/R Addresses to RFC-822 Domains

This is achieved by a similar process to the above, but using the `associated-Domain` attribute to map the X.400 O/R name into a domain. Partial matches are handled as in RFC1327. The DIT portion used is the same as in the O/R routing tree.

A.2.6 Conversion of RFC-822 Domains into X.400 O/R Names

This is achieved by mapping RFC-822 domain components into attribute type `domainComponent`. For example the name

```
nexor.co.uk
```

would be converted into

```
O=Internet@domainComponent=uk@domainComponent=co@dom
ainComponent=nexor
```

This uses the RFC part of the DIT under `o=Internet`. The attribute `associatedORAddress` is used to perform the mapping. This again allows partial matching as described in RFC1327.

A.3 Compiling PP

PP should be compiled with

```
USE_X500
```

defined in the `h/config.h` file. This will compile in the X.500 directory access routines. In addition if the DNS is to be used for the verification of addresses, the variable

```
NAMESERVER
```

should be defined in `h/config.h`.

A.4 Tailoring for Use of X.500 and the DNS

A.4.1 Installation

Besides the usual installation of PP it is necessary to install a new set of object identifiers for QUIPU, `oidtable.*`. These new tables are found in Lib/x500 directory. They contain all the definitions from ISODE-8.0 plus the updated MHS routing definitions. To install them, copy the files into the ISODE source tree under the directory dsap, then type

```
make inst-oidtable
```

A.4.2 Lookup Policy

This is controlled by several PP tailor variables. The first two are

```
default_lookup_policy <mode>
default_lookup_timeout <time>
```

Key	Value
ds	Use the X.500 directory service
table[=prefix]	Use tables. If prefix present, prepend it to the standard table names: channel, domain, or, aliases and users
dns[=prefix]	Use the DNS. If prefix present, prepend it to the standard tables
dnstbl[=prefix]	Use the DNS in conjunction with the domain table. If prefix present, prepend it to the standard tables
nssoft	Continue with the lookup policies even if a transient failure occurred with the immediately preceding policy.
queue	Queue messages that can't be resolved due to temporary failure

Table A.1: Keys for Directory Usage

The first of these, `default_lookup_policy`, takes as arguments a list of variables which tailor the overall policy on the use of the X.500 Directory service and PP's tables and DNS. These should be chosen from the keys given in Table A.1. The default is `table`.

The second variable, `default_lookup_timeout`, is the time in seconds to spend resolving the query. This is used as a hint and so the time actually used may not be exact. The default is 30 seconds.

As well as the general tailoring, policies for individual channels may be specified by use of channel specific tailoring variables. These are defined as follows.

```
chan x40088 lookup="ds table", timeout=30,...
```

The parameters allowed in the lookup section are the same as those specified in Table A.1. Submit uses the policies associated with the unbend channel. Outbound channels (currently only x40088) use the policy to resolve O/R addresses to presentation addresses. With the exception of the `queue` parameter which can occur anywhere, the ordering of the parameters is important. The parameters are followed in strict order as applicable. The `nssoft` parameter only makes sense directly after a directory or namesever interaction policy (such as `ds` or `dns`). It is possible to divide the inbound and outbound lookup policies by use of the separate channel tailoring parameters inlookup and outlookup.

One other important variable is the Distinguished Name of the local MTA. This is tailored using

```
loc_mtadnname <value>
```

This is a Distinguished Name in QUIPU format that points to the local MTA. It is used to work out if the MTA is local (supportingMta is also used) and to retrieve the transport communities supported to compare network connectivity.

Object Identifier	OID name	Protocol
2.6.0.1.6	ac-p1-1988	X.400(88)
1.3.6.1.4.1.453.5.1	ac-p1-1984	X.400(84)
1.3.6.1.4.1.453.5.2	ac-smtp	SMTP
1.3.6.1.4.1.453.5.3	ac-uucp	UUCP
1.3.6.1.4.1.453.5.4	ac-jnt-mail	JNT Grey Book Mail

Table A.2: Application Contexts

A.4.3 Channel Binding

It is necessary to set the Application Context that a channel supports using the `appcont` keyword. PP uses it to facilitate channel binding. A channel may support multiple contexts, in which case it is a comma separated list. The recommended values are given below in Table A.2. An example usage might be:

```
chan x40088... appcont="2.6.0.1.6",...
chan smtp  ... appcont="1.3.6.1.4.1.453.5.2",...
```

These object identifiers are declared in the oidtables distributed with PP. At present they must be entered in numeric format (to cut down on tailoring overhead for all channels).

A.5 Information to be Added

This section covers the basic information that needs to be added to the X.500 Directory to make it function correctly. A fuller description is given in the Internet draft document "MHS Use of the Directory, Version 1".

A.5.1 MHS Routing Information

The most important use of the Directory for mail is to find the routing information for delivery. This is obtained by looking up in the Directory an OR address represented by a Distinguished Name. The conversion between OR address components and Distinguished Name components is given in Table A.3

Other protocols may use the directory to convert to/from X.400 addresses. The first step is to register as much of your OR address as is necessary in the Directory. What is necessary depends on a number of circumstances. However the basic algorithm used for directory lookup is the following:

1. Convert O/R Address to Distinguished Name and attempt to read that entry
2. If this succeeds, Goto 5

```
objectClass= top & quipuObject &
oRAddressComponent & pRMD
pRMDName= NeXor
authorativeAddress= NON-AUTHORATIVE
mTAInfo= c=GB@o=NeXor@cn=lancaster@
cn=nexor-mta # 5 #
mtaInfo= c=GB@o=University London Computer
Centre@ cn=mhs-relay # 10 #
```

Figure A.1: Example OR Routing Component

3. Else, take the longest match found and read that entry
4. Read the `mtainfo` attribute associated with this entry
5. If there is no `mtainfo` attribute, remove the last component of the DN and Goto 4
6. Read the given MTA
7. If the MTA is not suitable, remove one component of the DN and Goto 3

Attributes that are used in the routing decision are:

`MTAInfo` This is the attribute that specifies the MTAs associated with this part of the route. For each MTA there is a distinguished name, a weighting factor and optionally a set of attributes taken from the entry of the given distinguished name. The MTAs are sorted into their weighted order. If the local MTA is included in this list, MTAs of higher weighting are ignored.

`authorativeAddress` This attribute indicates that all addresses below this part of the tree are registered. Thus, if a partial match in the OR tree is being attempted and an `authorativeAddress` with value `AUTHORATIVE` is found, it indicates the address that is being attempted as a match does not exist. So, if set, all possible components under this point in the tree must be registered.

N.B. Other attributes mentioned in the paper "MHS use of the Directory, version 1" are not implemented.

A fair amount of defaulting can be achieved. If you have a large organisation with several levels of organisationalUnits, then it may be appropriate to just add an MTA at the organisational level.

Once an MTA has been located through the `Mtainfo` attribute, it is checked for suitability. At present, this is just a check that your MTA and the given MTA have a transport community in common, and that the application context is useable.

This gives the basic routing information required. An example of an entry in the routing tree, in QUIPU EDB format, is shown in Figure A.1.

An MTA referred to in the `mtainfo` of an OR routing component should be of Object Class `mta`, and an example of such an entry is shown in Figure A.2.

The important attributes here are the following:

`PresentationAddress` Indicates the address to which to connect.

OR Name Component	DistinguishedName Object Class	X500 Naming Attribute
Country	mhsCountry	countryName
Country (numeric)	mhsCountry	mHSNumericCountryName Name
ADMD	aDMD	aDMDName
PRMD	pRMD	pRMDName
Organisation	mHSOrganisation	mHS-O
OrganisationUnit	mHSOrganisationUnit	mHS-OU
X121	mHSX121	mHSX121Address
CommonName	mHSNamedObject	mHSCN
DomainDefined	mHSDomainDefinedAttribute	mHS-DDA%mHS-DDV

Table A.3: Distinguished Name Components

```
objectClass= top & applicationEntity & \
quipuObject & mhsMessageTransferAgent & mTA
cn= nexor-mta
description= NeXor’s main MTA
presentationAddress= "597"/TELEX+00737346+RFC-
1006+03+128.243.9.1|TELEX+00728722+X.25(80)+02
+00002100102999|TELEX+00728722+X.25(80)+06+204
33450210399|DCC+826+d110000210021900015000
supportedApplicationContext= ac-p1-1988
mTAName= UK.CO.NEXOR
transportCommunity= tc-janet & tc-ixi & tc-
internet & tc-cons
```

Figure A.2: Example MTA Entry

`SupportedApplicationContext` This is the application context supported by this MTA, and is used within PP to map the entry on to a list of possible channels. This attribute may be multi valued, indicating the MTA can handle more than one protocol. In practice this is unlikely as the different protocols will have to use the same presentation address. Instead a common situation will be a pseudo MTA defined for each protocol the real MTA supports.

`mTAName` This is used in the initial RTS handshake, along with a password of a single space.

`transportCommunity` This indicates which transport communities the MTA can support. This attribute is important as it is compared with your own MTA’s transport communities to ensure they have at least one community in common. The values consist of object identifiers and common values are given in Table A.4.

This is all that is required to do the basic routing. Pictorially the relevant entries in the Directory are

Transport Community	Object Id	Usage
tc-cons	1.3.6.1.4.1.453.4.1	OSI CONS Network connectivity
tc-clns	1.3.6.1.4.1.453.4.2	OSI CLNS Network connectivity
tc-internet	1.3.6.1.4.1.453.4.3	Access to the Internet via RFC-1006
tc-int-x25)	1.3.6.1.4.1.453.4.4	International X.25 connectivity (not CONS
tc-ixi	1.3.6.1.4.1.453.4.5	Access to the European IXI network
tc-janet	1.3.6.1.4.1.453.4.6	Access to the UK Janet Network
<object-identifier>	???	An object identifier may be given to specify a private community

Table A.4: Transport Communities

shown in Figure A.3.

A.5.2 Entries for O/R to RFC822 Conversion

As well as configuring routing information, it is useful to configure the conversion between RFC-822 domains and O/R address components. This is done by including conversions at suitable points in the tree. As with routing, use of defaulting by using the longest match allows many entries to be omitted.

To add a mapping entry for O/R to RFC-822 domains, it is necessary to add an entry at the appropriate part of the OR name tree in the directory, or objectclass `x400ToRFC822Mapping` with the attribute `associatedDomain` containing the RFC-822 domain. For example:

In practice this entry is usually an existing entry in the OR routing tree, normally an organisation, prmd or admd.

To achieve the reverse mapping, it is necessary to add an entry to the domain part of the X.500 tree. This is routed at the top node or `O=Internet`.

The same process as for X.400 to RFC-822 mapping is used, except the objectclass to use in this case is `rFC822toX400Mapping` and the attribute to use is `associatedORAddress`.

Again, this may be a node that already exists.

With these two nodes in place it is possible to use the directory to drive the conversion of RFC-822 domains into X.400 address components, and vice-versa.

A diagram of these parts is given in Figure A.6

A.6 Advanced Usage

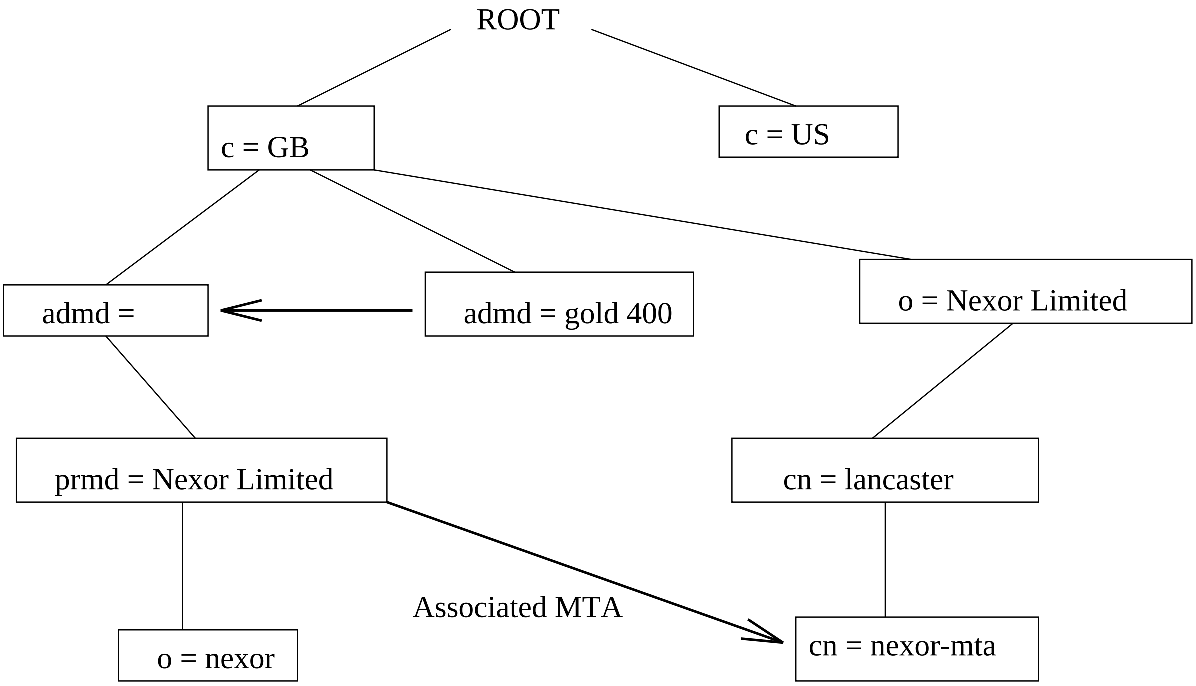

Figure A.3: The DIT Set Up for Routing

```
objectClass= quipuObject & x400ToRFC822Mapping
associatedDomain= nexor.co.uk
mHS-O= nexor
```

Figure A.4: Example Entry for Mapping X.400 to RFC-822

It is possible to use routing trees other than those described above for X.500 lookup. If your MTA entry contains the attribute `routingTreeList` then this is taken as an ordered list of routing trees to attempt to read. This list may include the default set of trees described above. For example:

```
routingTreeList= c=GB@o=NeXor
Ltd@cn=routing tree $ $ c=GB@cn=routingTree
```

This would use the following trees: a private one, followed by the global public tree, and finally a GB specific tree. The `routingAction` attribute described in the Internet draft document "MHS Use of the Directory, Version 1" is not used.

```
domainComponent= nexor
objectClass= quipuObject & rFC822toX400Mapping
associatedORAddress= /O=NeXor/PRMD=NeXor/
ADMD= /C=GB/
```

Figure A.5: Example Entry for Mapping RFC-822 to X.400

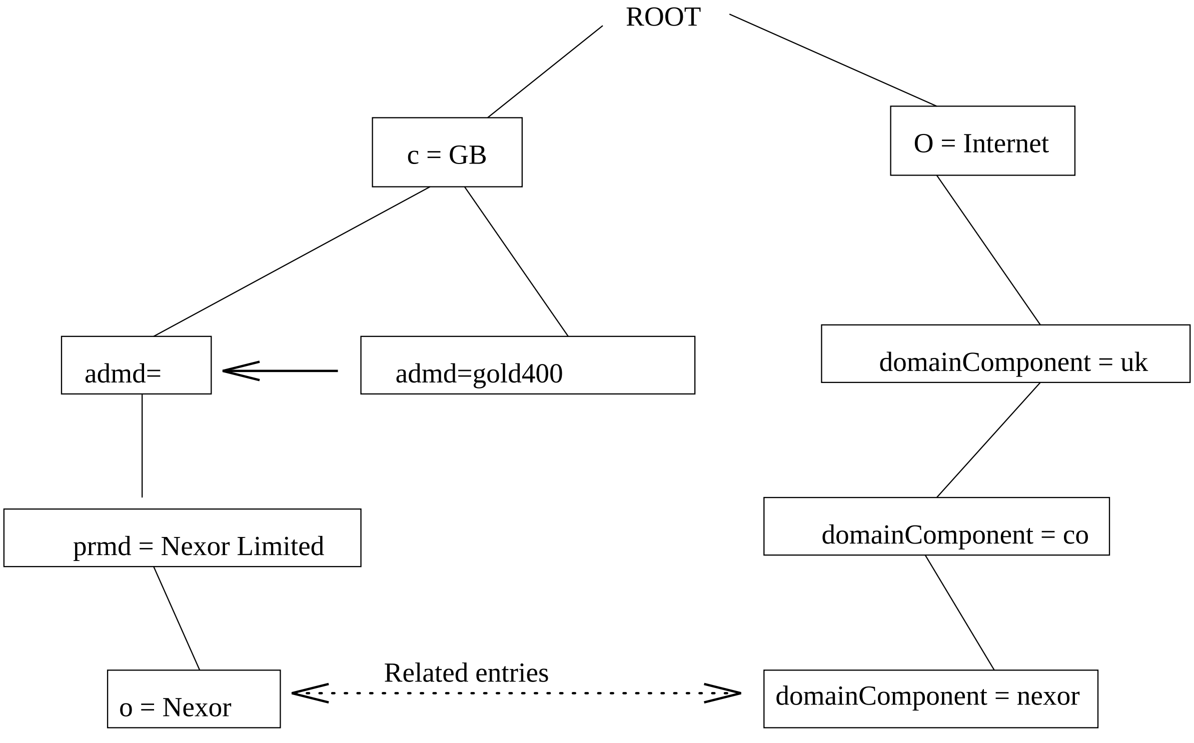

Figure A.6: Example layout of the Internet tree

Appendix B

Example of the Tailor File

What follows is a suggested layout for the PP runtime tailor file. As the file is read in by most programs, it is best if the number of long comments is kept to a minimum.

```
###############################################################
#
# Tailor file for PP
#
#        This version for sites with local smtp
#
###############################################################

###############################################################
#
# The following variables will need tailoring
#
###############################################################
#
# The loc_dom_mta is the host that the MTA will reside on and execute from
# It is a fully qualified domain name and represent the physical machine
#        Examples might be
#                cheetah.nyser.net
#                violet.berkeley.edu
#                john.cs.nott.ac.uk
#
loc_dom_mta      lancaster.nexor.co.uk
#
# The Sitename is what the rest of the world sees you as. This is typically
# a domain shorter than the full domain name. It is the name used to represent
# your site or group as a whole. This name normally hides the individual
# machine names
#        Examples might be
#                berkeley.edu
#                nyser.net
#                cs.nott.ac.uk
#
loc_dom_site     nexor.co.uk
#
# The postmaster should include the local site name
# e.g. postmaster@berkeley.edu or postmaster@cs.nott.ac.uk
#
postmaster       postmaster@nexor.co.uk
```

```
#adminstration_assigned_alternate_recipient pac

# X.500 information
loc_mtadnname "c=GB@admdname= @prmdname=NeXor@cn=nexor-mta"
#default_lookup_policy table ds
default_lookup_timeout 20

#
# The pplogin should be the username of the owner of pp.
#
pplogin             pp
#
# The qmgrhost should be the same as the name found in isoentities file.
# This can be the same as loc_dom_mta name
#
pptsapd_addr        LOCAL−ETHER=lancaster+20001
qmgrhost            lancaster
#qadaddr            LOCAL−ETHER=tornado+18001
#
# The local or name should be the name of your site in X.400 terms.
# This is attribute value based, and starts with organisational units and
# works up to country. If you are unsure, leave it with just a country
# component, e.g. "/C=GB/"
#
loc_or              "/o=nexor/prmd=NeXor/admd= /c=GB/"
x400mta    UK.CO.NEXOR

# The name of the dsaptailor file
#dsaptailor /nexor/pp/pp−beta/Lib/x500/dsaptailor

############################################################
#
# That is the end of the things you must tailor − the rest are
# probably ok − but you should check.
#
############################################################

delim1              "\1\1\1\1\12"
delim2              "\1\1\1\1\12"
#returntime 36
#queuestruct        100
dap_user            "c=GB@o=NeXor@cn=lancaster@cn=pp qmgr"


authchannel         free
authloglevel        high

wrndfldir           warnings
#submit_addr        lancaster:pp−submit

# Body types

headertype          822 822−jnt 822−us p2 822−uk p22 ipn g3fax ps
bodypart            encrypted g3fax ia5 ipm national sfd tif0 tif1
bodypart            tlx undefined videotex voice odif iso6937
bodypart            bilateral external
bodypart            dmd ia5−dk ps ms
bodypart            oid.0.9.2342.60200172.99.1.1.4
bodypart            oid.2.6.1.4.12
bodypart            ia5−asn

#for regression tests at mo but who knows the future ?
headertype          p222
```

```
# **-- Mapping tables --**
#
# All of these table MUST have the names given here
tbl     aliases                 show="Aliases:  mapping -> local id",
                                flags=dbm
tbl     users                   show="Users:  mapping local id -> disposition",
                                flags=dbm
tbl     x400-developers-aliases         show="Aliases:  mapping -> local id",
                                flags=linear
tbl     x400-developers-users           show="Users:  mapping local id -> disposition",
                                flags=linear
tbl     bu-aliases              show="Aliases:  mapping -> local id",
                                flags=linear
tbl     bu-users                show="Users:  mapping local id -> disposition",
                                flags=linear
tbl     gate-aliases            show="Aliases:  mapping -> local id",
                                flags=linear
tbl     gate-users              show="Users:  mapping local id -> disposition",
                                flags=linear
tbl     domain                  show="Mapping domain key ->full domain/MTA",
                                flags=dbm
tbl     or                      show="Mapping O/R Address -> MTA",flags=dbm
tbl     channel                 show="Binding MTA -> Channels",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     rfc1148gate             show="Gateways that perform RFC 1148 conversions",
                                flags=dbm
tbl     auth.channel            show="Authorisation:  channel policy",flags=dbm
tbl     auth.mta                show="Authorisation:  mta based",flags=dbm
tbl     auth.user               show="Authorisation:  user based",flags=dbm
tbl     auth.qmgr               show="Authorisation:  qmgr control",flags=dbm
tbl     auth.local              show="Authorisation for local users",flags=linear

# **-- Channel tables --**
#
# Needed by individual channels, not submit
tbl     local                   file="ch.local",
                                show="local id -> user id + home directory",
                                flags=dbm

tbl     x400in84                file="ch.x400in84",
                                show="MTA -> connect info",flags=linear
tbl     x400out84               file="ch.x400out84",
                                show="X.400(84) connect info -> MTA + other info",
                                flags=linear
tbl     x400in88                file="ch.x400in88",
                                show="MTA -> connect info",flags=linear
tbl     x400out88               file="ch.x400out88",
                                show="X.400(88) connect info -> MTA + other info",
                                flags=linear
tbl     shell                   file="ch.shell",
                                show="info for shell chan"
                                flags=linear
tbl     list                    file="ch.list",
                                show="Lists:  distribution lists",
                                flags=dbm
tbl     uucp                    file="ch.uucp"
                                show="UUCP: rfc822 -> uucp",
                                flags=dbm
tbl     fax                     file="ch.fax",
                                show="FAX: mappings and info",
```

```
                                flags=linear
tbl     mslocal                 file="ch.mslocal",
                                show="Message Store:local id -> user id,home directory",
                                flags=linear
tbl     p7local                 show="P7 server mappings",flags=linear

chan    822−local               prog=local,
                                show="Local Delivery channel",type=both
                                sort="user time",adr=822,adr−order=usapref,
                                hdrin=822,hdrout=822−us,
                                bptin=ia5,bptout=ia5,lookup="ds table"
                                content−out=822,outtable=local,access=mts,drchan=dr2rfc

chan    x40088          show="X400 (1988)",type=both, adr=x400, adr−order=ukpref,
                                key=x400in88, intable=x400in88, prog=x400out88,
                                content−out="p2,p22",hdrout="p22,p2,ipn",
                                outtable=x400out88,probe=y,appcont="2.6.0.1.6",lookup="ds table"

chan    x40084          show="X400 (1984)",type=both, adr=x400,probe=y
                                adr−order=ukpref, key=x400in84,
                                intable=x400in84, prog=x400out84, content−out=p2,
                                hdrout=p2, bptout="ia5,g3fax,odif",outtable=x400out84

chan    xua                     show="XUA Submission Channel",type=in,
                                access=mts,adr−order=ukpref,
                                adr=x400, access=mts,
#                               lookup="ds,table",timeout=30

chan    mslocal                 prog=mslocal,show="MS local delivery",type=out,
                                hdrout="p22,p2,ipn",
                                outtable=mslocal,
                                sort="user time",access=mts

chan    mslocal−beta            prog=mslocalbeta,show="MS local delivery",type=out,
                                hdrout="p22,p2,ipn",
                                content−out="none,p22",
                                outtable=mslocal,
                                sort="user time",access=mts

chan    ftrans                  show="File Transfer submission",
                                type=in,access=mts,adr=x400

chan    ftbp2txt                prog=fcontrol,type=shaper,
                                bptin="oid.2.6.1.4.12",bptout=ia5,
                                out−info="ftbp2txt $(822sender) $(822recip) $(qid)",
                                show="ftbp -> text"

chan    smtp                    prog=smtp,show="with SMTP (PP)",type=both,
                                adr=822,adr−order=usapref,drchan=dr2rfc,
                                bptout=ia5,hdrout=822−us,content−out=822,


chan    gb−janet                prog="greyout -P",show="via JANET with NIFTP (PP)",
                                type=both, adr=822,adr−order=ukpref,
                                hdrout=822−jnt,domain−norm=full,
                                bptout=ia5,drchan=dr2rfc,
                                content−out=822,out−info=janet


chan    fax                     prog=ps250, show="Fax channel",type=both,
                                hdrin=p22, bptin=g3fax,
                                hdrout=g3fax, bptout=g3fax, maxproc=1,
                                drchan=dr2rfc,outtable=fax, sort="user time",
```

```
                        out−info="out=/dev/faxout,confirm=always,softcar=used",
                        in−info="in=/dev/faxout,softcar=used,master=avr@nexor.co.uk"

chan    dirlist         prog=dirlist,show="Directory List channel",type=both,
                        adr=x400,adr−order=usapref,
                        content−out="p2,p22,none",sort="user time",
 out−info="dn=C=GB@O=NeXor@CN=lancaster,localdit=C=GB@O=NeXor@ou=distribution lists"

chan    list            prog=list,show="List channel",type=both,
                        adr−order=usapref,out−info="linked",
                        content−out="p2,p22,none",
                        outtable=list,sort="user time"

chan    uucp            prog=uucp−out,key="uucp-out,rmail",
                        key=uucp−in, show="UUCP channel",type=both,
                        adr=822,adr−order=ukpref,outtable=uucp,
                        hdrout=822−jnt, bptout=ia5,
                        out−info="uux=/tmp/uux,host=nott-cs",drchan=dr2rfc

chan    dr2rfc          prog=dr2rfc,show="Dr2rfc channel",type=both,
                        adr=822,adr−order=ukpref,
                        bptin=ia5,hdrin=822,
                        out−info="return=all,admininfo=true",
                        content−out=822

chan    shell           prog=shell,type=out,access=mts
                        show="Shell channel",
                        outtable=shell,sort="user time",
                        bptout="ia5,ps",hdrout=822−uk

# QMGR special channels (load/manage/clean)

chan    qmgr−load       prog=qmgr−load,show="Loading the QMGR",
                        type=qmgrload,out−info="5";

chan    msg−clean       prog=msg−clean,show="Removing finished message"
                        type=delete

chan    trash           prog=trash,show="Removing trash"
                        type=debris,out−info=3h

chan    timeout         prog=timeout,show="Timeout messages"
                        type=timeout

chan    warning         prog=warnings,show="Send warning messages",
                        type=warn

chan    delay           prog=delay, type=delay, adr=any,
                        content−out="p22,p2,none",
                        show="Delayed submission channel"


# **−− shaper channels −−**

# Flatteners

chan    p2flatten       prog=p2flatten,show="Structured message -> P2",
                        type=flat,content−out=p2,hdrout=p2

chan    822flatten      prog=rfc934,type=flat,content−out=822,
                        hdrout=822,bptout=ia5

chan    p22flatten      prog=p2flatten,show="Structured Message -> P22",
```

```
                                type=flat,content−out=p22,hdrout=p22

#  Unflatteners

chan    p2explode       prog=p2explode,type=unflat,content−in="p2,p22",
                        out−info="ia5=822"
#chan   p22explode      prog=p2explode,type=shaper,content−in=p22


# Header Filters
chan    822tofax        prog=fcontrol,type=shaper,
                        hdrin=822−us,hdrout=g3fax,n,
                        cost=20,solo−proc=yes, outtable=fax,
  out−info="hdr2fax -from $(400sender) -to $(400recip) -subject $(ua-id) -table $(outtable) -outmta $(outmta)"

chan    822touk         prog=fcontrol,type=shaper,
                        hdrin=822,
                        hdrout=822−uk,
                        out−info="rfc822norm -adr-order ukpref -jnt -bigend"

chan    822tojnt        prog=fcontrol,type=shaper,
                        hdrin=822,hdrout=822−jnt,
          out−info="rfc822norm -adr-order ukpref -jnt -bigend -jntsender $(822SENDER)"

chan    822tous         prog=fcontrol,type=shaper,hdrin=822,
                        hdrout=822−us,
                        out−info="rfc822norm -adr-order usapref -822 -littleend"

chan    822top22        prog=RFCtoP2,type=shaper,hdrin=822−us,
                        conv=1148
                        hdrout=p22,cost=10,
                        show="RFC1148 RFC 822 -> P22"

chan    822top2         prog=RFCtoP2,type=shaper,hdrin=822−us,conv=1148
                        hdrout=p2,cost=15,
                        show="RFC1148 RFC 822 -> P2"

chan    p2to822         prog=P2toRFC,type=shaper,hdrout=822,conv=1148,
                        hdrin="p2,ipn",cost=10,
                        show="RFC1148 P2 -> RFC 822"

chan    p22to822        prog=P2toRFC,type=shaper,hdrout=822,conv=1148
                        hdrin="p22,ipn",cost=10,
                        show="RFC1148 P22 -> RFC 822"

chan    p22top2         prog=fcontrol,type=shaper,hdrout=p2,hdrin=p22,
                        show="P22 to P2 downgrade",
                        conv=conv,cost=5,out−info="p2norm -downgrade"

#identity map
chan    p2top22         prog=fcontrol,type=shaper,hdrout=p22,hdrin=p2,
                        show="P2 to P22 upgrade",cost=1,
                        out−info="/bin/cat"

# Body part filters
chan    ia5−asn         prog=fcontrol,type=shaper,
                        bptin=ia5,bptout=ia5−asn,
                        show="822 UNIX ia5 to X400 ASN ia5",
                        out−info="ia52asn"

chan    asn−ia5         prog=fcontrol,type=shaper,
                        bptin=ia5−asn,bptout=ia5,
                        show="X400 ASN ia5 to 822 UNIX ia5"
```

```
                        out-info="asn2ia5"

chan    ia5-dk2ia5      prog=fcontrol,type=shaper,bptin=ia5-dk,
                        bptout=ia5,show="Danish ia5 to ia5",
                        conv=loss,out-info="charset DK 38 UK 29"

chan    iso6937totxt    prog=fcontrol,type=shaper,bptin=iso6937,
                        bptout=ia5,conv=loss,cost=15,show="ISO6937 -> IA5",
        out-info="asn -inchar iso_6937-2-add -outchar irv -inasn motis-86-6937 -outasn ia5",

chan    g3faxtotxt      prog=fcontrol,type=shaper,bptin=g3fax,
                        bptout=ia5,out-info="removebp G3Fax",
                        show="Fax -> Text mapper",cost=100,conv=loss

chan    tif0totxt       prog=fcontrol,type=shaper,bptin=tif0,
                        bptout=ia5,out-info="removebp Tif0",
                        show="Tif0 -> Text mapper",cost=100,conv=loss

chan    txttofax        prog=fcontrol,type=shaper,bptin=ia5,
                        bptout=g3fax,
                        out-info="ia52fax -table $(outtable)",outtable=fax,
                        show="ia5 -> g3fax mapper"

chan    faxtops         prog=fcontrol,type=shaper,bptin=g3fax,
                        bptout=ps,out-info="fax2ps",show="fax -> postscript"

chan    mstops          prog=fcontrol,type=shaper,bptin=ms,
                        bptout=ps,out-info="/usr/local/bin/psroff -ms -t",
                        show="ms macros -> postscript"

chan    p7server        type=in,show="P7 inbound server",key=xp7server,
                        outtable=p7local

chan    mstoia5         prog=fcontrol,type=shaper,bptin=ms,
                        bptout=ia5,out-info="/bin/cat",
                        show="MS->Text mapper",cost=100,conv=loss

chan    xxtotxt         prog=fcontrol,type=shaper,bptin=oid.0.9.2342.60200172.99.1.1.4,
                        bptout=ia5,out-info="removebp oid.0.9.2342.60200172.99.1.1.4"
                        show="XXX -> Text mapper",cost=100,conv=loss

chan    voicetotxt      prog=fcontrol,type=shaper,bptin=voice,
                        bptout=ia5,out-info="removebp Voice"
                        show="Voice -> Text mapper",cost=100,conv=loss

# **-- Logging for the rest of the prog --**
#
authlog level=notice, size=-1
operlog level=notice, size=400
normlog level=notice, size=400, sflags=zero, dflags=close

822flatten      normlog   file=filter
p2flatten       normlog   file=filter
822touk         normlog   file=filter
822tous         normlog   file=filter
dr2rfc          normlog   file=filter
fcontrol        normlog   file=filter
odif2txt        normlog   file=filter
p2explode       normlog   file=filter
rfc822norm      normlog   file=filter
trash           normlog   file=filter

822top2         normlog   file=rfc987
```

```
p2to822        normlog   file=rfc987

list           normlog   file=lists

qmgr           normlog   file=qmgr
qmgr−load      normlog   file=qmgr

smtp           normlog   file=smtp
smtpsrvr       normlog   file=smtpsrvr

greyout        normlog   file=greyout
gb−janet       normlog   file=greyin
submit         normlog   file=submit

local          normlog   file=local
msg−clean      normlog   file=msg−clean

x40084         normlog   file=x400in84,level=pdus
x400in84       normlog   file=x400in84,level=pdus
#x400in84      isodelog rtsaplevel file=rts,level=all
#x400in84      isodelog ssaplevel file=ssap,level=all
x400out84      normlog   file=x400out84

x400in88       normlog   file=x400in88 level=pdus,level=trace
x400out88      normlog   file=x400out88 level=pdus
xx400out88     normlog   file=x400out88 level=pdus level=trace

p7server       normlog   file=p7server level=trace
xp7server      normlog   file=p7server level=trace level=pdus

xfax           normlog   file=fax level=debug
xfaxd          normlog   file=fax #level=debug
G3Fax          normlog   file=fax
ps250          normlog   file=fax
hdr2fax        normlog   file=fax
ia52fax        normlog   file=fax
fax            normlog   file=fax
txttofax       normlog   file=fax
822tofax       normlog   file=fax

#x400out88 isodelog      rtsaplevel file=rts−out          level=all
#x400out88 isodelog      acsaplevel file=acsap−out level=all
#x400out88 isodelog      psap2level file=psap2−out level=all
#x400out88 isodelog      ssaplevel file=ssap−out          level=all


#x400in88 isodelog       rtsaplevel file=proto−in level=all
#x400in88 isodelog       acsaplevel file=proto−in level=all
#x400in88 isodelog       psap2level file=proto−in level=all

pptsapd        normlog   file=pptsapd, dlevel=notice
dirlist        normlog   file=dirlist
mslocal        normlog file=mslocal
mslocal−beta   normlog file=mslocal
mslocalbeta    normlog file=mslocal
xua            normlog level=trace file=p7server

x40084         normlog              file=x400in84 level=pdus, level=trace
x40084         isodelog             rtsaplevel file=x400in84 level=pdus
x40084         isodelog             ssaplevel file=x400in84 level=pdus
xx400in84      normlog              file=x400in84 level=pdus, level=trace
xx400in84      isodelog             rtsaplevel file=x400in84 level=pdus
#xx400in84     isodelog             ssaplevel file=x400in84 level=pdus
```

xx400out84	**normlog**	**file**=x400out84 **level**=**trace**
xx400out84	**isodelog**	rtsaplevel **file**=x400out84 **level**=pdus

Figure B.1: An Example Tailor File

Appendix C

Example of the PP Start Script

What follows is an example of a script which is run to start the PP mail system.

```
#! /bin/sh

# SMTP running under inetd
#cd /usr/pp/pp/lib/chans
#if [ -f smtpd -a -f smtpsrvr ]; then
#		(./smtpd `pwd`/smtpsrvr smtp > /dev/null 2>&1 ) &
#		echo -n ' smtp'
#fi

cd /usr/pp/pp/lib
su pp <<EOF
echo -n ' pp.startaspp :  '
if [ -f pptsapd ]; then
		echo -n ' pptsapd '
		pptsapd > /dev/null 2>&1
fi
if [ -f qmgr ]; then
		echo -n ' qmgr '
		qmgr > /dev/null 2>&1
fi
echo '.'
EOF
```

Figure C.1: An Example Start Script

Appendix D

Example of the PP Script Run Nightly

What follows is an example of a script which is regularly run to keep the PP mail system up to date.

```
#! /bin/sh

PATH=/crg/pp/bin:/crg/pp/cmds/tools:/usr/local/bin:/usr/ucb:/bin:/usr/bin:/usr/5bin:
export PATH
#
# Shell script run nightly to collect up things etc.
#
set −x
cd /crg/pp
L=logs
T=tables
STATDIR=/crg/pp/logs/statistics

exec 1> $L/pp−nightly.log 2>&1

freespace . 2000 || { echo NO space left on device; exit 1; }

#Save the stats files
if [ −f $L/stat ]
then
        DATE=‘date +%h−%d.%T‘
        [ −d $L/tmp ] || mkdir $L/tmp
        mv $L/stat $L/tmp/stat.$DATE
fi

# And then once a week...
if [ ‘date +%w‘ −eq 0 ]
then
        YDATE=‘date "+%h-%d-%y‘
        [ -d $STATDIR ] || mkdir $STATDIR
        if pstat $L/tmp/* /dev/null | bin/stat.awk > $STATDIR/stats.$YDATE
        then
                echo 'Sucessful - removing old files'
                rm $L/tmp/*
        fi
fi

# This should be first building command, as it updates the DERFIL2 file
echo 'building the PP tables'
(cd tables; make install)
```

```
# update the niftp stuff
echo 'building the niftp database'
(cd niftp; make )

echo UPDATE complete
```

Figure D.1: An Example Nightly Script

Appendix E

Obtaining X.400 Tables

The following tables are necessary for correct X.400 operation. These tables should be obtained from UCL. The tables are:

`or`: The O/R Table.

`rfc2or`: The RFC 822 to O/R mapping table

`or2rfc`: The O/R to RFC 822 mapping table.

`rfc1148gate`: The RFC 1148 known gateways file.

These tables can be fetched in a number of ways.

FTP: If you have access to Internet FTP then use anonymous FTP to cs.ucl.ac.uk [128.16.5.31] to retrieve the files from the directory *pp-tables/*. For example:

```
% ftp cs.ucl.ac.uk
Connected to bells.cs.ucl.ac.uk.
220 bells.cs.ucl.ac.uk FTP server (SunOS 4.0) ready.
Name (cs.ucl.ac.uk:pp): anonymous
331 Guest login ok, send ident as password.
Password:
230 Guest login ok, access restrictions apply.
ftp> get pp-tables/or or
...
ftp> get pp-tables/or2rfc or2rfc
...
ftp> close
%
```

FTAM: If you have access to FTAM on Janet, PSS or the Internet, you may obtain the tables from UCL using one of the following:

- Janet DTE 000005111600013 with TSEL 259 (ascii-encoded)
- PSS DTE 23421920030013 with TSEL 259 (ascii-encoded)
- Internet address cs.ucl.ac.uk ([128.16.5.31]) port 102 with TSEL 259 (ascii-encoded)

Use the "anon" user-identity and retrieve the files in the *pp-tables/* directory. For example:

```
% ftam ucl
user (ucl:pp): anon
ucl... connected
ANONymous user permitted, access restrictions apply
ucl> get pp-tables/or /tmp/or
ftam: unstructured text file transfer
ftam: 5792 bytes received in 5.88 seconds (0.96 Kbytes/s)
...
ucl> close
%
```

NIFTP: If you run NIFTP over the public X.25 or over Janet, and are registered in the NRS at salford, you can use NIFTP with username "guest" and your name as password, to access UK.AC.UCL.CS to retrieve the files found under `<pp-tables>`. For example:

```
% cpf '<pp-tables>or'@uk.ac.ucl.cs or
%
```

MAIL: The files may be obtained by mail by sending a suitable query to the address `info-server@cs.ucl.ac.uk`. This query should consist of a message body with the lines:

```
Request: pp-tables
Topic: or
Topic: rfc2or
Topic: or2rfc
Topic: rfc1148gate
```

E.g.:

```
% mail info-sver@cs.ucl.ac.uk <<EOF
Request: pp-tables
Topic: or
Topic: rfc2or
Topic: or2rfc
Topic: rfc1148gate
EOF
%
```

Appendix F

X.400 Coordination Service

The international academic and research community has been operating a stable X.400 service for over 6 years. This community currently uses static tables to document X.400 routing and MTA connection information. A central project team at SWITCH in Switzerland coordinates the service. The project team coordinates and distributes the international RFC1327 mapping tables (i.e., the *rfc2or*, *or2rfc*, and *rfc1148gate* tables.

If you are starting to run X.400 as an academic or research site (or over the Internet), you should contact the project team to join the service. If others from your country are already participating in the service, the project team will put you in contact with your national coordinator(s). Otherwise they will provide information on how you can join the service as a new country. You can contact them at:

```
SWITCH
MHS Coordination Service
Limmatquai 138
CH-8001 Zurich
Switzerland

Tel:    +41 1 2623143
Fax:    +41 1 2618133
Email X.400: S=project-team; O=switch; P=switch; A=arcom; C=ch;
     RFC822: project-team@switch.ch
```

The project team maintains an information server with the international routing, mapping, MTA connection information, documents, reports, etc. You can access the server the following ways:

```
email
 address:  RFC822: mhs-server@nic.switch.ch
           X.400:  S=mhs-server; OU1=nic; O=switch; P=switch; A=arcom; C=CH
 body:     help                          # you receive this document
           index ['directory']           # you receive a directory listing
           send 'directory''filename'    # you receive the specified file

FTP
 address:  Internet: nic.switch.ch
 account:  cosine
```

```
 password: 'your email address'

interactive
 address:  Internet: nic.switch.ch
 address:  PSPDN:    +22847971014540
 address:  EMPB/IXI: 20432840100540
 account:  info
 directory: e-mail/COSINE-MHS/

FTAM
 address:  Internet: nic.switch.ch
 address:  PSPDN   : +22847971014540
 address:  EMPB/IXI: 20432840100540
 address:  ISO CLNS: NSAP=39756f11112222223333aa0004000ae100, TSEL=0103Hex
 account:  ANON

gopher
 address:  Internet: nic.switch.ch
```

Appendix G

X.400 Connection Logs

The following are log samples of X.400 connection attempts, both 1984 and 1988. A variety of failure modes are demonstrated.

The Originating MTA refers to the MTA that initiated the connection. The Receiving MTA refers to the MTA which received the incoming connection. In most cases, logs are shown from both MTAs. First the logs of the Originating MTA are shown, then two blank lines are inserted, and then the logs of the Receiving MTA are shown.

Three general failure modes are identified. These are

(a) The Receiving MTA does not recognize the calling address of the Originating MTA. The Receiving MTA rejects the connection.

(b) The Receiving MTA does not match the MTA name or password of the Originating MTA. The Receiving MTA rejects the connection.

(c) The Originating MTA does not match the returned MTA name or password of the Receiving MTA. The Originating MTA rejects the connection.

A complete description of the sequence of events of establishing X.400 connections is included in section 5.3.

```
==== successful connection attempt (84) ==============================
==== (only Originating MTAs logs shown) ==============================

 3/26  2:38:36 x400out8 11037 (yee     )  Starting x400out84 (X400 outbound (198
4))
 3/26  2:38:37 x400out8 11037 (yee     )  Reading message 'msg.11032-0'
 3/26  2:38:37 x400out8 11037 (yee     )  Connecting to site atlas.arc.nasa.gov
 3/26  2:38:39 x400out8 11037 (yee     )  Sender '/S=yee/OU=spectre/O=NASA/PRMD=
ARC/ADMD=TELEMAIL/C=US/'
 3/26  2:38:39 x400out8 11037 (yee     )  Recipient  '/S=yee/O=NASA/PRMD=ARC/ADM
D=TELEMAIL/C=US/'
 3/26  2:38:41 x400out8 11037 (yee     )  Transfer Completed: 509 bytes in 1.34
seconds (0.37 Kbytes/s)
```

```
 3/26  2:38:41 x400out8 11037 (yee     )  >>> User-Mpdu msg.11032-0 transfered t
o atlas.arc.nasa.gov
 3/26  2:38:41 x400out8 11037 (yee     )  Connection successfully terminated


==== successful connection attempt (88) ==============================
==== (only Originating MTAs logs shown) ==============================

 3/26  2:49:45 x400out8 11136 (yee     )  Starting x400out88 (X400 outbound (198
8))
 3/26  2:49:45 x400out8 11136 (yee     )  Reading message 'msg.11076-0'
 3/26  2:49:45 x400out8 11136 (yee     )  Connecting (RTS 88 mode) to site atlas
.arc.nasa.gov
 3/26  2:50:40 x400out8 11136 (yee     )  Connected Sucessfully
 3/26  2:50:40 x400out8 11136 (yee     )  Sender '/S=yee/OU=spectre/O=NASA/PRMD=
ARC/ADMD=TELEMAIL/C=US/'
 3/26  2:50:40 x400out8 11136 (yee     )  Recipient Address '/S=yee/O=NASA/PRMD=
ARC/ADMD=TELEMAIL/C=US/'
 3/26  2:50:42 x400out8 11136 (yee     )  Transfer Completed: 734 bytes in 1.74
seconds (0.41 Kbytes/s)
 3/26  2:50:42 x400out8 11136 (yee     )  >>> Message msg.11076-0 transfered to
atlas.arc.nasa.gov


==== failure mode (a) - Initiating MTA's TSAP not known (84) =========
==== (rejected by Receiving MTA) =====================================

 3/26  2:56:43 x400out8 11193 (yee     )  Starting x400out84 (X400 outbound (198
4))
 3/26  2:56:43 x400out8 11193 (yee     )  Reading message 'msg.11187-0'
 3/26  2:56:43 x400out8 11193 (yee     )  Connecting to site atlas.arc.nasa.gov
 3/26  2:56:44 x400out8 11193 (yee     )  Unable to connect to site atlas.arc.na
sa.gov (reason Validation failure)
 3/26  2:56:44 x400out8 11193 (yee     )  Connection not made


 3/26  2:56:45 x400in84 00533 (yee     )  Starting x400in84 (X400 inbound (1984)
)
 3/26  2:56:46 x400in84 00533 (yee     )  Can't locate entry for ""591"/Internet
=128.102.128.58" ("591"/TELEX+00728722+RFC-1006+03+128.102.128.58)


==== failure mode (a) - Initiating MTA's TSAP not known (88) =========
==== (rejected by Receiving MTA) =====================================

 3/26  2:53:15 x400out8 11161 (yee     )  Starting x400out88 (X400 outbound (198
8))
```

```
 3/26  2:53:15 x400out8 11161 (yee     )  Reading message 'msg.11156-0'
 3/26  2:53:15 x400out8 11161 (yee     )  Connecting (RTS 88 mode) to site atlas
.arc.nasa.gov
 3/26  2:53:16 x400out8 11161 (yee     )  Association rejected: [Rejected by res
ponder]
 3/26  2:53:16 x400out8 11161 (yee     )  Connection not made


 3/26  2:53:18 x400in88 00526 (yee     )  Starting x400in88 (X400 inbound (1988)
)
 3/26  2:53:18 x400in88 00526 (yee     )  Can't locate entry for ""597"/Internet
=128.102.128.58" ("597"/TELEX+00728722+RFC-1006+03+128.102.128.58)
 3/26  2:53:18 x400in88 00526 (yee     )  Can't locate host params


==== failure mode (b) - Initiating MTA's MTAname not matched (84) ====
==== (rejected by Receiving MTA) =====================================

 3/26  3:04:01 x400out8 11263 (yee     )  Starting x400out84 (X400 outbound (198
4))
 3/26  3:04:02 x400out8 11263 (yee     )  Reading message 'msg.11252-0'
 3/26  3:04:02 x400out8 11263 (yee     )  Connecting to site atlas.arc.nasa.gov
 3/26  3:04:03 x400out8 11263 (yee     )  Unable to connect to site atlas.arc.na
sa.gov (reason Validation failure)
 3/26  3:04:03 x400out8 11263 (yee     )  Connection not made


 3/26  3:04:05 x400in84 00549 (yee     )  Starting x400in84 (X400 inbound (1984)
)
 3/26  3:04:05 x400in84 00549 (yee     )  parameter_checks/MTAName mismatch: spe
cified='spectre.arc.nasa.gov'  received='specter.arc.nasa.gov'
 3/26  3:04:05 x400in84 00549 (yee     )  Parameter check failed


==== failure mode (b) - Initiating MTA's MTAname not matched (88) ====
==== (rejected by Receiving MTA) =====================================

 3/26  3:12:56 x400out8 11363 (yee     )  Starting x400out88 (X400 outbound (198
8))
 3/26  3:12:56 x400out8 11363 (yee     )  Reading message 'msg.11358-0'
 3/26  3:12:56 x400out8 11363 (yee     )  Connecting (RTS 88 mode) to site atlas
.arc.nasa.gov
 3/26  3:13:01 x400out8 11363 (yee     )  Association rejected: [Rejected by res
ponder]
 3/26  3:13:02 x400out8 11363 (yee     )  Connection not made


 3/26  3:13:03 x400in88 00607 (yee     )  Starting x400in88 (X400 inbound (1988)
)
 3/26  3:13:03 x400in88 00607 (yee     )  MTAName mismatch: specified='spectre.a
```

```
rc.nasa.gov' received='specter.arc.nasa.gov'
 3/26  3:13:03 x400in88 00607 (yee     )  Parameter check failed
 3/26  3:13:03 x400in88 00607 (yee     )  Host authentication failed


==== failure mode (b) - Initiating MTA's Password not matched (84) ===
==== (rejected by Receiving MTA) =====================================

 3/26  3:18:32 x400out8 11440 (yee     )  Starting x400out84 (X400 outbound (198
4))
 3/26  3:18:32 x400out8 11440 (yee     )  Reading message 'msg.11436-0'
 3/26  3:18:32 x400out8 11440 (yee     )  Connecting to site atlas.arc.nasa.gov
 3/26  3:18:35 x400out8 11440 (yee     )  Unable to connect to site atlas.arc.na
sa.gov (reason Validation failure)
 3/26  3:18:35 x400out8 11440 (yee     )  Connection not made


 3/26  3:18:36 x400in84 00615 (yee     )  Starting x400in84 (X400 inbound (1984)
)
 3/26  3:18:37 x400in84 00615 (yee     )  parameter_checks/Password mismatch: sp
ecified=' ' received='foo'
 3/26  3:18:37 x400in84 00615 (yee     )  Parameter check failed


==== failure mode (b) - Initiating MTA's Password not matched (88) ===
==== (rejected by Receiving MTA) =====================================

 3/26  3:16:06 x400out8 11401 (yee     )  Starting x400out88 (X400 outbound (198
8))
 3/26  3:16:06 x400out8 11401 (yee     )  Reading message 'msg.11358-0'
 3/26  3:16:07 x400out8 11401 (yee     )  Connecting (RTS 88 mode) to site atlas
.arc.nasa.gov
 3/26  3:16:10 x400out8 11401 (yee     )  Association rejected: [Rejected by res
ponder]
 3/26  3:16:10 x400out8 11401 (yee     )  Connection not made


 3/26  3:16:12 x400in88 00611 (yee     )  Starting x400in88 (X400 inbound (1988)
)
 3/26  3:16:12 x400in88 00611 (yee     )  Password mismatch: specified=' ' recei
ved='foo'
 3/26  3:16:12 x400in88 00611 (yee     )  Parameter check failed
 3/26  3:16:12 x400in88 00611 (yee     )  Host authentication failed


==== failure mode (c) - Receiving MTA's MTAname not matched (84) =====
==== (rejected by Originating MTA) ===================================

 3/26  3:22:30 x400out8 11466 (yee     )  Starting x400out84 (X400 outbound (198
4))
```

```
 3/26  3:22:32 x400out8 11466 (yee     )  Reading message 'msg.11450-0'
 3/26  3:22:32 x400out8 11466 (yee     )  Connecting to site atlas.arc.nasa.gov
 3/26  3:22:33 x400out8 11466 (yee     )  parameter_checks/MTAName mismatch: spe
cified='atlas.arc.nasa.gov' received='batlas.arc.nasa.gov'
 3/26  3:22:33 x400out8 11466 (yee     )  Connection successfully terminated


 3/26  3:22:35 x400in84 00626 (yee     )  Starting x400in84 (X400 inbound (1984)
)
 3/26  3:22:35 x400in84 00626 (yee     )  Connection successfully terminated


==== failure mode (c) - Receiving MTA's MTAname not matched (88) =====
==== (rejected by Originating MTA) ===================================

 3/26  3:25:43 x400out8 11484 (yee     )  Starting x400out88 (X400 outbound (198
8))
 3/26  3:25:43 x400out8 11484 (yee     )  Reading message 'msg.11358-0'
 3/26  3:25:43 x400out8 11484 (yee     )  Connecting (RTS 88 mode) to site atlas
.arc.nasa.gov
 3/26  3:25:44 x400out8 11484 (yee     )  mta name mismatch batlas.arc.nasa.gov
atlas.arc.nasa.gov
 3/26  3:25:44 x400out8 11484 (yee     )  Connection not made


 3/26  3:25:46 x400in88 00632 (yee     )  Starting x400in88 (X400 inbound (1988)
)
 3/26  3:25:46 x400in88 00632 (yee     )  Normal disconnect


==== failure mode (c) - Receiving MTA's Password not matched (84) ====
==== (rejected by Originating MTA) ===================================

 3/26  3:29:36 x400out8 11526 (yee     )  Starting x400out84 (X400 outbound (198
4))
 3/26  3:29:36 x400out8 11526 (yee     )  Reading message 'msg.11523-0'
 3/26  3:29:36 x400out8 11526 (yee     )  Connecting to site atlas.arc.nasa.gov
 3/26  3:29:37 x400out8 11526 (yee     )  parameter_checks/Password mismatch: sp
ecified=' ' received='bar'
 3/26  3:29:37 x400out8 11526 (yee     )  Connection successfully terminated


 3/26  3:29:39 x400in84 00640 (yee     )  Starting x400in84 (X400 inbound (1984)
)
 3/26  3:29:39 x400in84 00640 (yee     )  Connection successfully terminated


==== failure mode (c) - Receiving MTA's Password not matched (88) ====
==== (rejected by Originating MTA) ===================================
```

```
 3/26  3:28:12 x400out8 11510 (yee     )  Starting x400out88 (X400 outbound (198
8))
 3/26  3:28:12 x400out8 11510 (yee     )  Reading message 'msg.11358-0'
 3/26  3:28:12 x400out8 11510 (yee     )  Connecting (RTS 88 mode) to site atlas
.arc.nasa.gov
 3/26  3:28:22 x400out8 11510 (yee     )  passwd mismatch bar
 3/26  3:28:22 x400out8 11510 (yee     )  Connection not made


 3/26  3:28:24 x400in88 00636 (yee     )  Starting x400in88 (X400 inbound (1988)
)
 3/26  3:28:24 x400in88 00636 (yee     )  Normal disconnect


==== This example is from version PP 6.0 with X.400(84) ==============
==== Receiving MTA running in sloppy mode, unknown TSAP accepted anyway

12/21 16:09:09 x400out8 12084 (pp      )  Starting x400out84 (X400 outbound)
12/21 16:09:09 x400out8 12084 (pp      )  Connecting to site calypso-mta
12/21 16:09:12 x400out8 12084 (pp      )  Connection from key='calypso-mta'  mta
='calypso.cs.wisc.edu'  password=' '
12/21 16:09:12 x400out8 12084 (pp      )  Connection successfully terminated


12/21 16:09:10 x400in84 24321 (#2094   )  Starting x400in84 (X400 inbound)
12/21 16:09:11 x400in84 24321 (#2094   )  Connection from key='"591"/TELEX+00728
722+RFC-1006+03+128.105.8.53'  key-pretty='"591"/Internet=128.105.8.53'  mta='relay.xnr
12/21 16:09:11 x400in84 24321 (#2094   )  No information found in the X.400 inco
ming tables - BUT have accepted the connection
12/21 16:09:13 x400in84 24321 (#2094   )  Connection successfully terminated
```

Bibliography

[1] Kevin J. Dunlap. *Name Server Operations Guide for BIND*. University of California, Berkeley. BSD 4.3 Release.

[2] Stephen E. Kille. A string encoding of Presentation Address. Request for Comments RFC 1278, Department of Computer Science, University College London, November 1991.

[3] Stephen E. Kille. X.400 1988 to 1984 downgrading. Request for Comments 1328, DDN Network Information Center, SRI International, May 1992.

[4] Steve Kille. PP - a message transfer agent. In *IFIP WG 6.5 Conference on Message Handling Systems and Distributed Applications*, pages 115–118. North Holland Publishing, October 1988.

[5] Marshall T. Rose and Einar A. Stefferud. Proposed Standard for Message Encapsulation. Request for Comments 934, DDN Network Information Center, SRI International, January 1985.

[6] Marshall T. Rose. *Changes to the Rand MH Message Handling System: MH #6.5 for 4.3BSD UNIX*, May 1986. MH documentation set.

[7] Marshall T. Rose. Post Office Protocol — Version 3. Request for Comments 1081, DDN Network Information Center, SRI International, December 1988.

[8] Marshall T. Rose and Dwight E. Cass. ISO Transport Services on top of the TCP. Request for Comments 1006, DDN Network Information Center, SRI International, May 1987.