`ROOTUSER`: The super user.

`CHOWN`: The *chown* (8) program. This may need to be a full pathname.

`CHMOD`: The *chmod* (1) program. This is normally ok as just `chmod`.

`BACKUP`: When PP installs a program, it normally takes a copy of the existing program for safety. If you wish this behaviour then you should set this variable to *cp* (1). If not, setting it to something harmless like ":" or *echo* (1) which will disable this action.

`INSTALL`: The command used to install a new binary. This is normally either *cp* (1) or *install* (1).

2.1.2 h/config.h

The only other tailoring that should be done before compiling is setting of `#define`s in the file *h/config.h*. These affect the more global options and there are very few of these.

`DBM/NDBM`: One of these values must be defined. It is *strongly* advised that you use *ndbm* (3) if you have it at all; the *dbm* (3) routines can create havoc if you are using another package that uses dbm too. If you are certain you don't have *ndbm* (3) then use *dbm* (3) and add `-ldbm` to your `LIBSYS` make variable.

`PP_DEBUG`: This variable should be set to one of the following:

Level	Meaning
`PP_DEBUG_ALL`	Include all debugging code
`PP_DEBUG_SOME`	Include basic debugging code
`PP_DEBUG_NONE`	Don't include any debugging code

`NAMESERVER`: If you wish to make use of the BIND nameserver for the SMTP channels, you should define this.

`HAS_FSYNC`: This variable should be defined if your system has the *fsync* (2) system call. This is defined for you if the system is BSD based.

`UKORDER`: This variable should be defined if you want user tools to use UK RFC-822 domain ordering by default.

`VAT`: This variable includes some minor extra features.

2.2 Overview of Channels

This is a brief overview of the channels distributed with PP. The binaries of these channels are installed in the *CHANDIR* directory.

2.2.1 Mandatory Channels

The channels under this section must be installed with all configurations of PP. They are mostly "housekeeping" channels used by the *qmgr*.

qmgrload: The *qmgr* maintains its own model of the queue, separate to the master model stored under the *quedir* directory. The *qmgrload* channel is used to initialise the *qmgr*'s model when the *qmgr* starts up. When running, the *qmgr* periodically uses *qmgrload* to check the consistency between its model and reality.

msg-clean: After all work has been done on a message and that message is no longer of interest to PP, the *qmgr* removes the message from the *quedir* directory via the *msg-clean* channel.

trash: From time to time, the *qmgr* invokes the *trash* channel. This channel hunts around the *quedir* directory for files and directories that are no longer of interest to PP and removes them.

timeout: When a message has been in the queue for too long, the *qmgr* uses the *timeout* channel to send a message back to its originator, telling him or her that PP timed out the original message.

dr2rfc: The *dr2rfc* channel is used to construct delivery receipt notifications or error messages for channels that cannot handle these types of messages directly. It converts delivery reports into standard RFC 822 messages which can then be sent back to the sender.

warning: The *warning* channel is used to send warning reports which informat senders of delay in delivering messages.

splitter: The *splitter* channel is used by *submit* when processing for a message has to be done on a per–recipient basis. The *splitter* channel splits a multi–recipient message into single recipient messages via resubmission.

2.2.2 Optional Channels

The channels under this section are optional. To see how they are combined to produce various configurations see Section 2.3.

822-local: The *822-local* channel is used to deliver messages to local users in RFC 822 format. It can deliver by a number of methods. For more details see Section 4.12 on page 73 and also *The PP Manual: Volume 3 – Users Guide*.

slocal: The *slocal* channel (structured local channel) should be used if local users are to receive structured messages, that is, if the MTA is to deliver messages made up of different parts, such as multi-media messages, which cannot be appended to a single file but which must be delivered as a directory of files. The *users* table will indicate, for each user, whether *slocal* is an optional channel.

> **NOTE:** This channel is still experimental and not designed for an operational configuration.

X400-84: Inbound and outbound channels for handing X.400 P1(84) messages are available. The channels will run over any media that ISODE can support transport connections on.

X400-88: Inbound and outbound channels for handing X.400 P1(88) messages are available. The channels will run over any media that ISODE can support transport connections on. This software should be considered as "*beta test*" software in the current release.

SMTP: An *smtp* channel is available for use over TCP/IP. It may be used over networks such as a local Ethernet or WAN such as the DARPA Internet.

JNT Mail: The JNT mail channel is used over X.25 networks, particularly in the U.K. The unix-niftp package is used to provide the blue book file transfer part of this.

UUCP: A channel that will interface with the UUCP protocols.

Note: This channel has not been tested extensively by the developers. Others have reported that it is satisfactory, however.

DECnet **MAIL11:** A channel that implements the DECnet protocol. This is contributed software.

p2explode: The *p2explode* channel is a restructuring channel that is used to explode a general p2 message into its separate component bodyparts. Note the same binary can explode p2(84) and p22(88). It is used on an incoming X.400 message when the MTA needs to manipulate the separate components of that message.

p2flatten: The *p2flatten* channel is a restructuring channel that is used to combine all the separate components of a message into one p2 file. It is the exact inverse of *p2explode*.

P2toRFC: The *P2toRFC* channel is a reformating channel that is used to convert an X.400 style header encoded in `p2(84)` or `p2(88)` to an RFC 822-style header. This conversion is performed according to RFC 1148[Kil89b] .

RFCtoP2: The *RFCtoP2* channel is a reformating channel that is used to convert an RFC 822 style header to an X.400 style header encoded in `p2(84)` or `p2(88)`. This conversion is done as described in RFC 1148.

rfc934: The *rfc934* channel flattens an RFC 822-style message which may contain forwarded messages, into a flat RFC 822 message. The flattening is done as described in RFC 934[Mar85].

list: The *list* channel expands a distribution list into its component members. The original message is then sent to each of these members. The channel uses the associated *list* table as described in Section 4.15 on page 77.

dirlist: The *dirlist* channel expands a distribution list into component members. It takes the list of members from the X.500 directory.

shell: The *shell* channel runs a specific process for the originator of the message. This allows delivery of a message to a special process rather than to a user. The mapping from the address the message is sent to and the process to run is described by the *shell* table, see Section 4.16 on page 78.

splitter: The *splitter* chanel is used in some instances to split a message into separate messages with one recipient each. This allows some specialist reformatting to be done. See details of the FAX driver for an example.

fcontrol: The *fcontrol* channel is used to run various filters on the bodyparts of the message. These filters can only affect the contents of the bodyparts and cannot affect the overall structure of the message. The filters are usually installed in the *FORMDIR* directory.

Here is a brief description of the filters distributed with PP:

`rfc822norm:` The filter is used for conversion between various styles of RFC 822 headers .

`p2norm:` The filter is used to normalise X-400 addresses in P2 headers. It can also be used to downgrade X-400 (88) P2 headers into X-400 (84) P2 headers. This downgrading is done according to the INTERNET draft "X.400 1988 to 1984 downgrading" by S.E. Hardcastle-Kille.

`removebp:` The filter is a simple shell script used to replace bodyparts that the MTA cannot support. It removes the indicated part and replaces it with a simple text message reflecting this fact.

2.3 Example Configurations

This section gives some details about the example configurations included in the distribution. These configurations should be looked on as a sample first step to allow a working system to be installed easily. This should then be evolved to meet local requirements. To use one of these configurations, do the following.

1. Change to the *examples* directory.
2. Select one of the configurations shown below, and change into that directory.
3. Read the *README* file, and carry out the instructions given in there.

This should give you a *very basic* working system.

Local SMTP

This configuration assumes a simple host doing local deliveries and passing all other messages to another host. It requires the *smtp* channel, the *822-local* channel and optionally the *list* channel.

SMTP mail

This configuration assumes a simple host doing SMTP relaying and local delivery only. It is similar to the above, but can handle Internet connectivity by use of the domain nameserver.

Janet mail

This host is assumed to be on the Janet network and has local SMTP capability. It is a skeleton only, as for proper operation, NRS data is necessary to identify the necessary domain and routes.

2.4 Configuration of User Agents

PP is a message transfer agent, and does not contain a sophisticated X.400 user agent. It is hoped that the interface documented in Volume 2 of these manuals will be useful in the development of new user agents or the interfacing of existing UAs. The latter should be straightforward for most RFC 822-based interfaces.

MH[Ros86] can be used directly with PP if compiled with the SMTP transfer interface.

PP provides a */bin/mail* like interface. This should be installed in the normal installation. It provides a simple way for submission of RFC 822 messages, and allows a few flags to be set. Also provided is a *sendmail*(8) emulation program that can be used to intercept those programs that insist on calling sendmail directly (e.g., *cron*(8)).

2.5 Checking the Configuration of Your System

Following installation, it is *strongly* recommended to use *ckconfig* (8) to ensure that the PP directory structure is correct and all the channel programs are correctly in place. *ckconfig* is described in Section 7.5.2.

Chapter 3

Configuring PP

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 A 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.

3.1 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.

3.2 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.

3.2.1 Mandatory Tailoring Variables

The following tailoring variables *must* be included in the *tailor* file:

`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 1148 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.

`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 altrered, this file should be removed.

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

`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.

`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: `Internet=localhost+20001`. This is suitable if you are an internet site, but not if you have just your own ethernet which is not connected to the internet. In this case a more suitable value might be `LOCAL-ETHER=localhost+20001` or `localHost=20001`. (See the ISODE manual for more details on defining address communities.)

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`).

3.2.2 Optional Tailoring Variables

The tailoring variables in this section are less critical, having defaults that are normally suitable.

`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.

`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.

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

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

`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*.

`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.

`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*.

`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.

`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.

`authchannel:` This gives the default policy for channel authorisation; it is in the form of a *channel-auth* record (see Section 4.10). The default is `block`. This determines the policy for any inbound/outbound channel pairing not explicitly entered in the channel authorisation table *channel-auth*.

> Note: This feature is deprecated and will be removed in future. To set a default authorisation policy, use a construct `*->*` in the *auth.channel* table.

`authloglevel:` This sets the level of authorisation logging. Three levels are defined: `low`, `medium` and `high`. These levels are now little used and should probably not be set.

`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`.

`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.

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

The default is two warnings.

`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.

`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.

`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.

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

`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.

`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.

`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> ]
```

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 3.8.1 on page 34 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 ususal.

`dap_user:` This is an X.500 directory distinguished name used when PP binds to the directory. It is useful to use this in versions of ISODE greater than 6.8 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.