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.

7.5 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.

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.

`-p:` Check the address and allow parsing of percent symbols in the route.

`-a:` Normalise all domains.

`-l:` Normalise domains with the USA ordering (`littlendian`) preferred.

`-b:` Normalise domains with the UK ordering (`bigendian`) 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 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 1148 Mapping Tables

RFC 1148 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 1148 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 realistic 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 sub-domains 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[2]. 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.

[2] To obtain a copy of this, contact Piete Brooks at Cambridge Computer Laboratory (pb@cl.cam.ac.uk)

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:

```
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)
```

7.8 Adding New Users and MTAs

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.