sysmailfilter 18

T

table
  dbm 23
  domain 42
  file 23
  flags 23
  linear 23
  list 53
  name 23
  shell 54
  show 23
  users 41
tables 50
  alias 40
  channel 45
  channels 50
  X.400 O/R 43
tail 93
TAILOR 8
tailor 16, 71, 75, 89, 91, 93, 98
tailoring
  channels 23
  filters 98
  logs 20
  simple 17
  tables 23
  variables 17
tbldfldir 53, 89
TBLDIR 8
tbldir 17, 18, 23
timeout 11, 37
timout 78
tjoin 96, 97
/tmp 20
TOOLDIR 5, 75
Tools 54
tools 93, 100
trash 11
tsapd 100

U

user 41
users 12, 39, 40, 41, 47, 53, 54, 93, 97
USRBINDIR 8, 31
/usr/local/lib/mailfilter 18
uuencode 98

W

warninterval 19
wrndfldir 19

X

X11 10
X25 access 2
X.400 84
  X.400 88 50
x400in84 30

mailfilter 18
.mailfilter 31, 55
make 5
Make.defs 5, 16, 31
make-lists 54
MANDIR 9
MANOPTS 9
mboxname
    mailbox 18
mkpasswd 61
mlist 53, 54
moderator 53
msg 101
msg-clean 37, 78
MTAconsole 60, 83
mu 115, 116, 117, 118, 119
mu.defaults 115

N
ndbm 10
normlog 21
nsapexpand 52
ntail 93
nwarnings 19

O
oper 100
operlog 21
OPTIONALCHANS 8
or 39, 43, 45, 63, 71, 93
or2rfc 39, 48, 65, 71

P
p2explode 13, 37
p2flatten 13, 36, 37
P2toRFC 13
pepy 7
PEPY 10
PGMPROT 10
postmaster 18
posy 7, 10
pp 18
pp.channel 96
ppdbm 18
pp.domain 95
pp-gen 5
pplogin 18
pp.night 100
pp.start 30, 99, 100
pp-tables/ 113, 114
pptaspd_addr 18
pptsapd 18, 93, 99
PPUSER 10
probe 91

Q
qmgr 11, 17, 18, 29, 37, 38, 60,
    78, 80, 81, 93, 99
qmgrhost 17
qmgrload 11, 37
QUEDIR 8
quedir 11, 17

R
rc.local 99
README 14
regex 59
report.* 101
returntime 20
rfc1148gate 48
rfc2or 39, 48, 65, 71
rfc822norm 33, 34
rfc934 13, 36
RFCtoP2 13
ROOTUSER 10
rosy 7, 10
rsh 4

S
sendmail 3, 15, 55
shell 14, 42, 54, 55
slocal 12, 42
sloppy 30
smtp 12, 14, 30
smtpd 30
smtpsrvr 99
stat 100
statp 100
submit 22, 39, 45, 92, 98, 99,
    115, 116, 117, 119

ckconfig 15, 16, 89, 92, 93
CMDDIR 5
cmddir 17, 18
c-nrs 95, 96
config.h 5
console 93
cp 10
cron 15

D
dbm 2, 8, 10, 18
dbmbuild 17, 92, 97, 100
debris 37
default 52
delete 11
delim1 18
delim2 19
delimiters 18
dirlist 14
domain 39, 42, 45, 47, 48, 94
dr2rfc 11, 24

E
echo 10
/etc/inetd.conf 30
/etc/isoentities 29
/etc/isoservices 29
/etc/passwd 18, 50, 54
examples 14
exec 31

F
fcntl 20
fcontrol 14, 18, 25, 31, 32, 33, 34, 98
filter control 31
filtercontrol 5
flock 20
FORMDIR 5, 14
formdir 18, 32, 35

G
gcc 9

H
h/config.h 10
hostname 17

I
inetd 30, 99
inetd.conf 30
INSTALL 10
isode 19
isode-gen 100
isodelog 22
isotailor 19

K
kill 100

L
LaTeX 132
LDOPTIONS 10
LIBCCOPTIONS 9
Lib/pp/static.c 36
LIBRESOLV 8
LIBSYS 8
LIBX 10
linear 17
lint 7
LINT 10
LINTFLAGS 10
list 13, 14, 35, 36, 53
list channel 93
local channel 93
loc_dom_mta 17
loc_dom_site 17
lockf 20
lockstyle 20
loc_or 17
log
    dflags 21
    dlevel 21
    file 20
    level 20
    sflags 21
    size 21
LOGDIR 8
logdir 17, 20

M
.mail 18
mailbox 18
maildrop 18

Index

822-local 12, 14, 31, 41, 55
822top2 99
822touk 34
822tous 34

A

abbrev 95
addr 101
alias 97
aliases 39, 40, 47, 48, 93
argh.c 116
authchannel 19
auth.channel 56, 57, 58
authlog 21
authloglevel 19
auth.mta 56, 57, 58
auth.qmgr 61
auth.user 56, 57
awk 100

B

BACKUP 10
base 101
/bin/mail 3, 15, 55
bodypart 18

C

cc 9
CCOPTIONS 9
CHANDIR 5, 11
chandir 18
channel 29, 39, 42, 45, 50, 94, 96
    822-local 31
    access 26
    adr 25
    adr-order 25
    auth-table 26
    bpout 26
    bptin 26
    chanout 24
    content-in 24
    content-out 24
    conv 27
    cost 24
    domain-norm 26
    info 25
    maxproc 26
    mta 26
    name 24
    probe 27
    prog 24
    show 24
    sort 25
    subadr 25
    table 26
    type 24
    uucp 31
channel-auth 19
channels 50
chans 18
ch.list 53
ch.local 50
chmod 7
CHMOD 10
chndfldir 89
chown 7, 10
ch.shell 54
ch.x400in84
    ch.x400in88 52
ch.x400out84 50
ckadr 90, 92
ckchan 88

Bibliography

[Dun] Kevin J. Dunlap. *Name Server Operations Guide for BIND.* University of California, Berkeley. BSD 4.3 Release.

[Kil88] S.E. 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.

[Kil89] Stephen E. Kille. Mapping between X.400(1988)/ISO10021 and RFC822. Request for Comments 1148, DDN Network Information Center, SRI International, July 1989.

[Mar85] Marshall T. Rose and Einar A. Stefferud. Proposed Standard for Message Encapsulation. Request for Comments 934, DDN Network Information Center, SRI International, January 1985.

[RC87] 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.

[Ros86] Marshall T. Rose. *Changes to the Rand MH Message Handling System: MH #6.5 for 4.3BSD UNIX*, May 1986. MH documentation set.

[Ros88] Marshall T. Rose. Post Office Protocol — Version 3. Request for Comments 1081, DDN Network Information Center, SRI International, December 1988.

[Ros89] Marshall T. Rose. *The ISO Development Environment: User's Manual.* The Wollongong Group, 5.0 edition, March 1989.

END

Figure F.1: QMGR ROS

```
        info                [8] IA5String OPTIONAL                  450
        }

DeliveryStatus ::= SEQUENCE OF IndividualDeliveryStatus


IndividualDeliveryStatus ::=  SEQUENCE {
        recipient [0] RecipientId,
        status        [1] ENUMERATED {
                                        -- The QMGR should treat the following
                                        -- as successful.  In the latter cases460
                                        -- a DR is processed in conjuction
                                        -- with a different recipient
                success (0),
                successSharedDR (1),
                failureSharedDR (2),

                                        -- The next set are treated by the
                                        -- QMGR as now having to send a DR
                negativeDR (3),   -- negative DR
                positiveDR (4),         -- positive DR                  470


                                        -- the final set are different types
                                        -- of transient failure, needing
                                        -- different QMGR retry strategies
                messageFailure (5),-- temp failure on the Message (e.g.
                                        -- protocol temp reject)
                mtaFailure (6),         -- MTA failure (e.g. couldn't connect)
                mtaAndMessageFailure (7)  -- Both (e.g. connection bust)
                },                                                      480
        info[2] IA5String OPTIONAL
        }


QMGRControl ::= SEQUENCE {
        op [0] QMGROp
        }

QMGROp ::= ENUMERATED {
        abort (0),                                                      490
        gracefulTerminate (1),
        restart (2),            -- as if from cold
        rereadQueue (3),   -- but don't disguard curent information
        disableSubmission (4),  -- Don't let processes submit
        enableSubmission (5),
        disableAll (6),
        enableAll (7),
        increasemaxchans(8),
        decreasemaxchans(9) }
                                                                        500
```

```
        priority Priority }

ChannelInfo ::= SEQUENCE {                                              400
        channel             [0] Channel,
        channelDescription [1] PrintableString,
        oldestMessage       [2] UTCTime,
        numberMessages      [3] INTEGER,    -- not including DRs
        volumeMessages      [4] INTEGER,      -- bytes
        numberActiveProcesses [5] INTEGER,
        status              [6] ProcStatus,
        numberReports   [7] INTEGER,
        direction           [8] BITSTRING {
                inbound (0),            -- supports inbound traffic     410
                outbound (1)            -- supports outbound traffic
                                        -- both bits on => does both
                                        -- both bits off is internal
                },
        chantype            [9] ENUMERATED {
                mta (0),                -- is an MTA level channel
                mts (1),                -- is an MTS level channel
                internal (2),           -- is an internal channel (formatter etc)
                passive (3)             -- responding channel?
                },                                                      420
        maxprocs            [10] INTEGER DEFAULT 0
        }

MtaRead ::= SEQUENCE {
        channel Channel,
        time UTCTime OPTIONAL
        }

MsgRead ::= SEQUENCE {
        channel Channel,                                                430
        mta Mta,
        time UTCTime OPTIONAL
        }

PrioritisedMtaList ::= SEQUENCE OF PrioritisedMta

PrioritisedMta ::= SEQUENCE {
        mta MtaInfo,
        priority Priority }
                                                                        440
MtaInfo ::= SEQUENCE {
        channel             [0] Channel,
        mta                 [1] Mta,
        oldestMessage       [2] UTCTime,
        numberMessage       [3] INTEGER, -- not DRs
        volumeMessages      [4] INTEGER,
        status              [5] ProcStatus,
        numberDRs           [6] INTEGER,
        active              [7] BOOLEAN DEFAULT FALSE,
```

```
ProcStatus ::=      SEQUENCE {
        enabled             [0] BOOLEAN,
        lastAttempt         [1] UTCTime OPTIONAL,
        cachedUntil         [2] UTCTime OPTIONAL,                    350
        lastSuccess         [3] UTCTime OPTIONAL
        }

RecipientInfo ::= SEQUENCE {
        user                [0] User,
        id                  [1] RecipientId,
        mta                 [2] Mta,
        channelList         [3] SEQUENCE OF Channel,
        channelsDone    [4] INTEGER,
                                -- Number of channels processed     360
                                -- If EQUAL to the number of channels,
                                -- this implies DN is still pending
        procStatus          [4] ProcStatus OPTIONAL,
        info                [5] IA5String OPTIONAL }


FilterList ::= SEQUENCE OF Filter

Filter ::= SEQUENCE {
        contenttype         [0] ContentType OPTIONAL,                370
        eit                 [1] EncodedInformationTypes OPTIONAL,
                            -- message must have only specified types
        priority            [2] Priority OPTIONAL,
                            -- messages of this priority and above
        moreRecentThan      [3] UTCTime OPTIONAL,
        earlierThan         [4] UTCTime OPTIONAL,
        maxSize             [5] INTEGER DEFAULT 0,
                            -- size in bytes
        originator          [6] User OPTIONAL,
        recipient           [7] User OPTIONAL,                       380
        channel             [8] Channel OPTIONAL,
        mta                 [9] Mta OPTIONAL,
        queueid             [10] QID OPTIONAL,
        mpduiden            [11] MPDUIdentifier OPTIONAL,
        uaContentId         [12] UAContentId OPTIONAL }

ChannelReadResult ::= SEQUENCE {
        channels            [0] PrioritisedChannelList,
        load1               [1] INTEGER,
        load2               [2] INTEGER,                             390
        currchans           [3] INTEGER,
        maxchans            [4] INTEGER }

PrioritisedChannelList ::= SEQUENCE OF PrioritisedChannel

PrioritisedChannel ::= SEQUENCE {
        channel    ChannelInfo,
```

```
                    cacheClear [2] NULL,
                    cacheAdd [3] UTCTime }

ChannelControl ::=  SEQUENCE {
                    channel [0] Channel,
                    control [1] Control
                    }                                                    300

MtaControl ::= SEQUENCE {
                    channel [0] Channel,
                    mta [1] Mta,
                    control [2] Control
                    }

MsgControl ::= SEQUENCE {
                    qid [0] QID,
                    users [1] UserList,                                  310
                    control [2] Control
                    }

UserList ::= SEQUENCE OF RecipientId

RecipientId ::= INTEGER

MsgStruct ::= SEQUENCE {
        messageinfo         [0] PerMessageInfo,
        recipientlist       [1] SEQUENCE OF                              320
                                    RecipientInfo
                            -- only recipients with responsibility bit set
                            -- AND originator (recipient 0)
                            -- The originator is needed for DRs
        }


PerMessageInfo ::= SEQUENCE {
        queueid             [0] QID,
        mpduiden            [1] MPDUIdentifier,                          330
        originator          [2] User,
        contenttype         [3] ContentType OPTIONAL,
        eit                 [4] EncodedInformationTypes OPTIONAL,
        priority            [5] Priority,
        size                [6] INTEGER,
                            -- size in bytes
        age                 [7] UTCTime,
        warnInterval        [8] INTEGER,
        numberWarningsSent [9] INTEGER,
        expiryTime          [10] UTCTime,                                340
        deferredTime        [11] UTCTime OPTIONAL,
        uaContentId         [12] UAContentId OPTIONAL,
        errorCount          [13] INTEGER OPTIONAL,
        inChannel           [14] Channel OPTIONAL
        }
```

```
                    },
                    information[1] IA5String OPTIONAL
          }

-- DATA TYPES

Mta ::= IA5String

Channel ::= PrintableString                                          250

Priority ::= ENUMERATED {low(0), normal(1), high(2)}
                              -- Internal priorities COULD be different
                              -- to external ones

UAContentId ::= PrintableString

ContentType ::= PrintableString

EncodedInformationTypes ::= SEQUENCE OF PrintableString              260

QID ::= PrintableString

User ::= IA5String
                              -- QMGR just sees users as strings.
                              -- This is 822 or /= X.400 syntax
                              -- The caller must get the right form!


ReadMessageArgument ::= SEQUENCE {                                   270
                    filters  FilterList,
                    interval UTCTime OPTIONAL
                              -- interval for control of the LINKED operation
                    }

MsgList   ::= SEQUENCE {
                    msgs SEQUENCE OF
                              MsgStruct,
                    deleted SEQUENCE OF QID OPTIONAL
                    }                                                280

ProcMsg ::= SEQUENCE {
          qid QID,
          users UserList,
                              -- if USER 0, do all outstanding DNs
          channel Channel  -- Channel to apply
                    -- must be same for each user!
                    -- Warning and Expiry channels have "special" behaviour
}
                                                                     290

Control ::= CHOICE {
                    stop [0] NULL,
                    start [1] NULL,
```

```
                                                                    190
-- ERRORS

                        -- congestion at the responder
congested ERROR
        ::=             0

                        -- unknown channel
noSuchChannel ERROR
        ::=             1
                                                                    200
                        -- unknown operation
illegalOperation ERROR
        ::=             2

                        -- operation on mta makes no sense
mtaNotInQueue ERROR
        ::=             3

                        -- error in the protocol
protocol ERROR                                                      210
        ::=             4

authenticationFailure ERROR
        ::=             5

-- Bind Arguments

BindArgument ::= CHOICE {           -- structure used in association request
        noAuthentication [0] NULL,
        weakAuthentication [1] WeakAuthentication                   220
        }

WeakAuthentication ::= SEQUENCE {
        username [0] IA5String,
        passwd   [1] IA5String OPTIONAL
        }

BindResult ::= SEQUENCE {
                result[0] ENUMERATED {  -- result from the association request
                        acceptedLimitedAccess(0), -- accepted limited access
                                                                    230
                        acceptedFullAccess(1)       -- accepted full access
                },
                information[1] IA5String OPTIONAL,
                version[2] IA5String OPTIONAL
        }


BindError ::= SEQUENCE {
                reason[0] ENUMERATED {
                        badCredentials(0),          -- dont know this person
                                                                    240
                        congested(1)                -- some system problem
```

```
                                        -- read about a mta
                                        -- Console -> QMGR
mtaread OPERATION
        ARGUMENT MtaRead
        RESULT PrioritisedMtaList
        ERRORS { congested, protocol, noSuchChannel }
        LINKED {mtainfo}
        ::= 7

                                        -- QMGR -> Console
                                        -- Incremental update
mtainfo OPERATION
        ARGUMENT PrioritisedMtaList
        ::= 12

                                        -- control a mta
                                        -- Console -> QMGR
mtacontrol OPERATION
        ARGUMENT MtaControl
        RESULT MtaInfo
        ERRORS { congested, noSuchChannel, mtaNotInQueue,
                authenticationFailure}
        ::= 8

                                        -- control a message
                                        -- Console -> QMGR
msgcontrol OPERATION
        ARGUMENT MsgControl
        RESULT NULL
        ERRORS { congested, protocol, noSuchChannel, mtaNotInQueue,
                authenticationFailure}
        ::= 13

                                        -- QMGR -> Channel (process)
channelInitialise OPERATION
        ARGUMENT Channel
        RESULT NULL
        ERRORS { congested, protocol, noSuchChannel}
        ::= 9

qmgrControl OPERATION
        ARGUMENT QMGRControl
        RESULT NULL -- always works
        ERRORS { congested, protocol, authenticationFailure}
        ::= 14

readChannelMtaMessage OPERATION
        ARGUMENT MsgRead
        RESULT MsgList
        ERRORS { congested, protocol, noSuchChannel, mtaNotInQueue }
        ::= 15
```

```
                                    -- processes a message
                                    -- QMGR -> Channel
processmessage OPERATION
        ARGUMENT ProcMsg                                              90
        RESULT DeliveryStatus
        ERRORS { congested, protocol }
        ::= 2

                                    -- start up a channel
                                    -- Channel -> QMGR
channelbegin OPERATION
        ARGUMENT FilterList
                        -- type of message requested
        RESULT FilterList                                             100
                        -- what you are going to get
        ERRORS { congested, protocol }
        ::= 3

                                    -- read the entire queue
                                    -- QMGR -> Q reader
readqueue OPERATION
        ARGUMENT NULL
        RESULT MsgList
        ERRORS { congested, protocol }                                110
        ::= 4

                                    -- read a channel
                                    -- Console -> QMGR
                                    -- get info at channel level
channelread OPERATION
        ARGUMENT UTCTime
        RESULT ChannelReadResult
        ERRORS { congested, protocol }
        LINKED {chaninfo}                                             120
        ::= 5

                                    -- QMGR -> Console
                                    -- Incremental update
chaninfo OPERATION
        ARGUMENT ChannelReadResult
                        -- info on ALL channels
        ::= 11

                                    -- control a channels behaviour     130
                                    -- Console -> QMGR
channelcontrol OPERATION
        ARGUMENT ChannelControl
        RESULT PrioritisedChannelList
        ERRORS { congested, protocol, noSuchChannel, illegalOperation,
                 authenticationFailure}
        ::= 6
```

```
passiveChannel OBJECT
                PORTS { processmessage[S],
                        channelbegin[C] }
        ::= id-pt-passiveChannel

submitProcess OBJECT                                                    40
      PORTS { newmessage[C] }
        ::= id-pt-submitProcess

queueReadProcess OBJECT
      PORTS { readqueue[S] }
        ::= id-pt-queueReadProcess

managementUA OBJECT
                PORTS { processmessage[C],
                        readmsginfo[C],                                 50
                                msginfo[S],
                        channelread[C],
                                channelinfo[S],
                                channelcontrol[C],
                        mtaread[C],
                                mtainfo[S],
                        mtacontrol[C],
                                readchannelmtamessage[C] }
        ::= id-pt-managementUA
                                                                        60
-- OPERATIONS

                                        -- new message arrived
                                        -- Submit -> QMGR
newmessage OPERATION
        ARGUMENT MsgStruct
        RESULT NULL
        ERRORS { congested, protocol }
        ::= 0
                                                                        70
                                        -- read a message
                                        -- Console -> QMGR
                                        -- QMGR -> QMGR
readmsginfo OPERATION
        ARGUMENT ReadMessageArgument
        RESULT MsgList
        ERRORS { congested, protocol }
        LINKED {msginfo}
        ::= 1
                                                                        80
                                        -- QMGR -> Console
                                        -- Incremental update
msginfo OPERATION
        ARGUMENT MsgList
        ::= 10
```

Appendix F

Qmgr Protocol

This appenix contains the ASN.1 description of the qmgr protocol.

```
-- qmgr.ry - Qmgr operation definitions

Qmgr
DEFINITIONS ::=
BEGIN

IMPORTS MPDUIdentifier FROM QmgrP1;

-- EXPORTS everything
                                                                    10

-- PORTS
qmgr-use OBJECT
      PORTS { newmessage[S],
                  readmsginfo[S],
                  msginfo[C],
                  channelinitialise[C],
                  processmessage,
                  channelbegin[S],
                  readqueue[C],                                     20
            channelread[S],
                  channelinfo[C],
                  channelcontrol[S],
                  mtaread[S],
                  mtainfo[C],
                  mtacontrol[S],
                  readchannelmtamessage[S] } ::= id-pt-qmgr-use


activeChannel OBJECT                                                30
            PORTS { channelinitialise,
                        processmessage [S] }
          ::= id-pt-activeChannel
```

After processing all inputs, a sequence of procedure calls to are made to *submit* the message (as outlined in Volume 2). Thus (summarised) :

```
io_init(rp);      /* initialisation */
io_wprm(&prm,rp); /* write management parameters */
io_wrq(&q,rp);    /* per message parameters */

ap = ad_originator;            /* originator */
io_wadr(ap,AD_ORIGINATOR,rp);
ap = ad_recipient;
do {                           /* list of recipients */
   io_wadr(ap,AD_RECIPIENT,rp);
   ap = ap->ad_next;                                          10
   } while ( ap != NULL );
io_adend(rp);                  /* terminate address list */

if((p=next_arg("body",argv)) != NULL) {
   io_tinit(rp);               /* prepare for body part(s) */
   do {
      io_tpart(p,FALSE,rp);    /* prepare one body part */
      loop_data(p);            /* transmit with io_tdata() /*
       io_tdend(rp);           /* end one file */
   } while(p=next_arg("body",argv));                          20
}
io_tend(rp));   /* complete message */
io_end(type);   /* terminate session with submit */
```

Figure E.1: io_ Interface

Each procedure call returns a status in the error structure `RP_Buf` which is displayed as follows:

```
RP_ERROR error_number error_string.
```

Such errors abort the submission.

Flag	Purpose
default	override mu.defaults
file	input from file(s)
value	command line specification of value
body	body part file name(s)
tree	directory name(s)
confirm	wait for y/n after `$` substitution
nofix	ignore errors resulting from user input
logfile	re-route output messages to file
help	list flag table analysis
–	same as –help
!	dummy next flag to terminate sequence (future use)
0	debug (see above)
1	debug
2	debug
3	debug
4	debug
5	debug

Table E.1: Summary of MU flags

```
Start-of-MsgEnvAddr
*Recip type=822 orig=$@uk.ac.ucl.$ End-of-addr
```

will prompt for `Content-type`, `Priority`, `MsgId string`, and two components from `Recip` lines until a null input. Each input line must be confirmed with '`y`', but any errors will be disregarded. Section delimiters such as `Start-of-MsgEnvAddr` are redundant, since *mu* accepts control lines in any order, but are benign.

E.3 mu internals

mu uses the argument handling routine `argh` and can therefore accept any unambiguous prefix for a flag specifier. (See separate document.)

mu has no knowledge of the significance of any input line other than that it is valid for *txt2prm*, *txt2q* or *txt2adr*. The one exception to this is that, to allow a recipient to be specified before the originator, each potential address line is tested against `"Origs"`. (Because the originator is held in a separate structure from the recipient list, and *io_wadr* needs to have `AD_ORIGINATOR` or `AD_RECIPIENT` passed to it explicitly.) Second and subsequent `"Origs"` lines are treated as errors.

E.2 User input

The reserved character `'$'` (TOKENCHAR) is used to indicate that the user should input a value to replace the `'$'`, unless the line is a comment line or the `'$'` is immediately followed by a second to escape a real `$` symbol. The portion of the input line up to and including the `'$'` is displayed on screen, then a carriage return delimited string is input to replace the `'$'`. The substituted line is scanned for further unpaired `'$'` characters. When all have been replaced, if the flag `-confirm` has been specified, the user input will be re-displayed awaiting user confirmation or cancellation with `'y'` or `'n'`. The line is then offered to the *submit* input routines. Any error will cause the input portions to be re-prompted *unless* the flag `-nofix` has been specified, in which case *mu* will ignore the error and continue.

To allow repeated user input, for example, of multiple recipient lines, another reserved character `'*'` (REPEATCHAR) may be placed at the start of a line to indicate repetition. If this character appears at the start of a line which contains no unpaired `'$'` characters, an error

```
error : no TOKENCHAR(s) in REPEAT line - REPEATCHAR ignored
```

is reported. Otherwise, user input takes place as before, terminated by a null input (just a carriage return) on any substitution field in the line. Repeat lines are stripped of the `'*'` character before passing to the *submit* input routines.

Example — command line :

```
mu  -confirm -file a1 -nofix
```

where file `a1` contains :

```
# Start-of-MsgEnvPrm
-body hdr.822
-body 2.ia5
# $$ implies $
-file level$$
Message-type User-Mpdu
# $ symbols in comments are ignored
Inbound-channel 822-local
Inbound-host UK.AC.UCL.CS
Content-type $
Encoded-info EncTypes=iA5Text
Priority $
PerMsg-Flags disclose-recipients
PerMsg-Flags return-contents
MsgId string $
```

1. To open the file and check its length. Two fatal errors are detected :
   a. `unable to read body part %s`
   b. `body part %s is null file`
2. To open the file in order to pass its contents to *submit* with a series of *io_tdata(buf,length)* calls. Again, failure to open is reported :
   `unable to open body part for reading %s`
3. To pass a name for *submit* to use in the queue. *submit* generates a message directory to match the supplied pathname. Thus:
   `-body 4.ipm/hdr.822uk` — generates :
   usr/spool/pp/queues/msg/msg.annnnnn/base/4.ipm/hdr.822uk

If a reserved character '*' (FILEMARKCHAR) is embedded in the `-body` pathname, *mu* strips all characters up to and including the '*' from the pathname before passing to *submit*. This allows *mu* to distinguish between the path used to find the file(s) and that required within the message. Thus:

```
-body nest1/nest2/*3.ipm/1.ia5
```

generates:

```
/usr/spool/pp/queues/msg/msg.annnnnn/base/3.ipm/i.ia5
```

The '*' is removed for purposes (1.) and (2.) above. The character '*' may also be placed in the pathname to a directory to truncate the pathname passed on to *submit* from a `-tree` command — in the same way as for `-body` specification.

Because the line parsing routines used in *mu* are exactly those in *submit* itself, any values/options which are acceptable to *submit* will also be accepted by *mu* (but in any sequence).

To illustrate :

```
mu  -- -0 -file test1 -body b1 /tmp/jtaylor/messdir/*b2 \
    -default new.defaults -value MsgiD Country=US
```

will read from `"new.defaults"`, (processing any `'-flag'` specifiers it contains), then from file `"test1"` finally setting a new country `US` to override any file-specified value. The `-0` flag turns on progress messages and `--` will list the flag table. During submission, files `"b1"` and `"b2"` will be transmitted individually.

fault file is parsed first (*mu.defaults* unless overridden with a `-default` flag on the command line), then any `-file` specified files from the command line, finally `-value` specifications from the command line. Body part files may be specified with `-body` commands, either on the command line or embedded as with other commands.

mu uses PP's standard logging —- additional logging messages generated by –0 –1 –2 –3 flags are output to a channel (`muout`) which defaults to `stdout` unless flag `-logfile` is specified. Note that output may be re-routed in the course of processing input files. The flag –5 allows these messages to be appended to the nominated logfile, rather than replacing it.

In addition to the LLOG_TRACE and LLOG_DEBUG calls associated with flags `-0` and `-1` shown below, there are some other LLOG_FATAL and LLOG_EXCEPTIONS calls. Note that functions `list_args` and `show_args` (in *argh.c*) do not generate LLOG events. *mu* accepts a number of debug flags `-0`, `-1` etc. — currently these are used as follows :

flag	function
-0	log to *muout* progress messages, and PPLOG(LLOG_TRACE ...)
-1	log to *muout* command files as they are processed, and PPLOG(LLOG_DEBUG ...)
-2	list to *muout* body part files as they are io_tdata'ed
-3	list to *muout* structures prior to submit io_ calls
-4	abort prior to submit io_ calls
-5	append to logfile rather than overwrite

mu accepts `-body` part file names with any arbitrarily long pathname. The full path is used :

Appendix E

Testing PP - the MTS User Interface

A special user interface known as *mu* is provided as a general tool to help test *submit* and consequent operations in PP. Setting defaults in the *mu.defaults* file allows a minimum number of message-specific parameters to be provided. It may be used to re-submit a modified text-encoded file taken from the queue.

E.1 Overview

The message submission tool *mu* takes input from the command line, a defaults file *mu.defaults* if present in the working directory, otherwise from the default command directory `cmddfldir` and then any other files specified with `-file` flags. A full summary of the flags to *mu* is given in Table E.1 on page 118. Any command line option may be placed on a line within an input file. *mu* calls its command line parser recursively when it encounters a line starting with `'-'` (FLAGCHAR). Any line starting with `'#'` (COMMENTCHAR) is ignored. Otherwise, all lines are offered to the various *submit* input routines in turn for incorporation into PP's internal message structures (*prm_vars*, *Qstruct* and *ADDR*). Any line which returns an error code (`rp_isbad`) from all these routines is assumed to be in error.

The input lines which specify assignments as specified in QUEUE(5) may be received in any sequence, and inputs may override earlier entries. Any value may also be assigned on the (original) command line with `-value v1 v2 ...` up to the next flag.

If input files are nested, input lines from the nested file are processed before the remainder of the file containing the `-file` specifier. The de-

```
EOF
%
```

- JANET DTE 00000511160013 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 obtianed 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
```

Appendix D

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 rfc1148 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

```
        [ -d $STATDIR ] || mkdir $STATDIR                                30
        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)                                                40

# update the niftp stuff
echo 'building the niftp database'
(cd niftp; make )

echo UPDATE complete
```

Figure C.1: An Example Nightly Script

Appendix C

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                                                          10
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                                                            20
        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`
```

Appendix B

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                                                    10
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                                                                   20
echo '.'
EOF
```

Figure B.1: An Example Start Script

```
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                          300

smtp                    normlog     file=smtp
smtpsrvr                normlog     file=smtp
greyout                 normlog     file=greybook
gb-janet                normlog   file=greybook
submit                  normlog     file=submit

local                   normlog     file=local
msg-clean               normlog     file=msg-clean
                                                                       310
x400in84                normlog     file=x400in84
                        level=trace,level=pdus
x400out84               normlog     file=x400out84
                        level=trace

x400in88                normlog     file=x400in88
                        level=all,dlevel=debug

x400out88               normlog     file=x400out88
                        level=all,dlevel=debug                         320
xx400out88              normlog     file=x400out88
                        level=all,dlevel=debug

#isodelog               rtsaplevel file=rts,level=all
pptsapd                 normlog     file=pptsapd
                        dlevel=notice
xpptsapd-stat           normlog     file=pptsapd

x400in88 isodelog psap2level level=all file=x400in88
x400in88 isodelog acsaplevel level=all file=x400in88                   330
x400in88 isodelog rtsaplevel level=all file=x400in88
```

Figure A.1: An Example Tailor File

```
        bptout=hdr.822-us,chanout=dr2rfc,
        info="rfc822norm -822 -littleend"
chan    822to987
        prog=fcontrol,type=shaper,bptin=hdr.822,
        bptout=hdr.822-987,chanout=dr2rfc,
        info="rfc822norm -percent -733 -stripdomain cs.ucl.ac.uk"
chan    822top2
        prog=RFCtoP2,type=shaper,bptin=hdr.822-987,conv=1148
        bptout=hdr.p2,chanout=dr2rfc,cost=10,
        show="RFC987 RFC 822 -> P2"
chan    822top22
        prog=RFCtoP2,type=shaper,bptin=hdr.822-987,conv=1148
        bptout=hdr.p22,chanout=dr2rfc,cost=10,
        show="RFC1138 RFC 822 -> P22"
chan    p2to822
        prog=P2toRFC,type=shaper,bptout=hdr.822,conv=1148,
        bptin=hdr.p2,chanout=dr2rfc,cost=10,
        show="RFC987 P2 -> RFC 822"
chan    p22to822
        prog=P2toRFC,type=shaper,bptout=hdr.822,conv=1148
        bptin=hdr.p22,chanout=dr2rfc,cost=10,
        show="RFC1138 P22 -> RFC 822"
chan    p22top2
        prog=foo,type=shaper,bptout=hdr.p2,bptin=hdr.p22,
        chanout=dr2rfc,show="P22 to P2 downgrade",
        conv=conv,cost=10
chan    p2top22
        prog=foo,type=shaper,bptout=hdr.p22,bptin=hdr.p2,
        chanout=dr2rfc,show="P2 to P22 upgrade",cost=1

# Body part filters

chan    g3faxtotxt
        prog=fcontrol,type=shaper,bptin=g3fax,
        bptout=ia5,info="removebp G3Fax",chanout=dr2rfc,
        show="Fax -> Text mapper",cost=100,conv=loss

#  **-- Logging for the rest of the prog --**
#
authlog level=notice, size=400
operlog level=notice, size=400
normlog level=notice, size=400, sflags=zero

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
```

```
chan        qmgr-load
            prog=qmgr-load,show="Loading the QMGR",
            type=qmgrload                                          190

chan        msg-clean
            prog=msg-clean,show="Removing finished message"
            type=delete

chan        trash
            prog=trash,show="Removing trash"
            type=debris

chan        timeout                                                200
            prog=timeout,show="Timeout messages"
            type=timeout

#  **-- shaper channels --**

#   Flatteners

chan        p2flatten
            prog=p2flatten,show="Structured message -> P2",
            type=shaper,content-out=p2,chanout=dr2rfc               210

chan        822flatten
            prog=rfc934,type=shaper,content-out=822,chanout=dr2rfc

chan        p22flatten
            prog=p2flatten,show="Structured Message -> P22",
            type=shaper,content-out=p22,chanout=dr2rfc

#   Unflatteners
                                                                   220
chan        p2explode
            prog=p2explode,type=shaper,content-in=p2,chanout=dr2rfc

chan        p22explode
            prog=p2explode,type=shaper,content-in=p22,
            chanout=dr2rfc

# Header Filters
chan        822touk
            prog=fcontrol,type=shaper,                             230
            bptin=hdr.822,
            bptout=hdr.822-uk,chanout=dr2rfc,
            info="rfc822norm -jnt -bigend"
chan        822tojnt
            prog=fcontrol,type=shaper,
            bptin=hdr.822,bptout=hdr.822-jnt,chanout=dr2rfc,
            info="rfc822norm -jnt -bigend -jntsender $(822SENDER)"
chan        822tous
            prog=fcontrol,type=shaper,bptin=hdr.822,
```

```
chan    gb-janet
        prog="greyout -P",show="via JANET with NIFTP (PP)",
        type=out, adr=822,adr-orderf=ukpref,
        bptout="hdr.822-jnt,ia5",info=janet,
        content-out=822,chanout=dr2rfc                                  140

#chan   gb-janet
#       prog=smtp,show="via JANET with NIFTP (PP)",type=out,
#       adr=822,adr-orderf=usapref,
#       bptout="hdr.822-us,ia5",
#       content-out=822,mta=much.cs.nott.ac.uk,chanout=dr2rfc

chan    gb-pss
        prog=smtp,show="via PSS with NIFTP (PP)",type=out,
        adr=822,adr-order=ukpref,                                       150
        bptout="ia5,hdr.822-jnt",
        content-out=822,mta=much.cs.nott.ac.uk,chanout=dr2rfc

chan    ipss
        prog=smtp,show="via PSS with NIFTP (PP)",type=out,
        adr=822,adr-order=ukpref,
        bptout="ia5,hdr.822-uk",
        content-out=822,mta=much.cs.nott.ac.uk,chanout=dr2rfc

chan    list                                                            160
        prog=list,show="List channel",type=both,
        table=list,chanout=dr2rfc

chan    uucp-out
        prog=uucp-out,show="UUCP outbound channel",type=out,
        adr=822,adr-order=ukpref,table=uucp
        chanout=dr2rfc,
        info="uux=/tmp/uux,host=nott-cs"

chan    uucp-in                                                         170
        show="UUCP inbound channel",type=in,
        adr=822,chanout=dr2rfc

chan    dr2rfc
        prog=dr2rfc,show="Dr2rfc channel",type=out,
        adr=822,adr-order=ukpref,
        bptin="hdr.822, ia5",
        content-out=822,chanout=dr2rfc

chan    shell                                                           180
        prog=shell,type=out,access=mts
        show="Shell channel",
        table=shell,chanout=dr2rfc,
        bptout="ia5,hdr.822-uk", content-out=822

# QMGR special channels (load/manage/clean)
```

```
tbl         list                    file="ch.list",
                                    show="Lists:  distribution lists",
tbl         uucp                    file="ch.uucp"
                                    show="UUCP: rfc822 -> uucp",

# Note that SMTP channels use BIND or /etc/hosts
# And greybook channels use NRS                                          90

#  **-- "Real" Channels --**
#

chan        822-local                   prog=local,
            show="Local Delivery channel",type=both,
            sort="user time",adr=822,adr-order=ukpref,
            bptin="ia5,hdr.822",bptout="ia5,hdr.822-uk",
            content-out=822,table=local,access=mts,chanout=dr2rfc
                                                                          100
chan        slocal                  prog=slocal,
            show="Structured Local Delivery channel",type=both,
            sort="user time",adr=822,adr-order=ukpref,
            bptin="hdr.822,dmd,g3fax,ia5",bptout="hdr.822,g3fax,odif,ia5",
            table=local,access=mts,chanout=dr2rfc


chan        x400in84
            prog=x400in84,show="X400 inbound",type=in,
            content-in=p2,adr=x400,                                       110
            adr-order=ukpref,table=x400in84,chanout=x400out84

chan        x400in88
            prog=x400in88,show="X400 inbound (1988)",type=in,
            content-in=p22,adr=x400,
            adr-order=ukpref,table=x400in88,chanout=x400out88

chan        x400out84
            prog=x400out84,show="X400 outbound",type=out,
            adr=x400,adr-order=ukpref,                                    120
            content-out=p2,bptout="ia5,hdr.p2,g3fax,odif",
            table=x400out84,chanout=x400out84

chan        x400out88
            prog=x400out88,show="X400 outbound",type=out,
            adr=x400,adr-order=ukpref,
            content-out=p22,bptout="ia5,hdr.p22,g3fax,odif",
            table=x400out88,chanout=x400out88

chan        smtp                                                          130
            prog=smtp,show="with SMTP (PP)",type=out,
            adr=822,adr-order=usapref,
            bptout="ia5,hdr.822-us",content-out=822,
            chanout=dr2rfc
```

```
authchannel         free
authloglevel        high
wrndfldir           warnings

#  **-- Directory variables --**
#
cmddir              /crg/pp/cmds
quedir              /crg/pp/spool
logdir              /crg/pp/logs                                    40
tbldir              /crg/pp/tables

# Body types

bodypart            hdr.822 hdr.822-jnt hdr.822-us hdr.p2 hdr.822-uk
bodypart            ia5 dmd odif g3fax ipm hdr.p22 hdr.ipn hdr.822-987


#  **-- Mapping tables --**
#                                                                   50
# All of these table MUST have the names given here
tbl         aliases             show="Aliases:  mapping -> local id"
tbl         users               show="Users:  local users"
tbl         domain              show="Domain table"
tbl         or                  show="Mapping O/R Address -> MTA"
tbl         channel             show="Binding MTA -> Channels"
tbl         or2rfc              show="RFC 987:  X.400 -> RFC 822"
tbl         rfc2or              show="RFC 987:  RFC 822 -> X.400"
tbl         rfc1148gate         show="RFC 1148 gateways",
                                flags=linear                        60
tbl         auth.channel        show="Authorisation:  channel policy"
tbl         auth.mta            show="Authorisation:  mta based"
tbl         auth.user           show="Authorisation:  user based"
tbl         auth.qmgr           show="Authorisation:  qmgr control"
#  **-- Channel tables --**
#
# Needed by individual channels, not submit
tbl         local               file="ch.local",
                                show="local id -> user id"
                                                                    70
tbl         x400in84            file="ch.x400in84",
                                show="MTA -> connect info",
                                flags=linear
tbl         x400out84           file="ch.x400out84",
                                show="X.400(84) connect 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",      80
                                flags=linear
tbl         shell               file="ch.shell",
                                show="info for shell chan"
```

Appendix A

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
#
##############################
#
# @(#)$Header$
#
# $Log$
#                                                                    10
##############################
#

#  **-- Logging of this tailor file when it is being read --**
#
authlog level=fatal, size=400
operlog level=fatal, size=400
normlog level=fatal, size=400

#  **-- Special strings --**                                         20
#
loc_dom_mta         trellis.cs.nott.ac.uk
loc_dom_site        cs.nott.ac.uk
loc_or              "/OU=CS/O=NOTT/PRMD=UK.AC/ADMD=GOLD 400/C=GB/"
postmaster          postmaster@cs.nott.ac.uk
pplogin             pp
pptsapd_addr        LOCAL-ETHER=trellis+20001
qmgrhost            trellis
delim1              "\1\1\1\1\12"
delim2              "\1\1\1\1\12"                                  30
```

An example text encoded delivery report is shown in Figure 6.3.

```
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 6.3: Example delivery report format

6.9.2 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 987.

6.9.3 Warnings

There are hooks to support warnings from the MTA to the message originator, typiocally of slow progress of a message through the MTS. This will be added in due course, by use of a new (private) protocol (to be specified). It is hoped to also provide a mechanism to query remote MTAs.

```
Start-of-MsgEnvPrm
Message-type User-Mpdu
Message-size 269
Return-interval 168
Content-type p2
Encoded-info EncTypes="hdr.p2,ia5"
Priority non-urgent
cont-return true
Inbound-channel list
Inbound-mta "cs.nott.ac.uk"
MsgId string="trellis.cs.981:30.07.90.12.10.44" Country=GB
Admin="GOLD 400" Prmd="UK.AC"
Trace mta="trellis.cs.nott.ac.uk" Country=GB Admin="GOLD 400"
Prmd="UK.AC" arrival="Thu, 30 Aug 1990 12:10:44 +0000" action=0
End-of-block
dl-exp-history "/S=private/OU=CS/O=NOTT/PRMD=UK.AC/ADMD=GOLD 400/C=GB/"
"" "Thu, 30 Aug 1990 12:10:44 +0000"
Queued-time "Thu, 30 Aug 1990 12:10:44 +0000"
Start-of-MsgEnvAddr
Origs rno=000 status=done reform-done=000 outchan=dr2rfc
outmta="cs.nott.ac.uk" ext-id=0 resp=false mta-rreq=basic
usr-rreq=basic type=822 orig=private-request
x400="/S=private-request/OU=CS/O=NOTT/PRMD=UK.AC/ADMD=GOLD 400/C=GB/"
rfc="private-request@cs.nott.ac.uk" End-of-addr
Recip rno=001 status=pend reform-done=002
reform-list="p2explode,p2to822,822touk,822flatten"
outchan=822-local outmta="trellis.cs.nott.ac.uk" ext-id=1
resp=true mta-rreq=basic usr-rreq=basic type=822
eits="ia5,hdr.822-uk" content=822 orig=pac
x400="/I=p/S=cowen/OU=CS/O=NOTT/PRMD=UK.AC/ADMD=GOLD 400/C=GB/"
rfc="p.cowen@cs.nott.ac.uk" End-of-addr
```

Figure 6.2: Example Address file

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.

6.9 Examining the PP Queue

Occasionally there may be problems encountered by PP that can not 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 composed of two directories *addr* and *msg*. An item in the queue is identified by a file in the *addr* directory with a unique name. This file is text encoded, and has information which corresponds approximately to P1 functionality. In the *msg* directory, there is a directory of the same name as the queue file, and contains the message. There is a subdirectory *base*, which contains the message as it arrived, and other directories, prefixed with the string *base.*, which contain transformed versions of the message content. There are also files *report.**, containing text encoded delivery reports associated with the message.

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 6.2.

6.9.1 Delivery Reports

The basic operation of a channel is discussed in Section 3.8 on page 38. 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 bar the lowest are marked to done in the queue, and the first is marked to dliv.

1. You are using the SMTP protocol.
2. You are not running SMTP from *inetd* (see Section 3.7.1 on page 30 for details of this).

- (optionally) *tsapd* 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.

6.8.2 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 C for an example)

The sequence of operations should be :

- Archive the day's logs and remove outdated logs (see section 6.8.3 on page 100)
- Collect statistics
- Import new table source data files if necessary
- Build new tables
- *dbmbuild*

6.8.3 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.

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

```
bodypart    hdr.wb
```

- The `ia5` body part is ok 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 as 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

```
chan wbhdr-2-822 prog=wbto822,type=shaper,bptin=hdr.wb,
                 bptout=hdr.822,
                 show="WhizBang to 822 Header converter"
```

6.8 The day–to–day running of PP

6.8.1 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 B) 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,
- (optionally) *smtpsrvr* - the SMTP listener which runs as the superuser. This is only necessary if

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 unecessary. 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, info="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.

6.7.2 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.

```
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)
```

6.6 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 don't 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.

6.7 Adding Filters, Shapers + EITs

6.7.1 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.

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 re-format 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. With 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

```
cs.nott.ac.uk:cs.nott.ac.uk
cs.ucl.ac.uk:cs.ucl.ac.uk
stc.co.uk:stc.co.uk
widget.uucp:widget.uucp
```

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:cs.nott.ac.uk
cs.nott.ac:cs.nott.ac.uk
cs.nott:cs.nott.ac.uk
cs:cs.nott.ac.uk
cs.ucl.ac.uk:cs.ucl.ac.uk
cs.ucl.ac:cs.ucl.ac.uk
cs.ucl:cs.ucl.ac.uk
stc.co.uk:stc.co.uk
stc.co:stc.co.uk
widget.uucp:widget.uucp
widget: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.channel* 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)
```

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 has only fully qualified domains on the LHS. This is compiled from one or more sources:

Local Name Your own domain name, and alternates are needed. For example:

```
cambridge.ac.uk:cambridge.ac.uk
cam.ac.uk: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 *pp.domain* can be used as direct input.

Internet A list of top level domains which you wish to be passed to the SMTP/DNS channels should be given. For example:

```
edu:edu *
gov:gov *
com:com *
mil:mil *
```

Other known domains Other known domains (e.g., those associated with local UUCP links) should be added in. For example:

```
iconet.uucp:iconet.uucp
inset.uucp:inset.uucp
```

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 filled in. An example input might be the following:

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

RFC 987 mapping tables

RFC 987 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 987 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
   - If there is a second component on the RHS of the domain table, then this indicates the number of levels of subdomain permitted ($* \equiv \infty$).

2. If LHS and RHS of domain table are not equal, replace domain with RHS and go back to 1. This substitution allows for both alternate names and for abbreviations (omitting the most significant domain components).

3. Look up RHS of 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.

- 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, don't forget to run *dbmbuild* or else change the relevant table to linear mode.

- Less likely, but worth checking is that submit is installed properly with the correct setuid bit on (it should be mode 4755 or similar).

On outgoing channels, it is more usually 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 the following:

- 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 progrma is much like running the unix *tail*(1) command with the `-f` flag on several files at once.

6.5 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.

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`: Interpret 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.

6.4 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.

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. 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

6.3.3 Checking of Addresses

It is useful to check out addresses sometimes 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, 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:

`-n`: Check the address without expanding aliases.

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

6.3.1 ckchan

The *ckchan* tool prints out information on the state of currently active channels and is invoked as follows:

```
ckchan [-v] <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.

6.3.2 ckconfig

The *ckconfig* tool performs various checks on the configuration of PP including:

- 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 (*tbldfldir*).

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 it's checks and report back the results. No automatic correction will take place and the user won't be prompted for corrections.

It is worth running this program at intervals and after every change you make to PP.

`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 6.1 shows an extract from the MTA consoles hardwired defaults.

```
#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 6.1: Extract From the Console's Tailoring

6.3 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 MTA console should be the main source of information for the system administrator.

Tailor File Values		Key
type	access	
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 6.2: Tailor File Entries to Console Keys.

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 6.2 gives all the types of channel that can be used in keys and how they are constructed. Some valid keys of this form are "mtaboth-chan", "passive-msg", "internal-msg".

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)

`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 either be relative to the current directory, a full path name or relative to 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 by 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".

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 6.1: MTA Console Accelerators

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 6.1 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.

6.2.4 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 through out 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.

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 `x400out84`, then the console would make `x400out84` 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

Channel buttons

Button	Effect
`info`	display information about the channel
`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, my clearing its delays and those or messages and mtas waiting on this channel

MTA buttons

Button	Effect
`info`	display information on this mta
`disbale`	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
`show all`	by default only information on ten messages is shown, this button will show all messages on this mta/channel.
`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 an deliver the message now by clearing all delays associated with this message.

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.

6.2.3 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 shows the effect of the command buttons that can be found on the *MTAconsole*.

Top row of buttons

Button	Effect
`Refresh/Reconnect`	Refresh the data or reconnect to the qmgr.
`quit`	exit the console program
`connect/disconnect`	connect or disconnect ot the qmgr.
`control/monitor`	change between the two modes
`configuration`	set various parameters
`Qmgr control`	perform global qmgr control

Heuristic Configuration

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

`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 endevour 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

`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 6.2.4. 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. `-colours 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`).

number of messages on that entity or the age of the oldest message on that entity. Section 6.2.4 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 three possible heuristics. These are described in section 6.2.2.

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.

6.2.2 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:

`command line arguments`

`user defaults`

- 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, timeout etc are special to the qmgr and will only ever be invoked one at a time.

The maximum number of channels defaults to 3, 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 console 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 is run for a while before adjustment is made.

6.2 MTA Console

The MTA console is a management tool that can be used to monitor and/or control a *qmgr*. This tool uses the Athena widget set on top of X11 release 3 or 4.

Note there is a known bug in the MTAconsole which causes it to crash occasionally. See the README file in the source directory for more details.

6.2.1 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

started in this mode, channels can be gradually enabled via the MTA console.

`-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 6 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.

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 channels at once is. This will be overridden by the per channel maximums, so they 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 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 used, starting multiple instances of such a channel is pointless.

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

6.1.3 Active Channel

An active channel is invoked by the QMGR, which starts a ROS association. The channel is fully synchronous w.r.t. ROS. 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.

6.1.4 Passive Channel

Passive channels have not yet been implemented.

6.1.5 Cleanups, timeouts 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.

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

6.1.6 Qmgr management and tailoring

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

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

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

`-D:` This switch starts up the qmgr with all channels disabled. This is most useful to allow controlled start up of channels. Once

Chapter 6

Managing PP

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

6.1 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 help by the queue manager. This is held in a sparse matrix, initially sorted by channel, host/user, and priority. The queue manager is not, inherently, a trusted process, as it does not access the queue directly. The protocol is listed in Appendix F in Figure F.1.

6.1.1 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.

6.1.2 Queue Reading

A ROS operation allows the queue to be read. This is used at startup and for occasional refresh.

- If rtsping84 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

5.2.10 A Common X.400 incoming connection problem.

Sometimes a Remote Site attempts to do a RTS recovery, due to some transmission failure. PP at the moment, does not support recovery, and so a "cannotRecover" reason is given. This sometimes does not terminate the connection due to a bug in the Remote Site's software, and so it keeps on trying, which leads to much network activity.

The X.400 Incoming logs will look something like:

```
2/ 8  8:30:31 x400in   01392 (#52     )
       Starting x400in (via X400(84) inbound)
2/ 8  8:30:33 x400in   01392 (#52     )
       RtBInit, [Congestion at RtSAP]
       rejecting attempted recovery
```

To get the Network address of the Remote Site attempting the recovery, look at the ISODE tsapd.log, and then match the Network address against the X.400 Inbound Table keys.

- 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 `info=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.

5.2.9 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 program rtsping84, located in the PP *TOOLDIR*, to make a RTS connection to the Remote Site. For information on how to use rtsping84 see its manual page. In the meantime try running:

  rtsping84 -c*x400out-channel-name RemoteSiteName*

  for example

  ```
  rtsping84 -cx400out gb-ucl
  ```

- 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

- 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 page 70) *but* there will need to be a number of inbound `rpsap` entries for each Network Address belonging to that Remote Site. Though all other details (`rname`, `rmta`, `rpass`) would be the same.

5.2.8 How to Locate a Remote Site making a test connection to you.

- 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 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 MTAName - "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:

```
#-***********************
#
#
#       -*** 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 X.400 Inbound Table

Please note when taking these decisions:

- If *info=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 *info=sloppy* should be set for testing purposes only.

- That for each Remote Site an X.400 inbound and outbound entry should exist. Otherwise if *info=sloppy* is set, and a Remote Site's Network Address keys successfully into the X.400 Inbound Table, if there is no 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 page 74) are very similar to the ones in the X.400 Outbound Table (see page 70), and should be set accordingly, except for 2 main differences:

- *rpsap* - which acts as the table entry key.

- *rname* - an extra parameter which specifies the Remote Site Name. 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 on page 64 and 64.

In the Figure 5.5 above, it can be seen that:

- A Remote Site 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 Site, giving "Validation failure" as the reason.
- If the key is valid, then the Remote Site's MTA Name and Password are checked against its *rmta* and *rpass* values specified within the X.400 Inbound Table.
- If the values match 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 Site, giving "Validation failure" as the reason.

NOTE: 2 additional fields may be set to facilitate X.400 Inbound Connection testing.

- *info=sloppy* 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.

5.2.7 Edit 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 *info=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 *info=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*.

5.2.4 Channel Table Usage

The section describes briefly the actions that are taken to validate an X.400 address.

1. Parse an address in X.400 style using the *rfc2or* and *or2rfc* tables if necessary to convert from RFC-822 format.

2. Use the *or* table to try and locate a Remote Site name, through which the address may be relayed

3. If a Remote Site Name is associated with the address, then look at the Channel table to locate the cheapest route by which that Remote Site may be reached.

5.2.5 Edit the Channel Table

The Channel Table below gives the NASA and UCL entries using the Remote Site Information specified on page 64 and 64.

- Note that the key to this table has the same value as that in the X.400 Outbound Table on page 70.

- Also, note that the name specified in the rounded brackets is the name of the X.400 Outbound Channel, and possesses the same value as that specified in the *tailor* file.

```
#-***************************
#
#       -*** nasa ***-
us-nasa:us-nasa(x400out)
#
#       -*** ucl ***-
gb-ucl:gb-ucl(x400out)
#
#-***************************
```

Figure 5.4: Example Channel Table

5.2.6 X.400 Inbound Procedures

A Brief Description of how the X.400 Inbound Table is used within PP.

```
#-************************************
#
#
# -*** 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 X.400 outbound table

- Remote Site Name - `key`.
- MTA Name - `rmta`.
- Password - `rpass`.
- Network Address, TSAP and pid - `rpsap`.

NOTE

- All keys in the X.400 Outbound Table must be unique.
- One or more of the "default" fields (`lmta`, `lpass`, `lpsap`) are only set within a Remote Site entry when they are to possess values different to those of the `default`. If these values are set within a Remote Site entry then, when a connection is requested to that particular Remote Site, the local PP will identify itself differently to normal. This is only set in very rare cases, for example, if a Remote Site cannot cope with a special character in the `lpass` field.
- 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 on page 70. The formats depend on whether the Network Address is an X.25 DTE or an IP one, and if an X.25 DTE, on whether the pid field is specified.
- The `trynext` field is optional, and may be used when there is an alternative Remote Site willing to accept mail on behalf of another. If this field is used, 2 or more Remote Sites will be tried in succession. See section 4.11 for more details. See also page 70, for an example of how this field may be used.
- The `tracing` field is optional, and is only used if trace compression is required. See section 4.11 again for more details.

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.

- If the Remote Site's MTA Name and Password matches up, PP starts to transmit a message, otherwise PP terminates the X.400 connection.

5.2.3 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 on page 70 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 page 63).

- MTA Name and Password (both optional).
- Network Address (mandatory)
- Transport Selector (optional)
- Protocol ID or pid (optional).

If a Remote Site has more than one Network Address then the PP administrator will need to decide:

- Which Network Address should be always tried first.
- Whether all the Network Addresses should be entered (maximum 4).

If all Network Addresses are entered, then the one that is always used, is specified first in the `rpsap` field whose formats are described on page 70.

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 on page 70 when doing this.

```
#   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

```
# 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

5.2.2 Editing the O/R Table

Replace the `_DEFAULT_MTA_` occurrences with appropriate remote site names. These are the names by which the sites are internally identifiable within PP. The choice of name 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 on pages 66 and 67 please note that:

- The O/R skeleton table in figure 5.1 on page 66 is a truncated example of the *retrieved* UCL O/R Master Table.
- The O/R skeleton table in figure 5.2 on page 67, is a truncated example of an *edited* UCL O/R Master Table.

X.400 Outbound Procedures

A Brief Description of how the X.400 Outbound Table is used within PP.

- PP connects to a Remote Site using its specified Network Address (in the `rpsap` field). If it cannot connect, because the Remote Site's X.25 or IP listener is not running for some reason, and if more than one address is specified, then all addresses are tried in the order specified by the `ts_communities` tailor variable in *isotailor*, until either failure occurs or a connection is made. If a specific order is required then use may be made of the `trynext` field.
- When the connection is requested, PP gives its local MTA Name and Password as identification.
- The Remote Site *either* accepts connection and returns its own MTA Name and Password *or* it rejects the connection and gives as its reason "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).

`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 X.400 information for UCL

- 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.

5.2 Setting up the Routing Tables

The following steps need to be taken to set up the X.400 routing tables.

5.2.1 Routing and mapping tables

First, a number of mapping tables need to be installed. If you don't have a copy of them, you will need to get one. Copies should be obtained from University College London (see Appendix D for details).

`or:` The O/R Table (See Figure 5.1 below and section 4.6)

`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 rfc1148 known gateways file.

The *rfc2or* and *or2rfc* are the RFC 987/RFC 1148 tables. Incorporate these RFC 987 tables in the Table directory.

	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 pre-fixed 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 X.400 information for NASA

5.1.1 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(1984)). The value "591" is often used.

Chapter 5

Setting up X.400 Links

This chapter shows how to go about setting up X.400 interconnection with other sites.

Information on X.400 routing and connection establishment is entirely table driven. There is no standardised support by the Message Directory Service. This section describes how to setup the *or* routing tables, and the 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.

5.1 Remote Site information

Obtain the information below prior to setting up an X.400 link.

`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 paramters

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.

By arranging permissions differently loose to highly restricted access is possible.

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

Duplicate recipients are eliminated (second and subsequent) if there is an exact match on the 822 or X400 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 non-delivery reports.

4.18 Qmgr Authentication

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=" <encrytped-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 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.

Failure Reasons
unknown error
access policy: none
negative: in-MTA (%s %s) sender (%s %s) out-MTA (%s %s) recip ((%s %s)
block: in-MTA (%s %s) sender (%s %s) out-MTA (%s %s) recip (%s (%s)
inbound MTA regex : %s does not include sender
inbound MTA regex : %s excludes sender
outbound MTA regex : %s does not include recipient
outbound MTA regex : %s excludes recipient
message size %s exceeds channel limit %s
message size %s exceeds mta limit %s
message size %s exceed user limit %s
illegal body part : %s (mta restriction)
illegal body part : %s (user restriction)
channel bind failed
duplicate recipient eliminated
Success Reasons
access policy: free
negative: in-MTA (%s %s) sender (((%s %s)
block: in-MTA (%s %s) sender (%s %s) out-MTA (%s %s) recip (%s (%s)

Table 4.1: Authorisation Disagnostics

- sender
- sending MTA
- recipient
- destination MTA
- message size
- a diagnostic message

Fields are space delimited and double quoted where necessary.

The possible diagnostic messages are shown in Table 4.1.

The `(%s %s)` fields above are filled in with inbound and outbound rights i.e. in/out/both/none/unset

`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)

Finally, the user table is where both the recipient and originator are looked up to determine access rights if none of the other tests proves conclusive. 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.

`content-excludes=<content>:` A list of contents 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/:\
  content-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
```

4.17.3 Statistical logging

Each line in the statistics log ('stat') contains :

- a success indicator (e.g. 'ok' or 'failed')
- unique message id
- inbound channel→outbound channel

`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.

An example of this format might look like the following:

```
822-local->pss:free
822-local->local:block
822-local->slocal:free
822-local->smtp:free, sizelimit=5000, warnsender=smtpwarnsender
822-local->janet:block, sizelimit=4000
x400in84->x400out84:none
x400in84->local:free
822-local->x400out84:block, sizelimit=10000, warnsender=restriction
smtp->smtp:block, test
```

The next table is the mta policy table in file *auth.mta*. This governs the policy for a given mta if the channel is not `free` or `none`. The key used in this table is 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.

`requires=<pattern>:` This mta requires that the user being authorised to send to this matches the given pattern. This is a standard regular expression as specified in *regex* (3).

`excludes=<pattern>:` The mta requires that the user being authorised does not match the given pattern.

`content-excludes=<content>:` The mta requires that the content type is not one of those given here. These are standard content strings as defined in the tailoring section.

- 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.

4.17.2 Table formats

There are three tables associated with authorisation, plus the tailor variables. 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 fall back 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, with text taken from the given file.

warnrecipient=<file>: Send a warning message to the recipient with text taken from the given file.

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.)

4.17 PP authorisation

Authorisation is applied after address parsing 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*.

NOTE:	It is intended that warnings may be sent to recipients and senders regardless of the channels status. This functionality is not yet available.

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.

4.17.1 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 must enable the message by specifying 'in' or 'both' for the inbound channel and 'out' or 'both' for the outbound channel. The message may not be transferred if none of the four methods enable it. Therefore the default is to not let messages through, the action on the configuration to enable it.

`negative:` Any one of the four 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.

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

4.16.1 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 is carried in the initial protocol handshake. This may be overridden for later messages.

Each (inbound) channel has an "auth-table" variable, which identifies a table which is used in the manner described below. The authentication table has three columns. These have meaning:

1. String encoded address, in the preferred format of the channel

   - X.400 O/R Address
   - RFC 822 address

2. List of valid UIDs

3. Password (as in /etc/passwd)

Example files:

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

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

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)
#
```

Figure 4.12: Example Shell Table

4.15 Supporting unix style mailboxes

By default, the *822-local* delivers in an MMDF style mailbox format. That is there is a delimeter string, the message then another delimeter string.

If however, you require the more traditional unix style mailbox format (A line starting with a `From␣` line up to the next such) then this is easy to emulate by having the *shell* channel call the standard unix mail program */bin/mail* in the same manner as *sendmail* calls it to deliver messages. However, make sure this has not been overwritten by the PP replacement for this program.

This can be achieved by a channel configuration such as

```
j.onions:jpo,,/bin/unixmail -d jpo
```

It could also be achieved in the *.mailfilter* file with a syntax like

```
pipe "/bin/unixmail -d jpo";
```

Finally, the *.mailfilter* has an emulation of the this form of delivery, accessible with the keyword `unixfile`. This is described in more detail in *Volume 5*.

4.16 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 pre-requisite.

```
#
# 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 List Table

4.14 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>]
            "," <command line to execute>
```

`address:` specifies the recipient address e.g `shellprog` or the X.400 form `/S=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*.

`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 kill the commands off.

`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 3.1 on page 33.

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.

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

`listname:` The name of the distribution list e.g `listname` or in X.400 perhaps `/S=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 *The PP Manual: Volume 3 – Users Guide*). Anyone with one of the specified uids may use *mlist* to modify the contents of the list.

`filename:` 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 *tbldfldir* 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.

`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 three and only three 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*.

If you have a large sendmail alias file to convert, you may find the utility *make-lists*(8) 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.

```
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 X.400 Outbound Channel Table

4.12 The X.400 (1984/1988) inbound table

This table is used by the inbound X.400 channels, and allows the channels to obtain the MTA name and other information from the OSI calling address. The syntax is the same as the X.400 outbound table, only the key is different.

The key here is the information given in the session connection of the remote calling host; the values are as indicated for the outbound table. It is encoded in the string encoding for presentation addresses using no macro substitution. Therefore, the keys look pretty horrible. It is usual to maintain these tables with keys in macro form and run them through the utility *nsapexpand* to expand the macros.

The matching of incoming information with an entry in this table provides the major security checking. Again an entry `default` in the table may be used to default many of the fields.

An example is shown in Figure 4.10 (again lines are folded for clarity).

```
"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 X.400 inbound Channel table

4.13 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:

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.

Obviously, this information is dependent on the capabilities of the channel (a 1984 based channel will not be able to use the 1988 protocol).

`trynext:` A key to another entry in the table. If a connection using the current information fails, lookup this key and try making a connection using that data.

`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 this PRMD.
`all`	keep all tracing information

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.

`rname:` The "real" name of the mta. This is an entry that is suitable for lookup in the domain table. It is not used for the outbound channel.

`other:` Other information. This is a string and is interpreted by the channel to allow other types of operation to be specified. It is not currently used in the X.400 outbound channel.

If an entry with a key `default` exists in the table, then this may be used to set default setting for all entries. The value `default` is looked up first and values filled in from that setting - then the real entry is looked up and those values override the default entry.

An example is shown if Figure 4.9 (lines are folded for clarity).

4.11 The X.400 (1984/1988) outbound table

This table is used by the outbound X.400 channels, allowing them to map the host name to the information needed for addressing and connecting to the remote MTA. The key 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 white space and/or commas.

```
mta key:rmta=xx,rpass=xx,rpsap='"591"/Janet=000005110000',
```

The `mta key` is the key from the address lookup. This is the remote host to which the outgoing connection it to be made. The following keys are recognised in the value field:

`lmta:` The name of the local mtaname to pass in the RTS connect parameters. This defaults to local site name if not set.

`lpass:` The passwd of the local site to pass in the RTS connect parameters. This defaults to the default password specified in the tailor file.

`rmta:` The name of the remote mta to pass in the RTS connect parameters. If this is not set, then no authentication will be specified in the RTS connect.

`rpass:` The name of the remote password to pass in the RTS connect parameters. The is no default for this and it should be set if the `rmta` is set.

`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.

`lpsap:` The local presentation address. This is only necessary if you need to claim 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 governs the mode passed initially, it does not imply that the channel will work in TWA mode.

`type:` The type of RTS connection. This is one of the following:

4.9.3 The known RFC 1148 gateway table

The *rfc1148gate* table has the same syntax as the *rfc2or* table. It's entries represent gateways between X.400 and RFC 822 (via RFC 1148). This tables is used in the cases where RFC822 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 RFC1148 gateway, then the mapping can proceed in the expectation that the other end can unwrap the message.

An example RFC1148 gateway file is shown in Figure 4.8.

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

Figure 4.8: Example RFC 1148 gateway table

4.10 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 to deposit mail to a specific mailbox address, to be dealt with by the user agent. The format is:

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

The unix ID is either a password username, or a userid groupid pair separated by a slash. If this is the case, no password entry is looked up and at least a home directory must be given.

If no directory component is present, then the user's directory will be taken from the */etc/passwd* file. If no mailbox is present, the system default is used.

For example:

```
Alina.DaCruz:alina /cs/research/dsa/users/vs1/alina
John.Taylor:jtaylor /cs/research/dsa/users/vs1/jtaylor
Postie.Pat:pp /cs/research/dsa/vs2/pp
bug-filer:32767/1001 /usr/spool/mail bugs
```

```
#
#       -*** 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 Or2rfc Table

```
#
# 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 Rfc2or Table

Note that the default local hub (`cs.nott.ac.uk`) should have a null `hubname`.

Now for `physics.nott.ac.uk` create the following entry in the table:

```
physics.nott.ac.uk : physics.nott.ac.uk 0 local physics
```

Finally an entry for each user in each hub should be placed in the main *aliases* file. This entry should have the type `822-synonym` or `x400-synonym`.

With these actions taken, PP will then act like two or more hubs.

For example, if a message came in for `jpo@physics.nott.ac.uk` then the lookup in the *domain* table would identify this as a local address for the hub with the name `physics`. PP would then lookup the tables `physics-aliases` and `physics-users` to find out how to deliver the message.

If `jpo` sent out a message, his user agent would probably fill in his address in terms of the default local hub (`cs.nott.ac.uk` in this case). PP could then correct this address to refer to the right hub by looking up the information placed in the main *aliases* file (`822-synonym` and `x400-synonym` entries).

Another use of the `x400-synonym` and `822-synonym` comes about when a user moves. In which case, the mail administrator could redirect that users mail by simply putting a relevant `822-synonym` or `x400-synonym` entry in the aliases file. This entry should be of the form:

```
<oldaddress> ":" "822-synonym" <newaddress>
```

4.9 RFC 987 mapping tables

These mappings are used in gateways to convert between X.400 addressing and RFC822 addressing. The mappings involved are not algorithmic making neccessary the use of such tables.

4.9.1 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.

4.9.2 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.

```
#
#
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)
#
torch.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 Channel Table

4.7 The channel table

Every remote host/MTA has an entry in this table - *submit* uses it to discover the channel and in some cases relaying host, that must be used to send to the host in question. See Section 6.5 on page 96 for a method of building this table. The format of an entry has the form:

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

`key:` is the fully qualified name derived from the *domain* table.

`mtaname/channel:` the `mtaname` derived from this lookup will be used as a key in the appropriate `channel` table. If there is no `mtaname` present, then the mta key 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.

There may be a number of possible channels for each host. Figure 4.5 illustrates an extract of this table.

An entry with an empty RHS is allowed and indicates that this is a valid domain name, but not a valid delivery route. This is usually used to allow checking to proceed when the address should be mapped to a different form of address.

4.8 One MTA locally supporting multiple organizations

In some situations one may want PP to behave like two or more distinct mail hubs e.g. `cs.nott.ac.uk` and `physics.nott.ac.uk`. This behaviour is possible using the `822-synonym` and `x400-synonym` keywords in the *aliases* table and the `local` keyword in the *domain* table.

To get this behaviour, the following actions should be followed.

First create, in separate files, the *aliases* and the *users* tables for each other hub. These files should have the names of the form `hubname-aliases` and `hubname-users`. So for `physics.nott.ac.uk`, create the *aliases* table `physics-aliases` and the *users* table `physics-users`.

Then place one entry for each local hub in the *domain* table. This entry should have the form:

```
<fullname> ":" <fullname> <lower>":"<upper> "local" <hubname>
```

```
#
# 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
#
#
#       -*** 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 Or Table

conversion in accordance with the RFC 987. Lines are encoded in the form:

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

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

Values are separated by white space, and so any space characters, must similarly be preceded by either "\" or else the value should be encapsulated in double quotes.

The characters "\" and "." in a `<value>` field must always be preceded by a "\".

The *or* table is used to evaluate O/R Addresses. The `<key>` is encoded as dmn-orname (see RFC 987). The default algorithm for O/R Name analysis should be considered as "tree-walking". The O/R address components are sorted, and components are looked up: first level 1; then level 1 + level 2 and so on. So repeated lookups are carried out, using this table.

The RHS has four possible encodings:

`valid` This means that this part of the tree is OK.

`mta <mtaname>` This identifies the mta `<mtaname>` as being associated with this vertex of the tree. This `<mtaname>` is a key to the PP channel tables. The search should NOT stop at this point, as further tree searching may determine a more optimal site.

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

`local` This identifies a local O/R Address. The remainder of the O/R Address should be used to determine a local mailbox.

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 unique key with which to reference the channel table.

- The `cs.nott.ac.uk` domain will match the explicit name `cs.nott.ac.uk` and will allow at most one domain component below that, such as `john.cs.nott.ac.uk`.
- The `cs.ucl.ac.uk` domain will only match the exact domain name `cs.ucl.ac.uk`. No subdomains at all are allowed by this entry.

In this way the routing for a host can be determined from the tables by resolving the host's top level domain, and then obtaining relaying information for that top level domain.

An example is shown in Figure 4.3.

```
# top level entries
edu:edu 1:*
mil:mil 1:*
pt:pt 1:*
# local entries
cs.ucl.ac.uk:cs.ucl.ac.uk
cs.ucl.ac:cs.ucl.ac.uk
cs.ucl:cs.ucl.ac.uk
cs.nott.ac.uk:cs.nott.ac.uk
cs.nott.ac:cs.nott.ac.uk
cs.nott:cs.nott.ac.uk
cs:cs.nott.ac.uk
#
# NRS derived entries
alass.nott.ac.uk:alass.nottingham.ac.uk
alass.nott.ac:alass.nottingham.ac.uk
alass.nott:alass.nottingham.ac.uk
alass.nottingham.ac.uk:alass.nottingham.ac.uk
alass.nottingham.ac:alass.nottingham.ac.uk
alass.nottingham:alass.nottingham.ac.uk
clan.nott.ac.uk:clan.nottingham.ac.uk
clan.nott.ac:clan.nottingham.ac.uk
clan.nott:clan.nottingham.ac.uk
clan.nottingham.ac.uk:clan.nottingham.ac.uk
clan.nottingham.ac:clan.nottingham.ac.uk
clan.nottingham:clan.nottingham.ac.uk
```

Figure 4.3: Example Domain Table

4.6 The O/R address table

The *or* family of tables provide mappings between x.400 O/R names and RFC 822 domain names, to be used by channels concerned with

on the RHS is used again as a LHS key until the LHS and RHS are identical - at this stage the fully qualified name has been found and the *channel* table will be consulted.

The table may include not only MTA or host names, but partial domains. If a full match on a host name is not found in the *domain* table, the constituent dot-separated parts of the host name - that is, the domains which comprise the name - will be used as keys for repeated searches of the table. If a key is found for a partial match then a second parameter on the RHS is checked for. This is a range specifying how many subdomains are allowed for this domain.

Entries in the table may also contain the keyword `local`. This indicates that the domain is a local hub. This feature is described in section 4.8.

So an entry in the domain is of the form:

```
<aliasname> ":" <fullname>
<fullname>  ":" <fullname>
<partial> ":" <name> <lowerbound>":"<upperbound>
<fullname> ":" <fullname>
```

The lowerbound and the ":" may be missing, in which case the default lower bound is set to `0`. If the whole parameter is missing both upper and lower bounds are taken as `0`. The special value `*` is used to indicate any number of domains for the upperbound. This allows for wildcarding domains such as those obtained from the NRS partial domain table, or for nameserver hosts. It also allows wildcards not to be used as valid addresses, by insisting on at least one domain component. The following example will illustrate some uses of the syntax:

```
edu:edu 1:*
fr:fr *
cs.nott.ac.uk:cs.nott.ac.uk 1
cs.ucl.ac.uk:cs.ucl.ac.uk
```

This implies the following actions:

- The `edu` domain will match all domain names with the most significant component of `edu` and at least one subdomain. Thus, `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 `fr` domain matches all domain names with the most significant component of `fr`. This includes `fr` itself.

4.4 The users table

The *users* table determines the local delivery channel for a particular local user. It is formatted in the form:

```
<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 local machine.

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 RFC822 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
warning:shell
#
```

Figure 4.2: Example 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 `warning` shows the *shell* channel rather than a local delivery channel.

4.5 The domain table

The *domain* table is used by submit to derive a fully-qualified domain from any aliases or short-form names. See Section 6.5 on page 94 for a description of how to go about constructing this table. The value found

A value resulting from searching this table is then used as the key for repeated search. This continues until no value is found. The last value to be derived will indicate which delivery channel is appropriate; if the final value is for a local address then that value will be used as a key to lookup the *users* table; if the final value is an RFC822 address an RFC822 delivery channel will be used; or if the final value is an X400 address an X400 delivery channel will be used.

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

```
################### extract of 'aliases' table ################
#
mailgroup-request:alias Irene.Hassell
postmaster:alias Irene.Hassell
pp:alias postmaster
#
a.dacruz:synonym Alina.DaCruz
alina:synonym Alina.DaCruz
#
jpo:822 jpo@uk.ac.nott.cs
#
pc:x400 "/I=P/S=Cowen/OU=cs/O=nott/PRMD=uk.ac/ADMD=gold 400/c=gb/"
#
```

Figure 4.1: Example Alias Table

The table includes an entry for a local alias `mailgroup-request` The value `Irene.Hassell` is used as a key for a further search, but will not replace the alias in the message header; if no match is found, and since the entry is of type `alias`, the address is assumed to be a local one and 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 entry indicates 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 `jpo@uk.ac.nott.cs` would be parsed and routed on an RFC 822 delivery channel.

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

```
"/I=Peter/S=Cowen/OU=cs/O=nott/PRMD=uk.ac/ADMD=gold 400/C=gb/"
```

would be parsed and routed on an X.400 delivery channel. The quotes (") are needed as the entry contains a space which is normally a separator.

4.2 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.

4.3 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. j.p.onions to j.onions)

It is constructed as follows:

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

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

`alias:` local user name is specified, but this name does *not* replace the original value. Aliases are not expanded for originators whereas synonyms are.

`synonym:` local user name is specified, and this name replaces the original value.

`822:` remote RFC822 user address is specified

`x400:` remote X400 user address is specified

`822-synonym:`

`x400-synonym:` these two values are used when one MTA is being used to look like more than one mail hub. This feature and these two values are described in section 4.8.

Chapter 4

PP Tables

This chapter describes the purpose and syntax of the various tables used by PP. There are five basic tables used by PP, the *aliases* table, the *users* table, the *domain* table, the *channel* table and the *or* table. How these are combined and used to deliver and route messages is described below.

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 Basic operation of the tables

When a message arrives, *submit* identifies if it is to be delivered by the local MTA or if it is to be routed to a remote host/MTA.

If the message is to be delivered by the local MTA, *submit* references the *aliases* table to translate any aliases to the real addresses they point to; and then looks at the *users* table to determine which local channel is appropriate for that address (for example, ordinary 822-local or structured local).

If the message is for a remote host/MTA, *submit* references either the *domain* table to discover the fully-qualified for RFC822 style addresses, or the *or* table for X.400 style addresses. If necessary the address is mapped from one form to another (e.g. RFC822 to X.400) by use of the mapping tables *or2rfc* and *rfc2or*. This produces the canonical name for the host. The *channel* table is then used to determine a list of channels that can be used to reach the host in question.

At this stage the channels are ordered in the preferred of use and then checked in order for authorisation until a channel/mta/user succeeds. Once this is achieved the necessary reformatting channels are calculated and the message is then accepted.

3.8 How channels work

A channel is called by the *qmgr*, and initialised. Then it is told 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 suceeded, increment the reformat counter
- If a delivery suceeded, set the status to DONE or DR, dependent on whether a positive Delivery Report is needed.
- If the operation failed, set the status to DR.

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 *qmgr* will then 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.

This section describes the tailoring requirements of some of the common channels.

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 `shaper` as the channel alters the shape of the message.

`content-in` This field should contain the inbound content, either `p2` or `p22`.

3.7.6 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`.

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`.

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`.

3.7.5 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. The tailor file entry for such a channel should be constructed with the following provisos:

`type` This field should be set to `shaper` as the channel alters the shape of the 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 doesn't recognise, the *rfc934* channel will fail. The key strings `hdr.822` and `ia5` are hardwired into PP. They can be changed by editing the relevant entries in the file *Lib/pp/static.c* within the source tree.

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 `shaper` as the channel alters the shape of 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)).

3.7.3 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

content-out

bptin

bptout

adr

subadr
```

Note that it may be necessary to set the field `adr-order`. The default for this field is usonly. If you have some uk ordering of domains in your lists you should set `adr-order=ukpref` or `adr-order=uspref`

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

```
chan list prog=list,show="List channel",type=both,
          table=list,chanout=dr2rfc
```

Note that the *list* channel can expand sublists of a list. This will improve performance if there is a number of nested lists *BUT* 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. To enable it, the `info` field should be set to the string `dosublists`.

3.7.4 The rfc1148 Channels

The *P2toRFC* and *RFCtoP2* channels perform 1148 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 `1148`.

The output of the 1148 channels maybe be altered via use of the `info` tailor entry.

With the *P2toRFC*, if the `info` 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 `info` entry contains the string `rfc987`, the conversion from rfc822 to P2 will adhere to the rfc987 mapping rather than the rfc1148. Note that these mappings are very similar anyway.

`-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.

`-percent:` This flag indicates that the `%` character should be treated as a `@` character when parsing addresses. This flag also indicates that all domains in addresses should be normalised. By default, only the next hop of an address is normalised.

`-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. e.g. `*.icl.stc.co.uk` or `stl.stc.co.uk` but not `frodo.*.co.uk`. WARNING: this flag should be used with care and should not normally be required.

`-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.

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 it's 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 info string for such a channel might be

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

An example tailor file entry for such a filter is shown in figure 3.4.

ODIF filters

There exist some filters that will convert between BBN Slate format and ODIF. These filters are not publically 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).

RFC 822 Filter

The RFC 822 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" form. A normalised header is one in which all address are fully qualified and are in the correct order. For this reason, the `bptin` and `bptout` fields in any associated tailor file entry should be some extended version of `hdr.822`, e.g `hdr.822-us`, `hdr.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)

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

```
rfc822norm -822 -striptrace -jntsender $(822sender)
```

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

The filter takes the following options:

`-822:` Format source routes according to RFC 822.

`-733:` Format source routes with a "%" notation (default).

`-bigend:` Format domains big–endian.

`-littleend:` Format domains little–endian (default).

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

`-striproutes:` Removes 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.

```
chan    822touk   prog=fcontrol,type=shaper,
                  bptin=hdr.822,bptout=hdr.822-jntmail,
                  info="rfc822norm -jnt -jntsender $(sender) -fold 50"

chan    g3faxtotxt prog=fcontrol, type=shaper, bptin=g3fax,
                  bptout=ia5, info='removebp "Group 3 Fax"',
                  show="Fax -> Text Mapper"

chan    dmd2odif  prog=fcontrol, type=shaper, bptin=dmd,
                  bptout=odif, info=f-dmd2odif,                    10
                  show="Diamond -> Odif"
```

Figure 3.4: Example 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 "********************"                                        10
exit 0
```

Figure 3.5: Simple shell script filter

Key	Expansion
`sender`	the address of the sender of the message (format unspecified)
`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 (format unspecified)
`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
`recipient`	the recipient in local format
`userid`	either the username if available else the uid
`groupid`	the group id of the user being delivered to
`size`	the message size
`channelname`	the name of the channel doing the delivery

Table 3.1: Filter Control Expansion Macros

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`.

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

Example tailor file entries of filters running in conjunction with *fcontrol* are shown in Figure 3.4.

Trivial filters such as voice to ia5 mappings can be done by a simple shell script of the type shown in Figure 3.5.

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

- 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 bust be in the `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).

3.7.2 Filter Control and Common Filters

Filters can be called via a special channel called *fcontrol*. The 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 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*.

`info:` The pathname of the filter's executable program should be placed in the `info` 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 3.1. 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`

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"...
```

X.400 inbound channel

The X.400 (1984) inbound channel (*x400in84*) takes note of the `info` 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 looked up in the channel table 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 x400 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.

UUCP channel

The uucp channel interfaces to the UUCP system. It uses the `info` 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-out prog=uucp-out,show="UUCP outbound channel",type=out,
    adr=822,adr-order=ukpref,table=uucp
    info="uux=/usr/bin/uux - -r,host=nott-cs"
```

822-local channel

The *822-local* channel has only one tailoring feature. If the `info` string is set to `resticted` then the *.mailfilter* file processing is reduced in functionality as follows:

3.7.1 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 part 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 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
```

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.

Finally, the smtp server makes use of one additional tailoring variable in the channel specification. If the `info` element of the channel tailoring entry contains the string `sloppy` 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 to a host name).

The smtp outbound channel has only one special option. It can be given a flag `-p port` to tell it to connect to a different tcp port other than

3.6 ISODE runtime tailoring

3.6.1 Queue manager tailoring

Information relating to the MTA, its *qmgr*, and their addresses are obtained from the */etc/isoentities* and take the 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 most usually the name of the machine. The second is the qmgr service; this should not be changed. The same goes for the other fields, except for the last one. This 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`.

3.6.2 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
```

This shows the service name `tsap/p1`, the transport selector `"591"` and indicates the location of the program. You should use a numeric IA5 form of TSEL for X.400(1984) compliance.

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

3.7 Tailoring of Common Channels

Channels perform operations on messages. The following terms are applied to channels which are endpoints. 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 message 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.

messages with header mappings and no body part conversions to go through and RFC1148/987 gateway.

Channels that change formats without loosing information (except rfc987/1148 channels) should be marked as `conv`. Channels which loose data (such as, say, G3Fax to Ia5) should be marked as loss (they should also be given a high cost).

`probe:` 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.

An example of a channel tailoring is shown in Figure 3.3 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,
        bptout="ia5,hdr.822", content-out=822,
        table=local,access=mts

chan    x400in84  prog=x400in84,show="X400 inbound",type=in,
        adr=x400, adr-order=ukpref,table=x400in84

chan    qmgr-load prog=qmgr-load,show="Loading the QMGR",          10
        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,table=none,info=f-fax2txt,
        show="Fax -> Text mapper"
```

Figure 3.3: Example Channel Tailor Entry

`table=value:` A table associated with this channel.

`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 inbound channel. 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

`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`.

`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 4.16 on page 55.

`conv:` This variable specifies what affect the channel has on a message. This is used to check on conversion restrictions. In particular the X.400 and RFC1148 conversion semantics. It may take one of the following values:

Value	Meaning
`none`	channel does no conversion
`1148`	channel does RFC1148/987 header mappings
`conv`	channel does conversion
`loss`	channel does conversion with loss

The default value is `none`. Channels that do mapping of P2 to RFC822 *headers* and the reverse should use the value `1148` to indicate that conversion can be carried out even in the presence of the conversion-prohibited flag of X.400. This will allow

`adr=value:` The type of addressing used by this channel. The value is one of those below:

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

`subadr=value:` The type of subaddressing used by a channel, if appropriate. Currently this is only relevant in conjunction with a definition of `adr=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

This list will change in the next release.

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

`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.

`bptin=value:` A list of body parts that the channel accepts as input. Volume 3 of this document provides more information. This must be one of the types defined in the `bodypart` tailoring item defined in Section 3.2.1 on page 18.

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

channels. The value refers to the channel that should be used to deliver receipt notifications if required or errors if a message fails. It should be set to *dr2rfc* if the specified channel cannot handle X.400 style delivery reports, or else an outbound channel if it can.

`content-in=value:` The content of this channel if it allows incoming messages. It is normally either `p2`, `p22` or `822`. It should be empty if the message is submitted in structured form.

`content-out=value:` The content of this channel if it allows outgoing messages. Values are as for `content-in`.

`cost=value:` The cost of running this channel. 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.

`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` or `user`. 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.
`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.

`info=value:` This contains channel specific information that can be used for a variety of purposes. For example, the `info` 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 info field as a sub-process.

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

Figure 3.2: Example Table Tailoring

associated with it; then comes a list of key/value pairs which provides more information on the channel.

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).

`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.
`shaper`	A reformatter that changes the shape of the message.
`warn`	A warning channel.
`delete`	A message deletion channel.
`qmgrload`	A queue manager loading channel.
`debris`	A debris collection channel.
`timeout`	A timeout channel.

`chanout=value:` This value must be set for any inbound or bidirectional channel. It should not be set for any other types of

```
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                                  10
isodelog tsaplevel      level=pdus
isodelog ssaplevel      level=pdus
isodelog rtsaplevel     level=pdus
```

Figure 3.1: Example Of Log Tailoring

`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.

`flags=value`: A set of flags that specify 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 doesn't require rebuilding the database and changes take immediate effect). Linearly searched files are still built into the database.

An example of table tailoring is shown in Figure 3.2.

3.5 Channel Tailor File Entries

Channels are perhaps the most complex aspect of tailoring. There are several types of channel. 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

`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 :-

`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

By default, all the ISODE logging is directed at the normal log. An example of logging tailoring is shown in Figure 3.1.

Notice that for a given program or channel, the logging information can be diverted. If a logging entry starts with the name of the binary then that line is specific to that program. E.g.

```
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 message from *submit* end up in the file *submit* in the log directory.

3.4 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, it is not normally used.

`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`.

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

`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 `close` and `create`.

`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.

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 th enumber of trace fields may contain. If it has more than this number, the message will be rejected as looping. The default is 25

x400_mta This is the name of this MTA in the X.400 world. It defauls to the same as loc_dom_mta.

3.3 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. 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.

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*.

`level=value:` This adds to the logging levels the given value. 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).

The default is `low`.

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

The default is `warnings`.

Note: Warnings are not used in this release.

`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 one day.

Note: Warnings are not used in this release.

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

The default is two warnings.

Note: Warnings are not used in this release.

`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 overriden by specifying two other times in hours after the normal return time. The first or these being the return time for high priority messages and the second for low priority messages.

The default normal time out is three days 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.

`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

`mboxname:` This is the maildrop file used by the 822-local delivery channel for delivering local messages into users' home directories. The default is *.mail*.

`mailfilter:` This is the mailfilter file used by the 822-local delivery channel (see *The PP Manual: Volume 3 – Users Guide*). The default is *.mailfilter*.

`sysmailfilter:` This is the mailfilter file the 822-local delivery channel uses if the user does not have one. The default is */usr/local/lib/mailfilter*.

`delim1:` This is the message delimiter appended to the maildrop file before each message. The default is `"\001\001\001\001\012"` i.e. four control-A's and a newline. (Beware, the *C* syntax `\n` does not work (yet)).

`delim2:` This is the message delimiter appended to the maildrop file after each message. The default is `"\001\001\001\001\012"` i.e. four control-A's and a newline.

`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 4.17). The default is `block`. This determines the policy for any inbound/outbound channel pairing not explicitly entered in the channel authorisation table *channel-auth*.

`authloglevel:` This sets the level of authorisation logging. Three levels are defined :

- `low:` One line of statistical information is generated for each recipient, giving the outbound channel if authorisation succeeds, or the first (of possibly several) channels to be tested which failed. (It is anticipated that the tables will be configured so that channels will be tested in order of ascending cost.)
- `medium:` One line is generated for each potential output channel which fails authorisation tests in the cases where no successful channel is found, but only a single line where authorisation is successful for that recipient.
- `high:` As for `medium`, but a line is produced for every channel which fails authorisation tests, whether or not the recipient is ultimately successful.

```
"/OU=CS/O=NOTT/PRMD=UK.AC/ADMD=GOLD 400/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.

`postmaster:` This is the 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.

3.2.2 Optional tailoring variables

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

`pptsapd_addr:` This is the address that the *pptsapd* listens on for incoming connections. For example the *qmgr* uses it to connect to channel processes. The default value is: `Internet=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.

`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 sub-directory of the *cmddir*.

`formdir:` This is the directory where the simple programs run by the *fcontrol* are found. Not usually tailored, it defaults to being the same as *chandir*. 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 in any case should be a username found in the */etc/passwd* file.

by quote marks ("<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.

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.

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`.

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`.

loc_or: This is the local O/R address defaults given in X.400 RFC1148 encoding form. For example:

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* 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,
- and 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 6.3.2.

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

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 6.3.2.

2. Select one of the co0nfigurations shown below, and change into that directory.

3. read the *README* file, and carry out the instrcutions 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 Transport Agent, and does not contain a sophisticated X.400 user agent. It is hoped that the interface documented in Volume 2 will be useful in the development of new user agents or the interfacing of existing UA's. 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* emulation. This should be installed in the normal installation. 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)).

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 4.13 on page 53.

dirlist

The *dirlist* channel expands a distribution list into its component members. However, this list channel makes use of the X.500 directory to retrieve the members of the list. This channel is experimental in this version of PP.

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 4.14 on page 54.

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.

`removebp:` The filter is used to replace bodyparts that the MTA cannot support 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.

UUCP

A channel that will interface with the UUCP protocols.

> **NOTE:** This channel has not been used very much. It is still rather unstable as the development sites use UUCP traffic very little. You have been warned!

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 (1984) and p22 (1988). It is used on an incoming x400 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 (1984)` or `p2 (1988)` to an RFC 822 style header. This conversion is performed according to RFC 1148[Kil89] .

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 (1984)` or `p2 (1988)`. 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].

822-local

The *822-local* channel is used to deliver messages to local users. It can deliver in a number of formats. For more details see Section 4.15 on page 55 and also *The PP Manual: Volume 3 – Users Guide*.

slocal

The *slocal* - 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(1984) 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(1988) 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.

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 *delete* 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 it's originator, telling him or her that PP timed out the original message.

dr2rfc

The *dr2rfc* channel constructs delivery receipt notifications or error messages and then submits them to the *qmgr* for return to the sender.

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.

`PPUSER:` This is the username that most PP programs are installed under. It is usually `pp`, but need not be.

`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) 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.

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.

- inet
- ubc
- dexpand
- x25b
- yorkbox

`CC:` The C compiler to use. The PP system has been compiled with a number of C compilers.

Warning: If you wish to use the *gcc* compiler on `sparc` based system, you must compile files making use of dbm style routines with *cc*. (The compiler will normally warn about these files).

`CCOPTIONS:` This contains global C compiler options. If your include files for `ISODE` are in a non standard place, a "`-I`" flag or two may be necessary here.

`LIBCCOPTIONS:` This contains global C compiler options to use when compiling files to place in libraries. In normal use this will be the same as `CCOPTIONS`.

`LDOPTIONS:` This variable contains global loader flags. It is normally empty or just contains the "`-s`" flag.

`PEPY/POSY/ROSY:` These variables are the pepy, *posy* and *rosy* programs. Normally they just contain those names, but may be full pathnames if they are not in the standard search path.

`X11:` If you have X11 then set this variable to true. Leave it empty and no X-based programs will be used.

`LIBX:` The X11 libraries. This contains a list of the X libraries necessary to compile the few X based programs in the distribution. This distribution is aligned to the X11R4 release, it will *not* run under earlier versions of X.

`LINT:` The man lint(1) command. This is not normally used unless you are doing development work.

`LINTFLAGS:` A list of global lint flags.

`PGMPROT:` This is the default mode to install programs as. Some programs need special modes, but these are handled separately.

figurations. However there are a number of protocol and experimental channels which can be included if required. This variable specifys which of the optional channels and protocols you wish to support. Currently this consists of a list made from the following.

`822-local:` Basic RFC-822 delivery

`list:` Support for distribution lists.

`grey:` Support for JNT Mail protocol (Grey book).

`shell:` Support for delivery to programs.

`slocal:` Support for structured local delivery (experimental).

`smtp:` Support for the SMTP protocol.

`uucp:` Support for the UUCP protocols (primitive).

`x40084:` Support for the X.400 P1(1984) protocol.

`x40088:` Support for the X.400 P1(1988) protocol.

`dirlist:` Experimental X.500 distribution list channel.

`MANDIR:` This is the directory under which manual pages are placed. The installation process understands about most layouts of manual pages, so this is usually a base directory.

`MANOPTS:` This variable configures the style of manual pages to be installed. It can take the following values:

Option	Format
`-bsd42`	man<N>/file
`-bsd44`	cat<N>/file.O
`-ros`	man<N>/file – using /etc/install
`-sys5`	<a|p|u>_man/man<N>/file
`-local`	manl/file.l
`-l`	man<N>/file.<N>l
`-hpux`	HP manual style

`NIFTPSRC/NIFTPINTERFACE:` These two variables control the PP interfacing with the unix-niftp code. These variables are ignored unless grey book mail is being installed. The variables should be set to the source directory of the unix-niftp package, and the particular interface the grey book channel will use. Suitable interface settings include:

- sun

`FORMDIR:` This directory contains the formatting programs that are run by *filtercontrol*. These are either binary executables or shell scripts.

`TOOLDIR:` This directory is where a number of miscellaneous tools are placed. These are useful programs that aid in the day to day running of PP but are not usually used by the PP system.

`LOGDIR:` This directory is where the logging files are written to by default. These logs are used for a variety of purposes, including debugging and statistics gathering. You should allow for a reasonable amount of space.

`TAILOR:` This specifies the location of the PP tailor file. It is important you get this right. Almost everything PP variable can be changed in the PP tailor file, but obviously the location of the PP tailor cannot. If you get this wrong, you will need to relink all the PP binaries.

`QUEDIR:` This is the directory under which the PP queue is maintained. As all messages sent are queued here, you should allow enough room to handle all the mail you are likely to receive. A busy site with just a day or two's network outage can gather several megabytes of mail.

`TBLDIR:` This is the directory in which all the PP routing tables and similar configuration files are kept.

`LIBSYS:` This variable contains all the external libraries that you may require when building most of the PP binaries. It should contain at least the `ISODE` library and possibly other libraries such as the *dbm*(3) library and any machine specific libraries you may need.

`LIBRESOLV:` This variable contains the library for the bind domain name server (or equivalent). It is only required if you are making use of the DNS (see section 2.1.2). Leave it empty otherwise.

`USRBINDIR:` This is a directory where binaries that are used by users are placed. This includes a simple mail interface, and some delivery time processors.

`OPTIONALCHANS:` PP requires a certain core set of channels to operate at all, plus a number which are used in all normal con-

Variable	Default	Description
CC	cc	The C compiler to use
CCOPTIONS	-O	C compiler options
LIBCCOPTIONS	-O	C compiler options specific to library files
LDOPTIONS	-s	Loaded options
PEPY	pepy	The *pepy* ASN.1 parser/compiler
POSY	posy	The *posy* stub generator
ROSY	rosy	The *rosy* remote operations stub generator
X11	true	does the sytem has X11
LIBX	-lXaw -lXmu -lXt -lX11	The X11 libraries - may need -lXext for X11R4
LINT	lint	The *lint* (1) command
LINTFLAGS	-hbuz	Lint flags
PGMPROT	755	Program protection mode
PPUSER	pp	The user id PP will run as
ROOTUSER	root	The user id for privileged channels
CHOWN	chown	The *chown*(8) command
CHMOD	chmod	The *chmod*(1) command
BACKUP	cp	A command to backup the old binary
INSTALL	cp	A command to install a new binary

Table 2.2: Make Variables that can be defaulted

Variable	Example	Description
CMDDIR	/usr/lib/pp	Directory for PP basic commands
CHANDIR	/usr/lib/pp/chans	Directory for channel programs
FORMDIR	/usr/lib/pp/format	Directory for the simple formatting channels
TOOLDIR	/usr/lib/pp/tools	Directory for miscellaneous shell scripts and debugging tools
LOGDIR	/usr/lib/pp/logs	Directory for PP logs
TAILOR	/usr/lib/pp/tailor	Location of the pp tailor file
QUEDIR	/usr/spool/pp	Location of the PP queue directory
TBLDIR	/usr/lib/pp/tables	The directory containing the table files
LIBSYS	-lisode	External libraries required - must include ISODE
LIBRESOLV	-lresolv	Resolver library for bind (empty if nameserver not required)
USRBINDIR	/usr/local/bin	Directory for user binaries
OPTIONALCHANS	list 822-local smtp	List of optional channels you require
MANDIR	/usr/local/man	Directory to place manual pages under
MANOPTS	-bsd42	Manual page installation options
NIFTPSRC	/usr/src/niftp	Directory containing the source to unix-niftp package. (Only required if you wish to run grey book mail).
NIFTPINTERFACE	sun	Niftp interface being used (restrictions as above)

Table 2.1: Make Variables that should be set

Chapter 2

Installing PP

This is a description of things to do when installing PP. The basic information is described in *pp-gen*(8), but this section gives the full details. It is recommended that you read this section before compiling or installing the software.

2.1 Compile Time Configuration

Most of the definitions that have to be set before compiling PP are given in *Make.defs* in the top level of the source tree, and *config.h* in the `h/` directory.

2.1.1 Make.defs

A number of *make* variables can be set. These variables come in two classes. The first are variables that are truly site specific. These are listed in Table 2.1. The second class are variables that you can probably leave as the defaults. These are listed in Table 2.2. For each entry a reasonable default is given.

A number of these variables are important, so care must be taken when configuring these variables. A description of each of these variables is now given, however Table 2.1 and 2.2 give a good overview of the variables.

`CMDDIR`: This is the directory into which most of the PP core commands are placed. This is usually the directory that contains subdirectories for channel and formatting programs.

`CHANDIR`: This directory is normally a sub-directory of `CMDDIR`. It contains all the channel programs that PP knows about.

1.7 Performance

Currently, not many statistics have been collected on the performance of PP, and little effort has been made on tuning as yet. However, some observed results may be useful.

PP has been running on a Sun 4/330 at UCL handling a large percentage of the electronic mail. This consists of around 50,000 messages a week consisting of around 175Mb of data. This system is also providing a directory service and does not show signs of saturation as yet.

1.5.1 MMDF

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

1.5.2 Sendmail

PP does not support sendmail style mailboxes by default. These can created by a variety of mechanisms listed later.

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

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 MTA's 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[Ros88] 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 1 and 4 MTA's. Reasons for choosing MTAs might include:

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

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[Dun] resolver library must be provided (see Table 2.1 on page 6).

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[RC87].

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

1.4 User Agents

This version of PP does not support 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[Ros86] user agent is considered for use with PP. MH should be configured with the SMTP transfer option which will connect to PP though 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.

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.

The basic distribution tape contains sources amounting to around 7Mb.

You will need approximately 40Mb of headroom extra to compile everything, although you can do it in stages if you don't have that much space.

The installed binaries will take up somewhere around 20-30Mb.

The routing table files plus aliases etc may take up about 1Mb if you have a lot of connectivity.

The *dbm* database built from these tables, perhaps another 1-2Mb

Of course you must also allow plenty of space for the mail passing through!

This suggests that you should allow about 50Mb of space for building everything, and when built, about 25Mb to run the system with whatever room you feel reasonable for spooling of messages (phew!).

1.2.3 Communications requirements

PP uses OSI/ROS for its internal communication, using the ISODE implementation of ROS and the lower layers[Ros89]. It is assumed that ISODE is installed and available on the system in question. ISODE ASN.1 tools are also required to build parts of PP. This version of PP assumes ISODE version 6.0.

1.2.4 X window system

The system has a very useful monitoring tool which is written using the X windowing system[2] 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 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 release 3 servers.

1.2.5 Quipu

This release has no requirements on the Quipu X.500 directory service agent. There is a prototype list channel that utilises the directory but this is optional. Future releases of PP will require a Quipu DSA.

[2] X Window System is a trademark of MIT

Chapter 1

General Considerations

1.1 Overview

Before rolling the tape onto a disk, 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.

1.2 General Requirements

1.2.1 Systems

PP is designed to be portable to a wide range of UNIX[1] systems. Configurations are given for the following systems:

- Sun 3 using SunOS 3.X
- Sun 3 using SunOS 4.X
- Sun 4 using SunOS 4.X

It is also known to run on the following systems:

- Vax Ultrix
- Mips

1.2.2 Disk space

A fair amount of space must be made available for the system. The figures given below are only a guide

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

Preface

This volume describes the installation and general operation of the PP software. More details about its history, development and objectives can be found in *PP Manual* introduction pages.

A general overview of PP can be found in the paper in the directory doc/ifip6.5-88[Kil88].

A guide to writing programs that interact with PP is given in *PP Volume 2: Programmer's Guide*.

A guide to the user visible parts of PP can be found in *PP Volume 3: Users Guide*.

C.1 An Example Nightly Script 112

E.1 io_ Interface 119

F.1 QMGR ROS 129

List of Figures

3.1 Example Of Log Tailoring 23
3.2 Example Table Tailoring 24
3.3 Example Channel Tailor Entry 28
3.4 Example Filter Control Tailoring 32
3.5 Simple shell script filter 34

4.1 Example Alias Table 41
4.2 Example Users Table 42
4.3 Example Domain Table 44
4.4 Example Or Table 46
4.5 Example Channel Table 47
4.6 Example Or2rfc Table 49
4.7 Example Rfc2or Table 49
4.8 Example RFC 1148 gateway table 50
4.9 Example X.400 Outbound Channel Table 52
4.10 Example X.400 inbound Channel table 53
4.11 Example List Table 54
4.12 Example Shell Table 55

5.1 Example of an O/R table to edit 66
5.2 O/R table after editing 67
5.3 Example X.400 outbound table 70
5.4 Example Channel Table 71
5.5 Example X.400 Inbound Table 74
5.6 Example X.400 incoming logs 74

6.1 Extract From the Console's Tailoring 89
6.2 Example Address file 102
6.3 Example delivery report format 102

A.1 An Example Tailor File 109

B.1 An Example Start Script 110

List of Tables

2.1 Make Variables that should be set 6
2.2 Make Variables that can be defaulted 7

3.1 Filter Control Expansion Macros 33

4.1 Authorisation Disagnostics 61

5.1 Example X.400 information for NASA 64
5.2 Example X.400 information for UCL 64

6.1 MTA Console Accelerators 86
6.2 Tailor File Entries to Console Keys. 88

E.1 Summary of MU flags 118

6.8.2 What to run each night 100
6.8.3 Saving Logs 100
6.9 Examining the PP Queue 101
6.9.1 Delivery Reports 101
6.9.2 Receipt Notifications 101
6.9.3 Warnings 101

A Example of the 'tailor' file 103

B Example of the PP start script 110

C Example of the PP script run nightly 111

D Obtaining X.400 tables 113

E Testing PP - the MTS User Interface 115
E.1 Overview 115
E.2 User input 117
E.3 mu internals 119

F Qmgr Protocol 120

5 Setting up X.400 Links **63**
5.1 Remote Site information 63
5.1.1 Conventions 65
5.2 Setting up the Routing Tables 65
5.2.1 Routing and mapping tables 65
5.2.2 Editing the O/R Table 65
5.2.3 Editing the X.400 Outbound Table 68
5.2.4 Channel Table Usage 70
5.2.5 Edit the Channel Table 71
5.2.6 X.400 Inbound Procedures 71
5.2.7 Edit the X.400 Inbound Table 72
5.2.8 How to Locate a Remote Site making a test connection to you. 73
5.2.9 How to make a test X.400 connection to a Remote Site. 75
5.2.10 A Common X.400 incoming connection problem. . 76

6 Managing PP **77**
6.1 Qmgr 77
6.1.1 Message Submission 77
6.1.2 Queue Reading 77
6.1.3 Active Channel 77
6.1.4 Passive Channel 78
6.1.5 Cleanups, timeouts and debris 78
6.1.6 Qmgr management and tailoring 78
6.2 MTA Console 80
6.2.1 The MTA Console's Display 80
6.2.2 Configuring The MTA Console 81
6.2.3 The Effect of the Mouse Buttons and Keys 83
6.2.4 Badness 86
6.3 Checkup procedures 88
6.3.1 ckchan 88
6.3.2 ckconfig 89
6.3.3 Checking of Addresses 90
6.4 Dealing with errors 92
6.5 Managing Tables 93
6.6 Adding new users and MTAs 97
6.7 Adding Filters, Shapers + EITs 98
6.7.1 Adding A New Encoded Information Type 98
6.7.2 Adding a new content type 98
6.8 The day–to–day running of PP 99
6.8.1 How to start up the PP system 99

3.3 Logging Tailor File Entries. 20
3.4 Table Tailor File Entries 22
3.5 Channel Tailor File Entries 23
3.6 ISODE runtime tailoring 29
3.6.1 Queue manager tailoring 29
3.6.2 X.400 tailoring 29
3.7 Tailoring of Common Channels 29
3.7.1 Protocol channels 29
3.7.2 Filter Control and Common Filters 31
3.7.3 The List Channel 35
3.7.4 The rfc1148 Channels 36
3.7.5 Shaping Channels 36
3.7.6 Miscellaneous Channels 37
3.8 How channels work 38

4 PP Tables **39**
4.1 Basic operation of the tables 39
4.2 The general syntax of table entries. 39
4.3 The aliases table 40
4.4 The users table 41
4.5 The domain table 42
4.6 The O/R address table 43
4.7 The channel table 45
4.8 One MTA locally supporting multiple organizations . . . 47
4.9 RFC 987 mapping tables 48
4.9.1 The O/R to RFC 822 Table 48
4.9.2 The RFC 822 to O/R table 48
4.9.3 The known RFC 1148 gateway table 48
4.10 The 822 local table 50
4.11 The X.400 (1984/1988) outbound table 50
4.12 The X.400 (1984/1988) inbound table 52
4.13 The list table 53
4.14 The shell table 54
4.15 Supporting unix style mailboxes 55
4.16 Authentication 55
4.16.1 Submit Authentication 56
4.17 PP authorisation 56
4.17.1 Access policy 57
4.17.2 Table formats 58
4.17.3 Statistical logging 60
4.18 Qmgr Authentication 60

Contents

1 General Considerations **1**
- 1.1 Overview . . . 1
- 1.2 General Requirements . . . 1
  - 1.2.1 Systems . . . 1
  - 1.2.2 Disk space . . . 1
  - 1.2.3 Communications requirements . . . 2
  - 1.2.4 X window system . . . 2
  - 1.2.5 Quipu . . . 2
- 1.3 Specific channel requirements . . . 2
- 1.4 User Agents . . . 3
- 1.5 Compatibility with older MTAs . . . 3
  - 1.5.1 MMDF . . . 3
  - 1.5.2 Sendmail . . . 3
- 1.6 MTA layout . . . 3
- 1.7 Performance . . . 4

2 Installing PP **5**
- 2.1 Compile Time Configuration . . . 5
  - 2.1.1 Make.defs . . . 5
  - 2.1.2 h/config.h . . . 10
- 2.2 Overview of channels . . . 11
  - 2.2.1 Mandatory channels . . . 11
  - 2.2.2 Optional channels . . . 12
- 2.3 Example Configurations . . . 14
- 2.4 Configuration of User Agents . . . 15
- 2.5 Checking the configuration of your system . . . 15

3 Configuring PP **16**
- 3.1 The general syntax of tailor file entries. . . . 16
- 3.2 Variable Tailor File Entries. . . . 17
  - 3.2.1 Mandatory tailoring variables . . . 17
  - 3.2.2 Optional tailoring variables . . . 18

PP

Volume 1: Installation and Operation

Peter Cowen
Alina da Cruz
Steve Kille
Julian Onions
March 18, 1991