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.

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.

An example command line:

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

where file `a1` contains:

```
# Start-of-MsgEnvPrm
-body hdr.822
```

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

```
-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 $
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.

After processing all inputs, a sequence of procedure calls to are made to *submit* the message (as outlined in Part IX). 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.

Appendix F

QMGR Protocol

This appendix 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],
                  qmgrstatus[S],
                  readchannelmtamessage[S] } ::= id-pt-qmgr-use

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

passiveChannel OBJECT
                PORTS { processmessage[S],
                      channelbegin[C] }
          ::= id-pt-passiveChannel
                                                                    40
```

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

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

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

-- OPERATIONS

                                        -- new message arrived
                                        -- Submit -> QMGR
newmessage OPERATION
            ARGUMENT MsgStruct
            RESULT NULL
            ERRORS { congested, protocol }                                  70
            ::= 0

                                        -- read a message
                                        -- Console -> QMGR
                                        -- QMGR -> QMGR
readmsginfo OPERATION
            ARGUMENT ReadMessageArgument
            RESULT MsgList
            ERRORS { congested, protocol }
            LINKED {msginfo}                                                80
            ::= 1

                                        -- QMGR -> Console
                                        -- Incremental update
msginfo OPERATION
            ARGUMENT MsgList
            ::= 10

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

                                        -- start up a channel
                                        -- Channel -> QMGR
channelbegin OPERATION
```

```
        ARGUMENT FilterList                                          100
                        -- type of message requested
        RESULT FilterList
                        -- what you are going to get
        ERRORS { congested, protocol }
        ::= 3

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

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

                                -- QMGR -> Console
                                -- Incremental update
chaninfo OPERATION
        ARGUMENT ChannelReadResult
                        -- info on ALL channels
        ::= 11                                                       130

                                -- control a channels behaviour
                                -- Console -> QMGR
channelcontrol OPERATION
        ARGUMENT ChannelControl
        RESULT PrioritisedChannelList
        ERRORS { congested, protocol, noSuchChannel, illegalOperation,
                authenticationFailure}
        ::= 6
                                                                     140
                                -- read about a mta
                                -- Console -> QMGR
mtaread OPERATION
        ARGUMENT MtaRead
        RESULT PrioritisedMtaList
        ERRORS { congested, protocol, noSuchChannel }
        LINKED {mtainfo}
        ::= 7

                                -- QMGR -> Console                   150
                                -- Incremental update
mtainfo OPERATION
        ARGUMENT PrioritisedMtaList
        ::= 12

                                -- control a mta
                                -- Console -> QMGR
mtacontrol OPERATION
```

```
        ARGUMENT MtaControl
        RESULT MtaInfo                                                  160
        ERRORS { congested, noSuchChannel, mtaNotInQueue,
                 authenticationFailure}
        ::= 8

                                        -- control a message
                                        -- Console -> QMGR
msgcontrol OPERATION
        ARGUMENT MsgControl
        RESULT NULL
        ERRORS { congested, protocol, noSuchChannel, mtaNotInQueue,     170
                 authenticationFailure}
        ::= 13

                                        -- QMGR -> Channel (process)
channelInitialise OPERATION
        ARGUMENT Channel
        RESULT NULL
        ERRORS { congested, protocol, noSuchChannel}
        ::= 9
                                                                        180
qmgrControl OPERATION
        ARGUMENT QMGRControl
        RESULT NULL -- always works
        ERRORS { congested, protocol, authenticationFailure}
        ::= 14

readChannelMtaMessage OPERATION
        ARGUMENT MsgRead
        RESULT MsgList
        ERRORS { congested, protocol, noSuchChannel, mtaNotInQueue }    190
        ::= 15

qmgrStatus OPERATION
        ARGUMENT NULL
        RESULT QmgrStatus
        ERRORS { congested, protocol }
        ::= 16
-- ERRORS

                        -- congestion at the responder                  200
congested ERROR
        ::=             0

                        -- unknown channel
noSuchChannel ERROR
        ::=             1

                        -- unknown operation
illegalOperation ERROR
        ::=             2                                               210

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

                        -- error in the protocol
protocol ERROR
```

```
        ::=             4

authenticationFailure ERROR                                          220
        ::=             5

-- Bind Arguments

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

WeakAuthentication ::= SEQUENCE {                                    230
        username [0] IA5String,
        passwd   [1] IA5String OPTIONAL
        }

BindResult ::= SEQUENCE {
                result[0] ENUMERATED {      -- result from the association request
                        acceptedLimitedAccess(0), -- accepted limited access
                        acceptedFullAccess(1)          -- accepted full access
                },
                information[1] IA5String OPTIONAL,                   240
                version[2] IA5String OPTIONAL
        }


BindError ::= SEQUENCE {
                reason[0] ENUMERATED {
                        badCredentials(0),          -- dont know this person
                        congested(1)                -- some system problem
                },
                information[1] IA5String OPTIONAL                    250
        }

-- DATA TYPES

Mta ::= IA5String

Channel ::= PrintableString

Priority ::= ENUMERATED {low(0), normal(1), high(2)}
                        -- Internal priorities COULD be different     260
                        -- to external ones

UAContentId ::= PrintableString

ContentType ::= PrintableString

EncodedInformationTypes ::= SEQUENCE OF PrintableString

QID ::= PrintableString
                                                                     270
User ::= IA5String
                        -- QMGR just sees users as strings.
                        -- This is 822 or /= X.400 syntax
                        -- The caller must get the right form!
```

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

MsgList   ::= SEQUENCE {
                    msgs SEQUENCE OF %[ type_Qmgr_MsgStructList %]
                                MsgStruct,
                    deleted SEQUENCE OF %[ type_Qmgr_QidList %] QID OPTIONAL
                    }

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
}

Control ::= CHOICE {
                    stop [0] NULL,
                    start [1] NULL,
                    cacheClear [2] NULL,
                    cacheAdd [3] UTCTime }

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

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

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

UserList ::= SEQUENCE OF RecipientId

RecipientId ::= INTEGER

MsgStruct ::= SEQUENCE {
          messageinfo       [0] PerMessageInfo,
          recipientlist     [1] SEQUENCE OF %[ type_Qmgr_RecipientList %]
                                    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,
        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,
        deferredTime        [11] UTCTime OPTIONAL,
        uaContentId         [12] UAContentId OPTIONAL,
        errorCount          [13] INTEGER OPTIONAL,
        inChannel           [14] Channel OPTIONAL
        }

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

RecipientInfo ::= SEQUENCE {
        user                [0] User,
        id                  [1] RecipientId,
        mta                 [2] Mta,
        channelList         [3] SEQUENCE OF %[ type_Qmgr_ChannelList %] Channel,
        channelsDone    [4] INTEGER,
                                -- Number of channels processed
                                -- 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,
        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,
        channel             [8] Channel OPTIONAL,
        mta                 [9] Mta OPTIONAL,
        queueid             [10] QID OPTIONAL,
        mpduiden            [11] MPDUIdentifier OPTIONAL,
        uaContentId         [12] UAContentId OPTIONAL }

ChannelReadResult ::= SEQUENCE {
```