```
        char            recip_reassign_prohibited;                      30
        char            recip_reassign_crit;

        char            dl_expansion_prohibited;
        char            dl_expansion_crit;

        char            conversion_with_loss_prohibited;
        char            conversion_with_loss_crit;

        char            *ua_id;             /* UA content id */
        char            *pp_content_correlator;                         40
                                            /* PP preferred string form  */
        struct qbuf     *general_content_correlator;
        char            content_correlator_crit;
                                            /* ASN.1 uninterpreted */

        FullName        *originator_return_address;
        char            originator_return_address_crit;

        int             forwarding_request;
        char            forwarding_request_crit;                        50

        struct qbuf     *originator_certificate;
        char            originator_certificate_crit;

        struct qbuf     *algorithm_identifier;
        char            algorithm_identifier_crit;
        /* content confidentiality */

        struct qbuf     *message_origin_auth_check;
                                /* also for Probe */                    60
        char            message_origin_auth_check_crit;

        struct qbuf     *security_label;
        char            security_label_crit;

        int             proof_of_submission_request;        /* boolean */
        char            proof_of_submission_crit;

        X400_Extension  *per_message_extensions;
                                                                        70
        ADDR            *Oaddress;          /* Orig addr - linked list */
        ADDR            *Raddress;          /* Recip addrs - linked list */

/* -- MTA AS Parameters */
        LIST_RCHAN      *inbound;           /* Inbound MTA/Channel */
        MPDUid          msgid;              /* Message ID */
        Trace           *trace;             /* Trace info */
        DLHistory       *dl_expansion_history;
        char            dl_expansion_history_crit;
                                                                        80
/* -- PP calculated parameters */
        EncodedIT       encodedinfo;        /* Encoded information types */
        UTC             queuetime;          /* Time that message was enqueued */
        UTC             departime;          /* Time that message left the queue */
        int             n_bodyparts;
        int             n_forwarded;

        /* -- Queue Control File offsets for fixed length variables -- */
```

Value	Meaning
CRITICAL_NONE	Not critical
CRITICAL_SUBMISSION	Critical for submission
CRITICAL_TRANSFER	Critical for Transfer
CRITICAL_DELIVERY	Critical for Delivery

Table 2.1: Criticality Codes

```
        off_t           nwarns_offset;      /* offset param for nwarns */
} Q_struct;                                                          90
```

Figure 2.4: Queue Structure

The queue structure controls most of the per message parameters. There are a number of utility routines to help manipulate these structures.

The routine `q_init` can be used initialise an empty structure. The routine `q_free` can be used to free off allocated fields within the structure.

```
int q_init (qp)
Q_struct *qp;

int q_free (qp)
Q_struct *qp;
```

Typically an inbound channel will only fill in one or two parameters, most can be safely defaulted unless required. It is important to set the `inbound` parameter and to fill in the list of encoded information types and possible the content type. The rest are all optional.

The elements of the queue structure are now described. Some elements have an associated flag with the name ending with `crit`. This refers to an element in the structure with a similar name. This element indicates what the criticality constraints on the element are. Values for the settings of these are shown in Table 2.1. These values may be logically or'd together to set several critical values.

`msgtype:` The type of message being submitted. This should be one of the following:

Value	Meaning
MT_UMPDU	Normal message
MT_DMPDU	A delivery report
MT_PMPDU	A Probe

The `q_init` routine defaults this value to `MT_UMPDU`.

`cont_type:` The content type of the message. This should be one of

the defined types; normally one of the following:

Value	Meaning
"822"	An RFC 822 content
"p2"	A P2 1984 content
"p22"	A P2 1988 content

As these values are common, they are available via the following string variables.

```
extern char *cont_822;
extern char *cont_p2;
extern char *cont_p22;
```

An empty content implies that the message will be submitted in structured form (usually the default for RFC 822 messages).

`encodedinfo:` A list of encoded information types. This should include all the encoded types that are in the message, and should be a member of the list defined in the bodypart tailor variable. Although any encoded information type present in the bodypart list is allowed, PP knows about some special cases. These are:

Value	Meaning
"ia5"	An Ia5 (ascii) body part
"hdr.822"	An RFC 822 header
"hdr.p2"	A P2 1984 header
"hdr.p22"	A P2 1988 header
"hdr.ipn"	An IPN header

These definitions are available in the external variables as follows:

```
extern char *hdr_822_bp;
extern char *hdr_p2_bp;
extern char *hdr_p22_bp;
extern char *hdr_ipn_bp;
extern char *ia5_bp;
```

The following variables can all be safely defaulted to those initialised by `q_init`, but can be varied as required.

`defertime:` The time the message should wait for before being delivered (for deferred delivery): optional.

`latest_time:` The latest delivery time this message should be delivered by.

`nwarns:` The number of warning messages to be sent if the message is delayed.

`warninterval:` The interval of time between successive warnings.

`retinterval:` The time to wait before returning the message as undeliverable.

`priority`: The priority associated with the message. This takes one of the following values:

Value	Meaning
`PRIO_NORMAL`	The default
`PRIO_NONURGENT`	A low priority message
`PRIO_URGENT`	A high priority message

`disclose_recips`: A true/false flag indicating whether recipients should be disclosed.

`implicit_conversion`: A true/false flag indicating if implicit conversion can be performed.

`alternate_recip_allowed`: A true/false flag indicating if the message can be sent to an alternate recipient.

`content_return_request`: A true/false flag indicating if the message content should be returned on delivery failure.

`recip_reassign_prohibited`: A flag indicating if the recipient may reassign the message to a new recipient.

`dl_expansion_prohibited`: A flag indicating if the message may not be expanded into a distribution list.

`conversion_with_loss_prohibited`: A flag indicating if conversion is allowed even if it loses information.

`ua_id`: The user agent ID. This is a string that identifies the message to a user agent.

`pp_content_correlator`: The internal version of the content correlator.

`general_content_correlator`: The content correlator of the message.

`originator_return_address`: The postal return address of the originator.

`forwarding_request`: An indication of whether the physical forwarding address should be returned to the originator.

`originator_certificate`: This element conveys the certificate of the originator when secuure messaging is being used.

`algorithm_identifier`: An indication of the encryption algorithm used for message encryption.

`message_origin_auth_check`: This element allows MTAs and recipients to authenticate the message origin.

`security_label`: A label indicating the security policy.

`proof_of_submission:` An indication of whether proof of submission is required.

`per_message_extensions:` A holder for any additional elements of the message.

`Oaddress:` Should be empty for submission.

`Raddress:` Should be empty for submission.

`inbound:` The inbound channel/mta pair This is important as this controls authentication and authorisation.

`msgid:` The MTS identifier of this message (P1 level).

`trace:` The trace fields.

`dl_expansion_history:` A trace of the distribution lists this message has been expanded by.

The Address structure

The address structure contains all the information about each address. The structure is defined in the file *adr.h*. The same structure is used for both Recipient and Originator address. The structure is shown in Figure 2.5. There are a number of support routines that help construct addresses.

The routine `adr_new` creates a new address structure with some of the values filled in. This structure is the minimum necessary set of values to submit a recipient. To create an Originator, the responsibility bit `ad_resp` should be set to false, and the status (`ad_stat`) should be set to `AD_STAT_DONE`. The routine is defined as:

```
ADDR *adr_new (str, type, rno)
char *str;
int type;
int rno;
```

The arguments to this call are

`str:` The address in string format, either RFC 822 or the string form of the O/R name.

`type:` This is the type of address; it should be one of the values suitable for `ad_type`.

`rno:` The recipient number. Set this to 0 if you do not care about this value.

The routine `ad_init` will initialise an `ADDR` structure to its default values. It is defined as:

```
void adr_init (ap)
ADDR *ap;
```

The routine **adr_add** adds an address onto the end of a linked list of addresses, or allocates the first node if the list is empty.

```
void adr_add (base, new)
ADDR **base;
ADDR *new;
```

The routine **adr_tfree** frees a list of addresses freeing of all allocated elements and the structures themselves.

```
void adr_tfree (ap)
ADDR *ap;
```

The routine **adr_free** frees off all the allocated elements within a single address structure but does not free the structure itself.

```
void adr_free (ap)
ADDR *ap;
```

```
typedef struct ad_redirection {
        struct ad_redirection *rd_next;
        char    *rd_addr;           /* O/R Name */
        char    *rd_dn;             /* Distinguished Name */
        UTC     rd_time;
        int     rd_reason;
} Redirection;

typedef struct      adr_struct {
        int                 ad_no;              /* recipient number */          10
                                                /* PP's key to this adr */

                                                /* Next 3 parms may be modfiied */
                                                /* in Q.  Fixed text encoding */
        int                 ad_status;          /* recipient status */
        int                 ad_rcnt;            /* reformatters done count  */

        /* -- MTS Service Parms */

        char                *ad_value;          /* address-original */          20
        char                *ad_dn;             /* Directory Distinguished Name */
                                                /* QUIPU string encoding */

        int                 ad_usrreq;          /* user-report-request bit */
        int                 ad_explicitconversion;      /* explicit conversion */
        int                 ad_type;            /* address-type */

        char                *ad_orig_req_alt; /* originator requested alternate */
                                                /* recipient - std encoded */
        char                ad_orig_req_alt_crit;                               30

#define AD_RDM_MAX      4
        int                 ad_req_del[AD_RDM_MAX];     /* requested delivery method */
        char                ad_req_del_crit;

        char                ad_phys_forward; /* boolean - is physical          */
                                                /* forwarding allowed */
```

```
char            ad_phys_forward_crit;

char            ad_phys_fw_ad_req;                                  40
char            ad_phys_fw_ad_crit;
                                /* boolean - request for phys */
                                /* foward address */

int             ad_phys_modes;
char            ad_phys_modes_crit;

int             ad_reg_mail_type;
char            ad_reg_mail_type_crit;
char            *ad_recip_number_for_advice;                        50
char            ad_recip_number_for_advice_crit;

OID             ad_phys_rendition_attribs;
char            ad_phys_rendition_attribs_crit;

int             ad_pd_report_request;
char            ad_pd_report_request_crit;
Redirection     *ad_redirection_history;
char            ad_redirection_history_crit;
                                                                    60
struct qbuf     *ad_message_token;
char            ad_message_token_crit;

struct qbuf     *ad_content_integrity;
char            ad_content_integrity_crit;

int             ad_proof_delivery;
char            ad_proof_delivery_crit;
                        /* boolean.  Is proof of delivery */
                        /* requested */                             70

/* -- MTA Service Params (calculated for MTS AS) */

int             ad_extension;       /* extension-id */
int             ad_resp;            /* responsibility bit */
int             ad_mtarreq;         /* mta-report-request bit */
int             ad_subtype;         /* address-subtype */


X400_Extension  *ad_per_recip_ext_list;                             80
                                /* bucket for new and private */
                                /* odds and sods */

/* -- parameters calculated by PP -- */
/* -- Note: the outbound mta is held in ad_outchan -- */

char            *ad_r400adr;        /* x400-addr */
char            *ad_r822adr;        /* rfc-addr */

/* -- should only be set for inbound x400 channels -- */               90
char            *ad_r400orig;       /* original form of x400 address */

/* -- next two should only be set for -- */
/* -- unroutable originator addresses -- */
char            *ad_r400DR;         /* x400 address for DRs */
char            *ad_r822DR;         /* 822 address for DRs */
```

```
        char            *ad_content;        /* outgoing content type */

        LIST_RCHAN      *ad_fmtchan;        /* reformatting channels */     100
        LIST_RCHAN      *ad_outchan;        /* outbound channel structure */
        LIST_BPT        *ad_eit;            /* outgoing eit's */


        /* -- from here onwards extra PP specific info -- */
        /* -- Diagnostics for any address parsing failures -- */

        int             ad_parse_stat;      /* address parsing err status */
        char            *ad_parse_message; /* address parsing err message */
        int             ad_reason;          /* DR reason failure */         110
        int             ad_diagnostic;      /* DR diagnostic code */
        char            *ad_add_info;       /* DR supplementary info */

        Aparse          *aparse;            /* parsing bucket */

        /* -- Address Control File offsets for fixed length variables -- */

        off_t           ad_no_offset;       /* offset param for recip number */
        off_t           ad_stat_offset; /* offset param for recip status */
        off_t           ad_rcnt_offset; /* offset param for reformat next */  120


        struct adr_struct   *ad_next;
} ADDR;

#define NULLADDR ((ADDR *)0)
```

Figure 2.5: Address Structure

Only the non-internal fields are described here:

`ad_extension:` The extension identifier for this address.

`ad_status:` The status of this address, one of:

Value	Meaning
AD_STAT_PEND	for recipients awaiting delivery
AD_STAT_DONE	for recipients not to be delivered to

`ad_dn:` The distinguished name component of the address.

`ad_usrreq:` This maps to the X.400 user request bits. It has the one of the following values:

Value	Meaning
AD_USR_NOREPORT	No delivery reports requested
AD_USR_BASIC	Delivery failutes should be reported
AD_USR_CONFIRM	Deliveries should be confirmed
AD_USR_NONE	Undefined value

`ad_explicitconversion:` Is explicit conversion required. It should be one of the values defined in Table 2.2.

Value
AD_EXP_NONE
AD_EXP_IA5_TEXT_TO_TELETEX
AD_EXP_TELETEX_TO_TELEX
AD_EXP_TELEX_TO_IA5_TEXT
AD_EXP_TELEX_TO_TELETEX
AD_EXP_TELEX_TO_G4_CLASS_1
AD_EXP_TELEX_TO_VIDEOTEX
AD_EXP_IA5_TEXT_TO_TELEX
AD_EXP_TELEX_TO_G3_FACSIMILE
AD_EXP_IA5_TEXT_TO_G3_FACSIMILE
AD_EXP_IA5_TEXT_TO_G4_CLASS_1
AD_EXP_IA5_TEXT_TO_VIDEOTEX
AD_EXP_TELETEX_TO_IA5_TEXT
AD_EXP_TELETEX_TO_G3_FACSIMILE
AD_EXP_TELETEX_TO_G4_CLASS_1
AD_EXP_TELETEX_TO_VIDEOTEX
AD_EXP_VIDEOTEX_TO_TELEX
AD_EXP_VIDEOTEX_TO_IA5_TEXT
AD_EXP_VIDEOTEX_TO_TELETEX

Table 2.2: Table of Explicit Conversion Values

`ad_type`: The address type. This is one of the following values:

Value	Meaning
AD_X400_TYPE	An X.400 format address
AD_822_TYPE	An RFC 822 format address
AD_ANY_TYPE	An address in any of the above formats

`ad_orig_req_alt`: The originator requested alternate recipient, if supplied.

`ad_req_del`: The requested delivery method. This is an array of delivery method alternatives.

`ad_phys_forward`: If physical forwarding is allowed.

`ad_phys_fw_ad_req`: A request to supply the physical forwarding address of the recipient.

`ad_phys_modes`: Specifies the type of delivery required.

`ad_reg_mail_type`: The type of registered mail in use.

`ad_recip_number_for_advice`: The number for advice to call for delivery.

`ad_phys_rendition_attribute`: The physical rendition attributes for the address when transfered to the postal service.

`ad_pd_report_request:` An option to supply a physical delivery report.

`ad_redirection_history:` The redirection history for this address.

`ad_message_token:` A security parameter for message tokens.

`ad_content_integrity:` A content integrity check parameter.

`ad_proof_delvery:` Proof of delivery is requested.

`ad_resp :` The responsibility bit. This should be set to either `TRUE` or `FALSE`.

`ad_mtareq:` This maps to the X.400 MTA report request bits. It should be set to one of the following:

Value	Meaning
`AD_MTA_NONE`	Undefined value
`AD_MTA_BASIC`	Generate Delivery failures
`AD_MTA_CONFIRM`	Generate Delivery confirmations
`AD_MTA_AUDIT_CONFIRM`	Generate full trace confirmations

`ad_subtype:` The subtype of the address if appropriate. This is only used for RFC 822 addresses at present. The values are:

Value	Meaning
`AD_NOSUBTYPE`	No particular sub type
`AD_JNT`	JANET style address
`AD_REAL733`	RFC 733 style address
`AD_REAL822`	RFC 822 style address

`ad_pre_recip_ext_list:` A list of extension attributes.

`ad_value:` This is the address submitted.

`ad_next:` The link to the next address if they are in a linked list.

2.2.2 Reply Value Structure

The reply value buffer structure is shown in Figure 2.6. This is used in all handshakes with *submit* to return a status and an explanatory string.

```
typedef struct rp_bufstruct     /* for reading reply strings */
{
   unsigned char      rp_val;
   char               rp_line[256];
} RP_Buf;
```

Figure 2.6: Reply Structure

The structure here is: