

# ***Mailing Protocols***

***Xerox System Integration Standard***

---

# Mailing Protocols

XEROX

---

## Notice

This *Xerox System Integration Standard* describes the Mailing Protocols.

1. This standard includes subject matter relating to patent(s) of Xerox Corporation. No license under such patent(s) is granted by implication, estoppel, or otherwise, as a result of publication of this specification.
2. This standard is furnished for informational purposes only. Xerox does not warrant or represent that this standard or any products made in conformance with it will work in the intended manner or be compatible with other products in a network system. Xerox does not assume any responsibility or liability for any errors or inaccuracies that this document may contain, nor have any liabilities or obligations for any damages, including but not limited to special, indirect, or consequential damages, arising out of or in connection with the use of this document in any way.
3. No representations or warranties are made that this specification, or anything made in accordance with it, is or will be free of any proprietary rights of third parties.

All text and graphics prepared on a Xerox 8010 Information System or a Xerox 6085 Professional Computer System.

This document is one of the family of publications that describe the network protocols underlying Xerox Network Systems.

Xerox Network Systems comprise a variety of digital processors interconnected by means of a variety of transmission media. System elements communicate both to transmit information between users and to economically share resources. For system elements to communicate with one another, certain standard protocols must be observed.

Comments and suggestions on this document and its use are encouraged. Please address communications to:

Xerox Corporation  
Xerox Systems Institute  
475 Oakmead Parkway  
Sunnyvale, California 94086



|                                                  |           |
|--------------------------------------------------|-----------|
| <b>1. Introduction</b>                           | <b>1</b>  |
| 1.1 Purpose                                      | 1         |
| 1.2 Relationship to other protocols              | 1         |
| 1.3 Documentation conventions                    | 2         |
| 1.3.1 Notation for Courier text                  | 2         |
| 1.3.2 Notation for Courier examples              | 2         |
| 1.4 Document organization                        | 3         |
| <b>2. Overview</b>                               | <b>5</b>  |
| 2.1 Clients and services                         | 5         |
| 2.2 Components of the Mailing Protocols Standard | 5         |
| 2.3 Mail system architectural model              | 6         |
| 2.4 Interpersonal messages                       | 8         |
| 2.5 MailTransport messages                       | 8         |
| 2.6 Transferring data                            | 8         |
| <b>3. MailTransport overview</b>                 | <b>9</b>  |
| 3.1 Posting and delivery via slots               | 9         |
| 3.2 Message structure: envelopes and contents    | 9         |
| 3.3 Posting of messages                          | 10        |
| 3.3.1 Locating a posting slot                    | 10        |
| 3.3.2 Posting the message                        | 11        |
| 3.3.3 Recipient validation                       | 11        |
| 3.4 Retrieval of messages                        | 12        |
| 3.4.1 Locating the delivery slot                 | 12        |
| 3.4.2 Polling for mail                           | 12        |
| 3.4.3 Retrieving a sequence of messages          | 12        |
| 3.5 Reports                                      | 13        |
| 3.5.1 Nondelivery reports                        | 13        |
| 3.5.2 Delivery reports                           | 13        |
| <b>4. MailTransport remote procedures</b>        | <b>15</b> |
| 4.1 Frequently used data types                   | 15        |
| 4.1.1 Authentication                             | 15        |
| 4.1.2 Naming of originators and recipients       | 16        |

|           |                                             |           |
|-----------|---------------------------------------------|-----------|
| 4.1.3     | Message contents                            | 17        |
| 4.1.4     | Envelopes                                   | 17        |
| 4.1.5     | Sessions                                    | 18        |
| 4.1.6     | Message IDs                                 | 18        |
| 4.2       | Posting of messages                         | 18        |
| 4.2.1     | Server poll                                 | 18        |
| 4.2.2     | Broadcast server poll                       | 19        |
| 4.2.3     | Types used during posting                   | 19        |
| 4.2.4     | Posting a message                           | 20        |
| 4.3       | Retrieval of messages                       | 21        |
| 4.3.1     | Types used during retrieval                 | 22        |
| 4.3.2     | Begin delivery                              | 22        |
| 4.3.3     | Retrieve envelope                           | 22        |
| 4.3.4     | Retrieve content                            | 23        |
| 4.3.5     | Abort retrieval                             | 23        |
| 4.3.6     | End retrieval                               | 23        |
| 4.4       | Checking for deliverable messages           | 24        |
| 4.4.1     | MailPoll                                    | 24        |
| 4.5       | Bulk Data formats                           | 24        |
| <b>5.</b> | <b>MailTransport remote errors</b>          | <b>25</b> |
| 5.1       | Authentication errors                       | 25        |
| 5.2       | Access errors                               | 26        |
| 5.3       | Invalid recipient error                     | 27        |
| 5.4       | Service errors                              | 27        |
| 5.5       | Transfer errors                             | 27        |
| 5.6       | Session errors                              | 28        |
| 5.7       | Other errors                                | 28        |
| <b>6.</b> | <b>MailTransport conformance subsets</b>    | <b>29</b> |
| 6.1       | Subset for delivery slot clients            | 29        |
| 6.2       | Subset for posting slot clients             | 29        |
| 6.3       | Subset for the service provider             | 29        |
| <b>7.</b> | <b>MailTransportEnvelopeFormat overview</b> | <b>31</b> |
| 7.1       | Envelope format                             | 31        |
| 7.2       | Relationship to X.400                       | 31        |
| 7.3       | Reports                                     | 32        |
| 7.4       | ContentsType                                | 32        |

|                                                           |           |
|-----------------------------------------------------------|-----------|
| <b>8. MailTransportEnvelopeFormat definition</b>          | <b>33</b> |
| 8.1 Common data types                                     | 33        |
| 8.2 Required envelope item types                          | 34        |
| 8.3 Other envelope item types                             | 35        |
| 8.4 Reports                                               | 37        |
| 8.4.1 Report envelopes                                    | 37        |
| 8.4.2 Report contents                                     | 37        |
| 8.4.3 The report envelope item                            | 37        |
| 8.5 Envelope item usage                                   | 40        |
| 8.6 Standard contents types                               | 42        |
| <b>9. MailTransportEnvelopeFormat conformance subsets</b> | <b>43</b> |
| 9.1 Subset for the simple client                          | 43        |
| 9.2 Subset for the X.400 client                           | 43        |
| 9.3 Requirements for the service provider                 | 43        |
| <b>10. MailFormat overview</b>                            | <b>45</b> |
| 10.1 Heading and body                                     | 45        |
| 10.2 Relationship to X.400                                | 45        |
| 10.3 Data encapsulation and transparency                  | 46        |
| <b>11. MailFormat definition</b>                          | <b>47</b> |
| 11.1 Interpersonal message format                         | 47        |
| 11.2 Common data types                                    | 48        |
| 11.3 Message heading attributes                           | 49        |
| 11.4 Other attributes                                     | 52        |
| 11.5 Message body part types for interpersonal messages   | 53        |
| <b>12. MailFormat conformance subsets</b>                 | <b>55</b> |
| 12.1 Subset for the simple client                         | 55        |
| 12.2 Subset for the X.400 client                          | 55        |
| <b>13. Inbasket overview</b>                              | <b>57</b> |
| 13.1 Inbaskets and delivery slots                         | 57        |
| 13.2 Message identification                               | 57        |
| 13.3 Message status                                       | 58        |
| 13.4 Body part status                                     | 58        |
| 13.5 MailTransport envelopes                              | 59        |
| 13.6 Sessions                                             | 59        |
| 13.7 Client good-citizenship considerations               | 59        |

|                                          |           |
|------------------------------------------|-----------|
| <b>14. Inbasket remote procedures</b>    | <b>61</b> |
| 14.1 Frequently used data types          | 61        |
| 14.1.1 Authentication                    | 61        |
| 14.1.2 Naming of inbaskets               | 62        |
| 14.1.3 Message indices and ranges        | 62        |
| 14.1.4 Inbasket anchor                   | 62        |
| 14.1.5 Inbasket state                    | 62        |
| 14.1.6 Inbasket sessions                 | 63        |
| 14.1.7 Message body parts                | 63        |
| 14.1.8 Status                            | 63        |
| 14.2 Initiating and terminating sessions | 64        |
| 14.2.1 Logon                             | 64        |
| 14.2.2 Logoff                            | 65        |
| 14.3 Checking inbasket state             | 65        |
| 14.3.1 Mail poll                         | 65        |
| 14.3.2 Mail check                        | 65        |
| 14.4 Examining inbasket contents         | 66        |
| 14.4.1 Retrieve envelopes                | 66        |
| 14.4.2 Retrieve body parts               | 66        |
| 14.5 Modifying inbasket contents         | 67        |
| 14.5.1 Change message status             | 67        |
| 14.5.2 Change body parts status          | 67        |
| 14.5.3 Delete                            | 68        |
| 14.6 Get size                            | 68        |
| 14.7 Bulk data formats                   | 68        |
| <b>15. Inbasket remote errors</b>        | <b>69</b> |
| 15.1 Authentication errors               | 69        |
| 15.2 Access errors                       | 70        |
| 15.3 Session errors                      | 70        |
| 15.4 Service errors                      | 71        |
| 15.5 Transfer errors                     | 71        |
| 15.6 Index errors                        | 71        |
| 15.7 Inbasket in use                     | 72        |
| 15.8 Other errors                        | 72        |
| <b>16. Inbasket conformance subsets</b>  | <b>73</b> |
| 16.1 Subsets for the client              | 73        |
| 16.2 Subset for the service provider     | 73        |

|                                           |           |
|-------------------------------------------|-----------|
| <b>17. Expedited procedure calls</b>      | <b>75</b> |
| 17.1 Expedited procedure call protocol    | 75        |
| 17.2 Packet formats                       | 76        |
| 17.2.1 Data in the Packet Exchange header | 76        |
| 17.2.2 Request packet                     | 76        |
| 17.2.3 Response packet                    | 77        |
| 17.3 When expediting fails                | 79        |

## **Appendices:**

|                                                         |     |
|---------------------------------------------------------|-----|
| A. References                                           | 81  |
| B. Type assignment procedures                           | 83  |
| C. MailTransport program declaration                    | 85  |
| D. MailTransportEnvelopeFormat program declaration      | 89  |
| E. MailFormat program declaration                       | 93  |
| F. Inbasket program declaration                         | 97  |
| G. Expanding ring broadcast                             | 101 |
| H. Locating a Mail Service for posting                  | 103 |
| I. Examples                                             | 107 |
| J. Relationship of the RName to foreign recipient names | 113 |



Computer-based message systems (generally referred to as "electronic mail systems") represent an important new communication medium. The most obvious use of such systems is interpersonal message communication among human users. More generally, electronic mail provides a reliable, buffered (non-real-time) data transport facility in a network system, suitable for communication among non-human agents (e.g., software systems) as well as for communication among human users.

A distributed mail system provides such communication through the cooperative interaction of an expandable set of *mail services*, working together to provide an integrated facility supporting the posting, transport and delivery of electronic mail. The mail services run on *server machines* and can be accessed by *client machines* using appropriate network protocols, which form the basis for the standard defined here. This standard provides for:

- posting of mail via "*posting slots*"
- delivery of mail via "*delivery slots*"
- medium-term storage of mail in mail service "*inbaskets*" for perusal and processing
- full data transparency, allowing the transport of arbitrary data from originator to recipient via the mail system
- a standard message format for interpersonal messages, facilitating a basic level of end-to-end compatibility among mail system clients

---

## 1.1 Purpose

---

This document defines the *Xerox Mailing Protocols Standard*, the set of protocols and related data formats for interaction between clients and the mail system. It does not describe any particular implementation of the protocols.

---

## 1.2 Relationship to other protocols

---

The protocols defined in this document are application-level protocols. They employ several other protocols. Requests to a mail service are communicated using the request-reply or transaction discipline defined by Courier [Courier]. Every type of request is modeled as a remote procedure as defined in Courier; every exceptional condition that may arise is modeled as a remote error. All parameters transferred between the client and the Mail service obey the conventions described in the Courier specification. This present specification, therefore, constitutes a set of Courier remote programs.

The contents of messages and other large data items are transferred using the protocol described in the Bulk Data Transfer Protocol addendum to Courier [Bulk Data].

---

## 1.3 Documentation conventions

---

Courier text and examples are depicted in special fonts, and generally conform to a certain style. The rules and style are set forth below.

---

### 1.3.1 Notation for Courier text

---

Throughout this document, special fonts are used to depict Courier text instead of using quote marks or other delimiters. This convention also aids the eye in discriminating between Courier text and the exposition. Items in **THIS FONT** indicate elements of the Courier language and are almost always in upper case. **This font** indicates items that are defined using the Courier language.

Identifiers that are defined in this protocol (as opposed to being defined by Courier) will have their first letter capitalized if they are the name of a type, error, or procedure; identifiers with a lower case first letter are usually the names of variables, arguments, or results.

---

### 1.3.2 Notation for Courier examples

---

In the examples that follow, a call to a remote procedure is denoted by the name of the procedure followed by the arguments supplied to it. A return from a remote procedure is denoted simply by the results, preceded – when confusion might otherwise result – by the keyword **RETURNS**. The argument or result list is modeled as a record; the arguments or results as the record's components. Accordingly, Courier's standard notation for record constants is used to specify argument and result lists.

For example, if the procedure **Add** is defined as:

**Add: PROCEDURE [first, second: CARDINAL] RETURNS [sum: CARDINAL] = 99;**

then a call to that procedure would be denoted by:

**Add [first: 7, second: 5]**

and the call would yield the result:

**[sum: 12] or RETURNS [sum: 12]**

Fine point: The above notation for procedure calls should not be confused with the standard notation for a record constant selected by means of a choice data type. The two are similar in appearance but otherwise unrelated.

Examples of remote errors are either just the name of the error, if it is defined without arguments:

### **Overflow**

or the same as a procedure call if it is defined with arguments. For example, if **Overflow** were defined as:

**Overflow: ERROR [carry: CARDINAL] = 99;**

then an example of its use might be:

**Overflow [carry: 1]**

indicating that **Overflow** was reported with argument **carry** having the value 1.

Courier requires values for a **SEQUENCE OF UNSPECIFIED** to be a sequence of numbers. So as to retain readability in examples, the content of a **SEQUENCE OF UNSPECIFIED** is described using Courier notation. The reader should understand that the numeric representation of these types is what should be used as the content of the sequence.

---

## **1.4 Document organization**

---

Section 2 of this document gives an overview of the concepts defined in the standard. The remaining sections of the document are grouped into four parts, corresponding to the four Courier programs that make up the Mailing Standard. Part I (Sections 3 - 6) describes the protocol for accessing the basic mail transport. Part II (Sections 7 - 9) defines the format of the information used by the basic mail transport to provide services such as delivery reports. Part III (Sections 10 - 12) defines the standard format of messages intended for interpersonal message communication. Part IV (Sections 13 - 16) defines the protocol that provides client access to inbaskets. Each of the four parts is divided into sections describing the main aspects of the corresponding Courier program, and concludes with a description of the options (if any) available to the client within the limits of compliance to the standard.

Appendix A lists other documents which supplement the specification. Appendix B describes the administrative procedures for obtaining ranges of values for each of the types of information carried transparently by the Mail System. Appendices C, D, E, and F list the MailTransport, MailTransportEnvelopeFormat, MailFormat and Inbasket remote programs in their entirety. Appendix G defines expanding ring broadcast, and Appendix H explains the use of this technique in locating a mail service for posting. Appendix I gives examples of interactions between a client and a mail service. Appendix J describes the relationship between names defined in the MailTransport protocol and names defined in the CCITT X.400 series of recommendations [X.400].



To understand the description of the Mail Service, an understanding of the following concepts and terms will be helpful.

---

## 2.1 Clients and services

---

This standard defines protocols for the communication of mailing requests. Requests to post a message, to accept delivery of a message, or to examine the contents of an inbasket are all examples of mailing requests.

A *service* is an entity (software or hardware or both) that accepts and responds to submitted requests for some type of service. A *Mail Service* is a service that handles mailing requests. The *Mail System* is composed of one or more Mail Services working cooperatively to provide an integrated facility. A *client* of the Mail System is an entity that submits requests to the services that make up that system. In this document, where a service is not otherwise specified, it is assumed to be a Mail Service. A client may or may not be operating on behalf of a human being.

All interaction between the client and the service is initiated by the client. A Mail Service never initiates activity with a client.

---

## 2.2 Components of the Mailing Protocols Standard

---

This standard is composed of four components, each in the form of a Courier program:

- MailTransport
- MailTransportEnvelopeFormat
- MailFormat
- Inbasket

To explain how these four components fit together, it is necessary to discuss the overall architecture of the Mail System.

## 2.3 Mail system architectural model

As a framework for discussing the four components of the Mailing Standard, it is useful to adopt an architectural model based on CCITT Recommendation X.400 [X.400]. In this model, the fundamental boundary is drawn between the User Agent Layer and the Message Transfer Layer. (In terms of the ISO Model of Open System Interconnection, these are sub-layers of the Application Layer. For brevity, they will be referred to below simply as layers.) The User Agent Layer includes all facilities for the original composition and posting of a message by its originator and for later delivery and processing of the message by its recipient(s). Actual transport of the message from the originator's user agent to that of the recipient is performed by the Message Transfer Layer. This layered structure and the fundamental interactions involved are depicted in Figure 2.1.

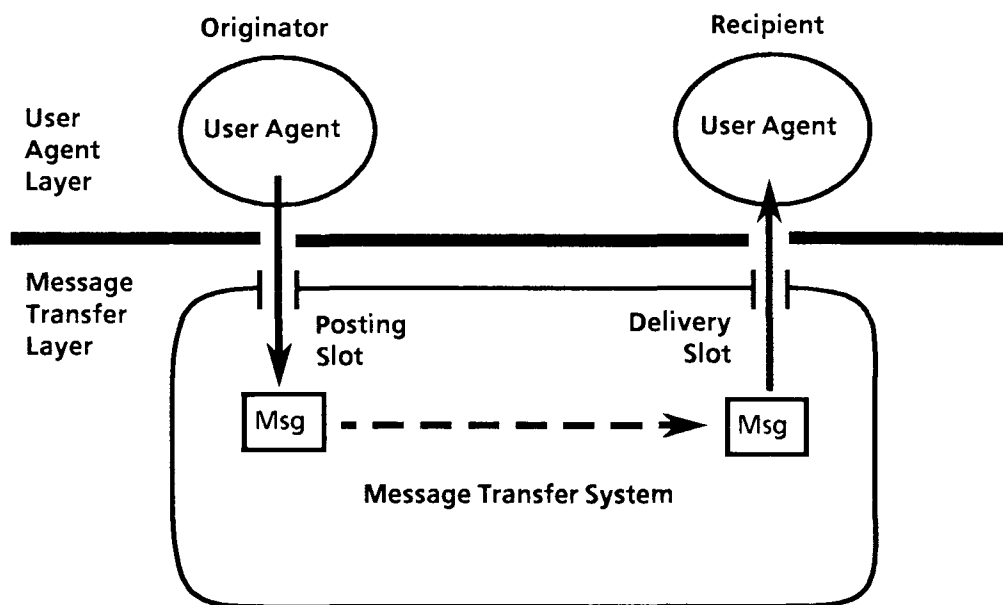


Figure 2.1: Basic X.400-based architecture

In terms of the basic model depicted in Figure 2.1, the MailTransport protocol defines the interaction between the Message Transfer Layer and the User Agent Layer, and provides operations for sending and receiving mail using the posting and delivery slots. Accessing the XNS Mail Service according to this protocol thus equates the physical boundary between the client and the service with the interaction between the two layers. Using the MailTransport protocol for delivery implies that mail is transferred sequentially (i.e., in a First In First Out manner) to the client machine (e.g., workstation).

The MailTransport protocol provides roughly the same functionality as X.411 of the CCITT X.400 series of recommendations [X.400]. However, in keeping with the XNS architectural principle that a service never spontaneously interacts with a client, the MailTransport protocol specifies that the passage of mail is always initiated by the client. This differs from the X.400 model in which the service is also allowed to "push" mail through to the client.

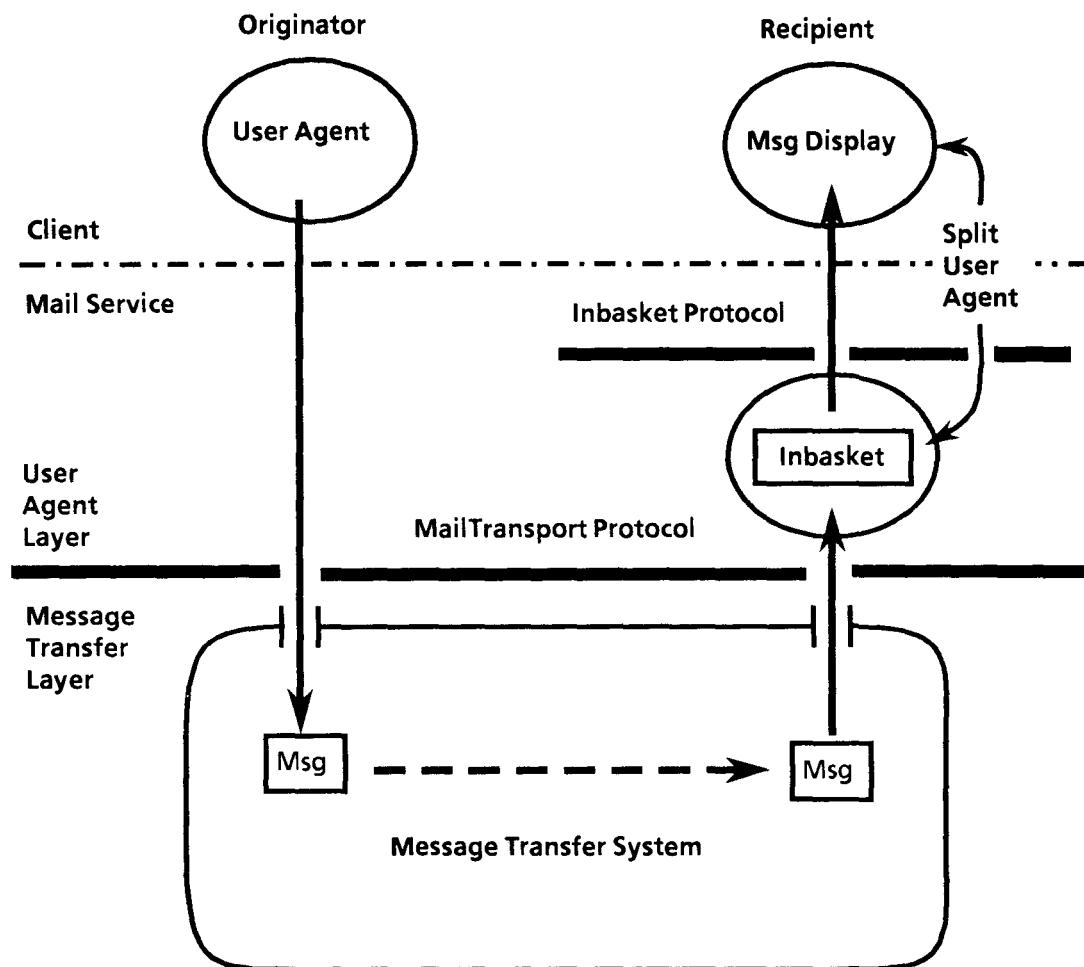


Figure 2.2: Extended architectural model

Sequential access to messages is fine for some clients. Users, however, often wish to gain access to their mail from any of several workstations. For example, suppose a message which contains text and graphics arrives while a user is reading mail from a workstation that cannot render the graphics. If the user has another workstation that can render the graphics, the ability to continue reading mail at the current workstation until it is convenient to move to the graphics workstation to retrieve the message containing graphics is useful. In order to allow this, the mail is retained on the Mail Service to make it globally accessible, and a second protocol called Inbasket is defined.

The Inbasket protocol provides random access to messages in a message store, or inbasket, which resides on the Mail Service. The Inbasket protocol corresponds architecturally to an internal interface of the User Agent layer. The extended architectural model is shown in Figure 2.2. In this extended model, the boundary between the client and the Mail Service is the MailTransport protocol for posting messages, and the Inbasket protocol for receiving them. The function that is architecturally assigned to the recipient's User Agent is thereby split between the Workstation and the Service as shown. The Inbasket protocol has no equivalent in the 1984 version of the CCITT X.400 series of recommendations [X.400].

---

## 2.4 Interpersonal messages

---

The Mail System defined by the MailTransport and Inbasket protocols provides a high degree of data transparency. While this serves to maximize its flexibility, it does not provide any standard for the format of messages exchanged among human users. This additional level of compatibility is provided by a third component of the Mailing Standard called MailFormat. Strictly speaking, MailFormat is not a third protocol, since it defines no additional operations to be invoked by Mail System clients. Instead, it defines a common format for interpersonal messages exchanged via the MailTransport and Inbasket protocols.

The MailFormat standard is similar to the CCITT X.420 standard for interpersonal messages [X.400]. In particular, the message contents are composed of a heading, which contains the descriptive information such as a list of the people the message is addressed to, and a body, which contains the actual substance of the message. The similarity to X.400 is discussed in more detail in Sections 10.2 and 11.3.

---

## 2.5 Mail Transport messages

---

All messages consist of a message envelope and the message contents. Standards such as MailFormat describe the message contents, but the Mail System defined by the MailTransport protocol does not provide any standard for the format of message envelopes exchanged among message transfer services. This additional level of compatibility is provided by a fourth component of the Mailing Standard called MailTransportEnvelopeFormat. Strictly speaking, MailTransportEnvelopeFormat is not a protocol, since it defines no additional operations to be invoked by Mail System clients. Instead, it defines a common format for envelopes exchanged via the MailTransport protocol.

The MailTransportEnvelopeFormat standard is roughly equivalent to the data type declarations in X.411 of the CCITT X.400 series of recommendations [X.400].

---

## 2.6 Transferring data

---

For those procedures that intrinsically require the transmission of a large amount of data, the Bulk Data Transfer Protocol [Bulk Data] is employed. Basically it works as follows: between the call to a remote procedure and the return from that procedure, the sender (either client or service) uses the bulk data transfer mechanism to send to the receiver the indicated data.

The MailTransport protocol is the fundamental protocol for submission and delivery of messages. This Courier program is exported by all Mail Services and used by mail clients for origination and receipt of mail.

---

### **3.1 Posting and delivery via slots**

---

The MailTransport protocol defines *posting slots* and *delivery slots* to support originators and recipients of messages, respectively. Using the posting and delivery operations described below, clients cause messages to flow through these slots. The passage of a message through a posting slot implies that the Mail System has accepted responsibility for the safe delivery of the message. Similarly, the passage of a message through a delivery slot implies that the receiving client has accepted responsibility for the message and that the Mail System has carried out its responsibility for delivering the message.

The Inbasket protocol is intended as the interface of choice for user agents supporting interpersonal messaging, and the delivery slot facility of the MailTransport protocol should not be used for this purpose. The delivery slot is best suited for situations in which the recipients of messages are computer processes rather than human beings.

---

### **3.2 Message structure: envelopes and contents**

---

In keeping with the X.400 architecture, the MailTransport protocol defines a message as an envelope and its associated contents. Unlike X.400, which only divides the contents of interpersonal messages into multiple components, the contents of all types of messages are divided into zero or more components, called message body parts. This structure is illustrated in Figure 3.1. The set of message body parts which may appear in a given type of message are described in the standard for that message type. For example, the set of message body parts which may appear in an interpersonal message are defined in the MailFormat standard.

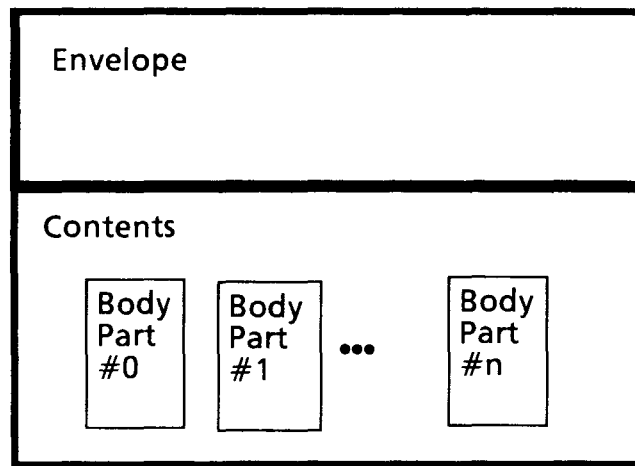


Figure 3.1 Envelope and Content

The envelope of a message names the recipients and otherwise describes the handling of the message as it is transported. It conveys information that is generally interpreted by the Mail System. This information can be communicated from the client to the Mail System during posting; communicated from the Mail System to the client during delivery; communicated by one Mail Service to another Mail Service during the transport of the message; or communicated from client to client (originator to recipient) without being interpreted by the Mail System.

The content of the message body parts is transparent data, and is not interpreted by the Mail System in any way. This transparency allows the MailTransport protocol to be used for transporting arbitrary data between clients.

---

## 3.3 Posting of messages

---

A message is posted by sending it to the server via a **BeginPost** operation followed by a **PostOneBodyPart** operation for each body part and terminated by an **EndPost** operation. The data used to build the envelope is passed via the parameters of the operation, and the content is sent as bulk data. Upon successful completion of the **EndPost** operation, the Mail Service assumes responsibility for the message. To maximize the likelihood of successful delivery, the Mail Service makes several checks before causing the **BeginPost** and **EndPost** operations to return successfully. For example, it determines whether the recipient names are either registered in the Clearinghouse [Clearinghouse] or reachable through a mail gateway, and ensures that the client has sent the correct number of body parts.

### 3.3.1 Locating a posting slot

---

Each Mail Service contains a single posting slot. Posting slots are anonymous, in the sense that mail addressed to any recipient can be posted at any slot. To post a message, a client locates a posting slot on some Mail Service and then invokes the posting operation of the

MailTransport protocol. The selection of a posting slot is composed of two steps: locating a set of available Mail Services, and choosing one of them to be used.

A client can locate one or more nearby Mail Services by broadcasting a query which mail servers respond to by indicating their degree of willingness to accept posted mail. This is described in detail in Appendix H.

Having received responses from a set of Mail Services, the client must decide on the criteria for selecting among them. The criteria should include such parameters as the Mail Service's willingness to accept mail, the size of the message being posted, the distance to the responding service, and the real-time delay to the responding service.

### 3.3.2 Posting the message

---

Once a posting slot has been selected, the client can submit messages to the Mail Service. Each message submission is accompanied by parameters which will be used to construct the message envelope. Some of these parameters indicate various optional choices on the part of the sender. These options influence the way in which the message is handled by the Mail System. For example, one of the optional parameters allows the client to prohibit distribution to groups of recipients.

#### 3.3.2.1 Sessions

---

Posting a message is done within the context of a session. A posting session begins with a call to **BeginPost** and ends with a call to **EndPost**. The session handle returned by **BeginPost** encapsulates state information about the session. Operations on the session handle should be performed sequentially; simultaneous calls on a single handle are not allowed. A session handle can become invalid at any time.

#### 3.3.2.2 Posting sequence

---

A typical posting sequence would consist of a call to **BeginPost**, multiple calls to **PostOneBodyPart**, and the concluding call to **EndPost**. An improper sequence of operations will result in an error. If the session times out after a call to **PostOneBodyPart** but before a call to **EndPost**, the Mail Service will discard the message.

### 3.3.3 Recipient validation

---

At posting time, the Clearinghouse is queried about each recipient name specified in the parameters to **BeginPost**. If any of the following conditions holds, then the name will be accepted: the recipient has a valid mailbox, the recipient is a group and the **allowDLRecipients** posting option was chosen, the message is destined for a valid gateway, or the necessary Clearinghouse information is unavailable. Otherwise the name is considered invalid. Acceptance of a recipient name does not mean that the Mail Service will be able to retrieve the message, because it is possible for a recipient name to become invalid between the time the Mail Service accepts the message and the time the message is actually processed. For example, the Mail Service could have accepted a message just before one of the recipients was deleted from the Clearinghouse.

## 3.4 Retrieval of messages

---

Retrieving messages via the delivery slot occurs in three stages. First, the delivery slot must be located. Then, the client periodically polls to determine whether new mail is available for delivery via the slot. Once mail has arrived at the slot retrieval, retrieval of the messages can be initiated, and typically continues until all new mail has been delivered. The client then reverts to polling until more new mail has arrived.

### 3.4.1 Locating the delivery slot

---

Unlike posting slots, delivery slots are not anonymous. Each delivery slot is associated with a specific recipient name. To retrieve mail via the slot, a client must locate the Mail Service on which the slot corresponding to its recipient name is located. This is done by looking up the recipient name in the Clearinghouse to determine the value of the **mailboxes** property associated with the name. The format of this property is defined in the Clearinghouse Entry Formats Standard [Clearinghouse Entries]. As defined there, the **mailboxes** property contains an array of Mail Service names. All slots listed should be considered as potential mail repositories and hence should be polled for mail.

### 3.4.2 Polling for mail

---

A delivery slot is polled for mail to find out if mail exists. No indication of the amount of existing mail is provided.

### 3.4.3 Retrieving a sequence of messages

---

Once the client has discovered that there is mail waiting at the delivery slot, the client can retrieve a sequence of messages.

#### 3.4.3.1 Sessions

---

Mail retrieval is done within the context of a session. A delivery slot session begins with a call to **BeginRetrieval** and ends with a call to **EndRetrieval**. The session handle returned by **BeginRetrieval** encapsulates state information about the session. Operations on the session handle should be performed sequentially; simultaneous calls on a single handle are not allowed. A session handle can become invalid at any time.

#### 3.4.3.2 Retrieval sequence

---

A typical sequence would consist of a call to **BeginRetrieval**, multiple calls to the sequence **RetrieveEnvelope**, **RetrieveContent**, and the concluding call to **EndRetrieval**. When a call to **RetrieveEnvelope** is immediately followed by a second call to **RetrieveEnvelope**, the contents of the first message are discarded. This might be done, for instance, when there is not enough space available to retrieve the message - but this is not the normal case. An improper sequence of operations will result in an error. Messages in the delivery slot are delivered in a FIFO (First In, First Out) manner. The message at the head of the slot must be disposed of before any succeeding messages can be accessed. If the session times out after a call to **RetrieveEnvelope** or **RetrieveContent** but before a call to **EndRetrieval**, the Mail

Service will retain responsibility for the message, and it will be at the head of the delivery slot when the next session is established.

---

## 3.5 Reports

---

A report is a type of message generated by the Mail System to indicate the fate of a previous message. For each of the recipients of a message, the client can request a nondelivery report, a delivery report, or no report at all. In addition, the client can request that the contents of a message be returned along with any nondelivery report. The format of the envelope and message body parts of a report is defined in the MailTransportEnvelopeFormat protocol (see Sections 7.3 and 8.4).

### 3.5.1 Nondelivery reports

---

Occasionally, the Mail System is unable to deliver a message that was successfully posted. If this occurs, a nondelivery report is generated, unless the client has requested that reports be suppressed, as described in Section 3.5.3.

#### 3.5.1.1 Causes of nondelivery

---

There are several reasons that a message might be undeliverable. The recipient name may have become invalid between posting and delivery time. Clearinghouse information necessary for delivery may not have become available within a given period of time. The message may not have been forwarded to the destination Mail Service within a certain period of time due to lack of service availability or network connectivity problems. As a final example, the destination Mail Service database may not have had enough room to accept the message.

#### 3.5.1.2 Recipients of nondelivery reports

---

Nondelivery reports are usually sent to the originator of the undeliverable message. However, if the problem recipient was a member of a distribution list, and the problem is **noSuchRecipient**, **noMailboxForRecipient**, or **illegalName**, a message will be sent to an appropriate entity to indicate that the distribution list needs to be modified. The appropriate entity is determined by implementation and installation.

#### 3.5.1.3 Prevention of nondelivery reports

---

If the client does not care to know the fate of a message, suppression of reports can be requested. When reports are suppressed, the client will not receive any delivery or nondelivery reports for the message. This is particularly useful when messages are sent to large distribution lists.

### 3.5.2 Delivery reports

---

If the client wants acknowledgment that a message has been delivered to an individual recipient, a delivery report can be requested. The Mail System generates the report when the message has been added to the messages in the delivery slot or inbasket of the recipient. The

time of delivery is included in the report. Delivery reports may not be requested for distribution lists. Note that the delivery report is triggered by the message being delivered to the delivery slot or inbasket, not by the retrieval of the message by the client.

Each Mail Service implements the set of MailTransport operations as a Courier [Courier] remote program. Each procedure description includes a declaration of the procedure in Courier's standard notation, a description of the procedure's arguments and results, and occasionally an example of its use. Several types and constants defined in the MailTransport program are also used in the MailFormat, MailTransportEnvelopeFormat and Inbasket protocols described in later sections of this document.

The following definition gives the program and version numbers of the MailTransport protocol and lists all other Courier-based protocols which are referenced from this program. A complete listing of the MailTransport Courier program is given in Appendix C.

```
MailTransport: PROGRAM 17 VERSION 5 =
BEGIN
    DEPENDS UPON
        Authentication (14) VERSION 2,
        BulkData (0) VERSION 1,
        Clearinghouse (2) VERSION 3;
    ...
END.
```

This indicates that MailTransport is program number 17. This document defines version 5. MailTransport references some types and constants that are defined in other protocols as shown in the above declaration. These protocols are documented in the Authentication Protocol [Authentication], the Bulk Data Transfer Protocol [BulkData], and the Clearinghouse Protocol [Clearinghouse].

---

## 4.1 Frequently used data types

---

This section discusses data types that are used throughout the Mailing protocol.

### 4.1.1 Authentication

---

Many MailTransport remote procedures require the client to provide data that unequivocally demonstrates the client's right to access the service. The qualifying information is composed of two items, *credentials* and a *verifier*, as defined in the Authentication Protocol.

**Credentials:** TYPE = Authentication.Credentials;  
**Verifier:** TYPE = Authentication.Verifier;

Together, these two parameters identify the client to the service and contain enough information to validate the client's right to perform the requested operation. The structure of the two items, **Credentials** and **Verifier**, is defined in [Authentication]. The values of the

**Verifier** are defined in [Authentication]. The **Credentials** need not be *strong*, and the names used in them should be those of the user and the Mail Service. In some of the delivery operations, the credentials are not passed every time, because they are validated at the time the session is established, and are thereafter implicit in the **Session** established for a sequence of operations.

#### 4.1.2 Naming of originators and recipients

---

Correspondents (originators and recipients of messages) are identified by their **RNames** (Recipient Names). Correspondents within the XNS Naming System are identified by their names as registered in the Clearinghouse. For those correspondents outside of the XNS Naming System who are reached via a gateway, multipart names are defined. These multipart names are not registered in the Clearinghouse.

```
RName: TYPE = CHOICE OF {  
  xns(0) = > Name,  
  gateway(1) = > RECORD [  
    country: STRING,  
    adminDomain: STRING,  
    privateDomain: STRING,  
    organization: STRING,  
    organizationalUnits: SEQUENCE 20 OF STRING,  
    personal: CHOICE OF {  
      whole(0) = > STRING,  
      broken(1) = > RECORD [given, initials, family, generation: STRING]}  
    gatewaySpecificInformation: SEQUENCE 10 OF Attribute];
```

```
Attribute: TYPE = RECORD [type, value: STRING];
```

The casing of characters is not significant when two **RNames** are compared for equality.

The relationship between an **RName** and the **ORName** defined in the CCITT X.400 series of recommendations is given in Appendix J, which also gives the characters allowed in each of the **ORName** fields. These restrictions do not apply to the **RName** unless the **RName** denotes a recipient in the X.400 Mail System.

The maximum length of the strings in a **gateway** variant of an **RName** are given by the X.400 Implementors Guide and are as follows:

- country**: maximum length of 3 bytes
- adminDomain**: maximum length of 16 bytes
- privateDomain**: maximum length of 16 bytes
- organization**: maximum length of 64 bytes
- organizationUnits**: maximum length of each element is 32 bytes
- personal whole**: maximum length of 40 bytes
- personal given**: maximum length of 16 bytes
- personal initials**: maximum length of 5 bytes
- personal family**: maximum length of 40 bytes
- personal generation**: maximum length of 3 bytes

The **gatewaySpecificInformation** is used for name fields specific to foreign mailing protocols. Several values for **Attribute** have been defined. These are listed in Appendix B along with the procedure for obtaining new values.

In places where a name is required to be a proper Clearinghouse name, this is indicated explicitly:

Name: TYPE = Clearinghouse.Name;

---

### 4.1.3 Message contents

---

#### 4.1.3.1 Contents types

---

As noted in the MailTransport overview (see Section 3.2), the contents type distinguishes one message format from another. Interpersonal messages are designated by one **ContentsType**, delivery and nondelivery reports are designated by another, and the Clearinghouse uses yet another contents type to designate the messages which keep its distributed database consistent. Several values for **ContentsType** have been defined, and these are listed in Section 8.6. The procedure for assigning new contents types is given in Appendix B.

ContentsType: LONG CARDINAL;

#### 4.1.3.2 Message body part types

---

The contents of a message are divided into one or more components which are called message body parts. A range of 3777777777B **BodyPartTypes** is allocated for each value of **ContentsType**. The **BodyPartTypes** used for the interpersonal message contents type are described in the MailFormat protocol, and the **BodyPartTypes** used for the report **ContentsType** are described in the MailTransportEnvelopeFormat protocol. The procedure for allocating new **BodyPartTypes** for an existing **ContentsType** is given in Appendix B.

BodyPartType: TYPE = LONG CARDINAL;

---

### 4.1.4 Envelopes

---

The envelope of a message is composed of a sequence of components called envelope items. An envelope item has two parts: its *type field* and its *value field*. The *type field* contains a 32-bit numeric tag. The *value field* is an uninterpreted block of data encoded as a **SEQUENCE OF UNSPECIFIED**, where the interpretation of the data (i.e., the **UNSPECIFIED**) as specific courier data type depends on the contents of the *type field*. Thus, the definition of a new envelope item consists of an assigned code for its *type field* (a 32-bit constant), and a specification of the Courier data type to be used in its *value field*. The procedure for allocating new envelope items is given in Appendix B. The Courier datatype **EnvelopeItem** represents this generic data structure:

Envelope: TYPE = SEQUENCE 100 OF EnvelopeItem;

EnvelopeItem: TYPE = RECORD [type: EnvelopeItemType, value: SEQUENCE OF UNSPECIFIED];

EnvelopeItemType: TYPE = LONG CARDINAL;

The various specific kinds of envelope items (i.e., specific values for the **type** field) are described in the MailTransportEnvelopeFormat protocol.

#### 4.1.5 Sessions

---

A **Session** is used during both posting and delivery to identify a sequence of operations.

**Session:** TYPE = RECORD [token: ARRAY 2 OF UNSPECIFIED, verifier: Verifier];

The first field is an identifying bit string whose content is unspecified. It should be passed unchanged by the client from one operation to the next. The second field verifies the identity of the client and should, therefore, be recomputed on each call in exactly the same way as on any other call which passes credentials and verifiers.

#### 4.1.6 MessageIDs

---

When a message is successfully posted, a unique ID for the message is returned to the client. This ID is also a part of the message envelope. It is used by both the client and the Mail System to identify the message.

**MessageID:** TYPE = ARRAY 5 OF UNSPECIFIED;

---

### 4.2 Posting of messages

---

As summarized previously, the posting of a message consists of location of a posting slot on an appropriate Mail Service and transmission of the message to that service. The former is ordinarily done using the **ServerPoll** operation, while the latter uses the posting operations.

#### 4.2.1 Server poll

---

The **ServerPoll** operation can be broadcast (see Section 4.2.2) to aid clients of the message transfer system in locating a Mail Service to use for posting messages. Appendix H describes when the broadcast should be done, and gives guidelines for choosing a Mail Service from the responses to the broadcast. Other methods of finding a Mail Service, such as always posting to the Mail Service that holds the delivery slot or inbasket of the originator, could be used. Such methods are not recommended because they make it difficult for the Mail System to balance its workload. When it is not broadcast, the **ServerPoll** operation can be used to check that the Mail Service currently being used for posting messages is still willing to receive them.

**Willingness:** TYPE = SEQUENCE 12 OF WillingnessMetric;

**WillingnessMetric:** TYPE = CARDINAL;

**ServerPoll:** PROCEDURE []

RETURNS [willingness: Willingness, address: Clearinghouse.NetworkAddressList,  
serverName: Name] = 0;

Results: **willingness** is an indication of the availability of the particular Mail Service at the time that the **ServerPoll** was performed. The value for **willingness** is a sequence of **CARDINALS**, each of which can have values from 1 (least willing) to 10 (most willing), which indicates the

extent to which the Mail Service should be considered available to accept posted mail. **Willingness[i]** gives the service's ability to accept a message of size  $8^i$  bytes to size  $8^{(i+1)} - 1$  bytes. The last element in the sequence gives the service's ability to accept a message of size  $8^{(\text{index of the last element in the sequence})}$  bytes to a message of unbounded size. **address** is the list of network addresses that can be used to contact the responding Mail Service. Note: the address list will contain more than one element only if the Mail Service is connected to more than one network. **serverName** is the three-part Clearinghouse name of the responding Mail Service.

#### 4.2.2 Broadcast server poll

---

There is a MailTransport operation that is not part of the MailTransport Courier program. This operation, called **BroadcastServerPoll**, can be used to locate Mail Services on a given network. It is invoked using the Packet Exchange Protocol. A directed broadcast is made on the designated network and each Mail Service on that network responds with its network addresses and other information describing itself and its state (See Appendix G for more details on directed broadcast). This operation, although outside the Courier program proper, is bit-for-bit identical to an expedited call to the normal **ServerPoll** operation, except that it is broadcast rather than being sent to a specific server address. See Section 17 for a description of the packet formats used.

#### 4.2.3 Types used during posting

---

A **Session** used for posting is an identifier for the sequence of operations used to post a message. The **Session** is defined in Section 4.1.5.

Clients of MailTransport can request delivery reports or prevent nondelivery reports on a per recipient basis. This is accomplished by identifying a recipient by more than just a name.

**ReportType:** TYPE = {none(0), nonDeliveryOnly(1), all(2)};

**Recipient:** TYPE = RECORD [name: RName, recipient ID: CARDINAL, report: ReportType];

**RecipientList:** TYPE = SEQUENCE OF Recipient;

The **id** indicates the position of the recipient in the **RecipientList**. The value for **id** is 1 for the first recipient, 2 for the second recipient, and so on. If a **ReportType** of **none** is selected, both delivery and nondelivery reports will be suppressed by the message transfer service. If **nonDelivery** is selected, the message transfer service will generate a nondelivery report if the message cannot be delivered. If **all** is selected, the message transfer service will generate a delivery report when the message is delivered, or a nondelivery report if the message cannot be delivered.

The information the Mail System will use to build the envelope of a message, direct it to its destination, and determine which reports to generate is in the **PostingData**. The **PostingData** contains the list of recipients the message will go to, the contents type, the size of the message contents, and a list of the body part types the message is composed of. There should be one element in the **BodyPartTypesSequence** for each body part in the message. The **ContentsSize** specifies the size of the message content in bytes. The size should be within the range [actual size - 5000 bytes .. actual size] or the Mail Service will raise an error. Note that the size of the envelope is not included in the **ContentsSize**.

```
PostingData: TYPE = RECORD [  
    recipients: RecipientList,  
    contentsType: ContentsType,  
    contentsSize: LONG CARDINAL,  
    bodyPartTypesSequence: SEQUENCE 500 OF BodyPartType];
```

When a message is posted, the **BeginPost** operation may immediately return an indication that one or more recipients are invalid. An **id** giving the position of the invalid name in the recipient list and the reason the name is invalid are returned for each invalid name. This information is returned in the form of an invalid name list.

```
InvalidName: TYPE = RECORD [id: CARDINAL, invalidReason: InvalidReason];  
InvalidNameList: TYPE = SEQUENCE OF InvalidName;  
InvalidReason: TYPE = {  
    noSuchRecipient(0), noMailboxForRecipient(1), illegalName(2), noDLsAllowed(3),  
    reportNotAllowed(4)};
```

The reasons that names are invalid are interpreted as follows:

**noSuchRecipient:** —the name doesn't exist.

**noMailboxForRecipient:** —the name exists, but there is no mailbox registered for the recipient.

**illegalName:** —the name does not have the syntax of an **RName**, or there are wildcards in any of the three parts of an XNS **RName**.

**noDLsAllowed:** —the recipient is a distribution list, and distribution lists were not allowed to appear in the recipient list when the message was posted.

**reportNotAllowed:** —the recipient is a distribution list, and the client requested a delivery report for the distribution list (delivery reports are not supported for distribution lists).

#### 4.2.4 Posting a message

---

Once an appropriate Mail Service has been located, outgoing messages can be posted using the **BeginPost** operation followed by a sequence of **PostOneBodyPart** operations, and terminated by the **EndPost** operation.

```
BeginPost: PROCEDURE [postingData: PostingData, postIfInvalidNames: BOOLEAN,  
    allowDLRecipients: BOOLEAN, optionalPostingData: SEQUENCE OF EnvelopeItem,  
    credentials: Credentials, verifier: Verifier]  
RETURNS [session: Session, invalidNames: InvalidNameList]  
REPORTS [AuthenticationError, InvalidRecipients, OtherError, ServiceError] = 1;
```

Arguments: **postingData** is described in Section 4.2.3. **postIfInvalidNames** controls the behavior of the operation when not all recipient names are valid: **TRUE** causes the message to be sent to those names which are valid (the invalid names will be returned as a result of the procedure), while **FALSE** causes the message not to be sent at all (and the invalid names will be signaled with an error). **allowDLRecipients** controls whether or not the message is sent to distribution lists. A value of **TRUE** causes the message to be sent even if one or more recipients are distribution lists, while **FALSE** causes any recipients which are distribution lists to be treated as invalid recipients. **allowDLRecipients** can be used, for example, by the user agent to prevent messages from being sent to distribution lists when there is no name that

recipients can direct replies to. **optionalPostingData** allows the setting of optional envelope items (see Section 8.3 for a list of these items). If a non-optional item is specified in **optionalPostingData**, an error will be raised. **credentials** and **verifier** are described in Section 4.1.1. The client cannot switch from strong to simple authentication or vice versa within a session.

**Results:** **session** is the identifier for operations used to post this message. **invalidNames** conveys a list of invalid recipients in the case that not all recipients were valid but the message was posted anyway. (This can only occur if **postIfInvalidNames** = **TRUE**.) Note that the **session** is invalid if **postIfInvalidNames** is **FALSE** and the recipient list contains invalid names. An error will be raised if such a session handle is passed to the **PostOneBodyPart** procedure.

**PostOneBodyPart:** PROCEDURE [session: Session, bodyPartType: BodyPartType,  
contents: BulkData.Source]  
REPORTS[AuthenticationError, OtherError, SessionError, ServiceError,  
TransferError] = 8;

**Arguments:** **session** is the identifier for operations used to post this message. **bodyPartType** specifies the type of this body part. A **SEQUENCE** of **BodyPartTypes** is part of the **postingData** passed to the **BeginPost** call. The **bodyPartType** passed to **PostOneBodyPart** is used to check that all body parts that were supposed to be included in the message are sent to the Mail Service. If body parts not specified in the **postingData** are received or if more than 5 minutes elapse between calls to **PostOneBodyPart**, an error is raised.

**EndPost:** PROCEDURE [session:Session, abortPost: BOOLEAN]  
RETURNS[messageID: MessageID]  
REPORTS[AuthenticationError, OtherError, SessionError, ServiceError, TransferError] = 9;

**Arguments:** **session** is the identifier for operations used to post this message. After **EndPost** returns, **session** is no longer valid. If **abortPost** = **TRUE**, the message will be discarded by the Mail Service.

**Results:** **messageID** is the unique message identifier assigned by the Mail Service during posting.

---

## 4.3 Retrieval of messages

---

Retrieving a message differs from posting a message in that the location of the slot is more clearly defined: the delivery slot for the desired recipient is located on some well-defined Mail Service and it is that service to which the appropriate Courier calls must be directed.

Note that retrieval of mail via the MailTransport delivery-slot paradigm is intended primarily for messages whose recipients are autonomous machine entities, rather than human users. The Inbasket protocol is intended as the preferred vehicle for retrieving interpersonal messages whose recipients are human users (see Sections 13-16).

### 4.3.1 Types used during retrieval

---

The only data type commonly used when retrieving messages is the **Session**. A **Session** used for retrieval is an identifier for the sequence of operations used to retrieve one or more messages via a given delivery slot. At any time, only one client can be accessing a given delivery slot, hence only one **Session** can be outstanding for each slot. Any attempt to obtain a second **Session** for a slot that is already active will result in an error. The **Session** is defined in Section 4.1.5. Simultaneous calls on a single session handle will also result in an error.

### 4.3.2 Begin retrieval

---

**BeginRetrieval** initiates retrieval of a series of (one or more) messages via a specified delivery slot. The entire multi-message retrieval sequence is bracketed by calls to **BeginRetrieval** and **EndRetrieval**. During the sequence, the active delivery slot is identified by the **session** returned by **BeginRetrieval**. This **session** is passed to each subsequent call in the sequence (with an updated verifier enclosed).

**BeginRetrieval**: PROCEDURE [deliverySlot: Name, credentials: Credentials, verifier: Verifier]

RETURNS [session: Session]

REPORTS [AccessError, AuthenticationError, OtherError, ServiceError] = 2;

Arguments: **credentials** and **verifier** are described in 4.1.1. The client cannot switch from strong to simple authentication or vice versa within a session. **deliverySlot** names the recipient whose delivery slot is to be accessed by subsequent calls using the returned **session**. The delivery slot name need not be the same as the name of the entity accessing the slot.

Results: **session** identifies the active delivery slot, and is used by subsequent calls until **EndDelivery** is called, at which time the session handle becomes invalid.

### 4.3.3 Retrieve envelope

---

**RetrieveEnvelope** starts the retrieval of the next message by initiating transfer of its envelope through the delivery slot. **RetrieveEnvelope** is legal following **BeginRetrieval**, **RetrieveContent**, **AbortRetrieval** or another call to **RetrieveEnvelope**; at any other time, its use is a protocol violation. When a call to **RetrieveEnvelope** is immediately followed by a second call to **RetrieveEnvelope**, the contents of the first message are discarded (e.g., if the envelope showed that the contents were uninteresting)

**RetrieveEnvelope**: PROCEDURE [session: Session]

RETURNS [empty: BOOLEAN, envelope: Envelope]

REPORTS [AuthenticationError, OtherError, SessionError, ServiceError] = 5;

Arguments: **session** identifies the active delivery slot through which the envelope will be retrieved.

Results: **empty** indicates the presence (if **FALSE**) or absence (if **TRUE**) of messages available via the active delivery slot. When messages are available, **envelope** is the message envelope being retrieved; if **empty** = **TRUE**, then **envelope** contains no envelope items.

#### 4.3.4 Retrieve content

---

**RetrieveContent** retrieves the content of the message whose envelope was retrieved by the immediately preceding **RetrieveEnvelope** operation. **RetrieveContent** is legal only immediately following a **RetrieveEnvelope** operation; at any other time, it is a protocol violation.

**RetrieveContent:** PROCEDURE [content: BulkData.Sink, session: Session]  
REPORTS [AuthenticationError, OtherError, SessionError, ServiceError,  
TransferError] = 4;

Arguments: **session** identifies the active delivery slot through which the content will be retrieved; **content** identifies the bulk data stream through which the content will flow during retrieval.

#### 4.3.5 Abort Retrieval

---

In some situations, the client would like the Mail Service to retain the message until the client can accept it. This is done using **AbortRetrieval** to abort the retrieval of the message.

An **AbortRetrieval** will typically be followed by a call to **EndRetrieval**. Note that following an **AbortRetrieval**, any subsequent messages at the slot will be inaccessible until the message in question is disposed of (e.g. by some other client machine better able to deal with it).

**AbortRetrieval:** PROCEDURE [session: Session]  
REPORTS [AuthenticationError, OtherError, SessionError, ServiceError, TransferError] = 3;

Arguments: **session** identifies the active delivery slot.

**AbortRetrieval** is legal following either **RetrieveContent** or **RetrieveEnvelope**; at any other time, it is a protocol violation and an error is raised. **AbortRetrieval** would be called after **RetrieveContent** if the client discovers, after the bulk data transfer has completed, that it cannot store the message.

#### 4.3.6 End retrieval

---

A retrieval sequence is terminated by calling **EndRetrieval**. Typically (although not necessarily) this will be done when there are no further messages to be delivered (i.e. when **RetrieveEnvelope** has returned **empty = TRUE**). **EndRetrieval** may be used following any other delivery operation to terminate the active retrieval sequence. If **EndRetrieval** is called before any other retrieval operation, an error is raised.

**EndRetrieval:** PROCEDURE [session: Session]  
REPORTS [AuthenticationError, OtherError, SessionError, ServiceError, TransferError] = 6;

Arguments: **session** identifies the active delivery slot. After **EndDelivery** returns, **session** is no longer valid.

## 4.4 Checking for deliverable messages

---

Periodically, the Mail System client will wish to check for the arrival of mail at the delivery slot. While this check could be made by simply attempting the normal retrieval sequence (i.e., **BeginRetrieval** followed by **RetrieveEnvelope**), the overhead of doing so would be exorbitant in the case where no messages had arrived. Since frequent checking for mail is encouraged, the delivery slot will probably be empty, and a special polling operation is therefore provided to check for the presence of messages at the delivery slot.

### 4.4.1 MailPoll

---

**MailPoll** checks for messages available via the specified delivery slot.

**MailPoll:** PROCEDURE [**deliverySlot:** Name, **credentials:** Credentials, **verifier:** Verifier]

RETURNS [**mailPresent:** BOOLEAN]

REPORTS [**AccessError**, **AuthenticationError**, **OtherError**, **ServiceError**] = 7;

Arguments: **credentials** and **verifier** are described in 4.1.1. **deliverySlot** identifies the recipient whose delivery slot is being polled for messages.

Results: **mailPresent** = TRUE indicates that there is mail at the delivery slot ready for retrieval.

**NOTE:** The **MailPoll** operation can be expedited (see Section 17).

---

## 4.5 Bulk Data formats

---

Bulk data is used as defined in the Addenda to Standards, XSI 138301. That is, the transfer consists of a series of packets, the last of which has end-of-message set to TRUE (if the transfer was successful). No additional structuring of the data is needed.

When a remote procedure completes successfully, it returns results as specified in the definition of the procedure. However, conditions can arise before or during execution of the procedure that make successful completion of the request impossible. For example, the client may have specified incorrect arguments in a remote procedure call, or some required resource may be unavailable.

When such conditions occur, an error is reported to the client to communicate the nature of the problem. Each error encompasses an entire class of possible conditions and the specific problem is further described by the arguments of the error. For example, **AccessError** indicates that the Mail Service rejected the requested operation. The particular problem is specified by the argument to **AccessError**, which is of type **AccessProblem**.

When an exceptional condition arises during execution of a remote procedure, the Mail Service makes every effort to undo the effects of the partial execution so that it appears to the client as though the procedure had never been called. However, the Mail Service does not guarantee that such effects can always be reversed. Therefore, when an error is reported, the client must be prepared for the possibility that the procedure was partially executed.

All MailTransport errors are defined below. Each error definition includes a declaration of the error in Courier notation, and a description of its arguments. For Remote errors generated by the Inbasket protocol, see Section 15.

---

## 5.1 Authentication errors

---

A Mail Service may refuse to perform an operation if the client's credentials or verifier are invalid or too weak. The refusal is reported by the error:

**AuthenticationError: ERROR [problem: Authentication.Problem] = 1;**

The reader should consult [Authentication] for the values and meanings of the various authentication problem codes.

## 5.2 Access errors

---

Even if the caller's credentials are valid, the caller may not be allowed to invoke the requested operation, due either to lack of access privileges or to the state of the service at the time the access is requested. This situation is reported by the error:

**AccessError: ERROR [problem: AccessProblem] = 0;**

The nature of the problem is indicated by the value of **problem**:

**AccessProblem: TYPE = {**  
    **accessRightsInsufficient(0),**  
    **accessRightsIndeterminate(1),**  
    **deliverySlotInUse(2),**  
    **noSuchDeliverySlot(3),**  
    **deliverySlotIndeterminate(4),**  
    **wrongService(5)};**

**accessRightsInsufficient** indicates that the caller did not have sufficient access rights to perform the operation in question. This could arise, for example, if a user attempted to access a mailbox to which he had not been given read permission.

**accessRightsIndeterminate** indicates that it was not possible to determine whether the caller had sufficient access rights (e.g., some other service that was needed in making the determination was unavailable).

**deliverySlotInUse** indicates that another Mail Service client was accessing the requested delivery slot at the time the call was made.

**noSuchDeliverySlot** indicates that the Mail Service determined that the specified delivery slot name was invalid. For example, this problem code is used to indicate that a specified delivery slot was not registered in the Clearinghouse; it is *not* used to indicate invalid recipients of messages during posting (see Section 5.3).

**deliverySlotIndeterminate** indicates that the validity of the specified delivery slot name could not be determined (e.g., some other service that was needed in making the determination was unavailable).

**wrongService** indicates that there is a delivery slot for the specified recipient, but it does not reside on this instance of a service.

### 5.3 Invalid recipient error

---

A **BeginPost** operation may be rejected because of invalid recipients (if the caller did not request that the message be sent to the valid recipients only). The rejection is reported by the error:

**InvalidRecipients: ERROR [nameList: InvalidNameList] = 3;**

If the **BeginPost** operation is invoked requesting delivery to valid recipients only, the call completes successfully and the invalid recipients (if any) are returned as a result of the operation.

---

### 5.4 Service errors

---

An operation may be rejected because of a lack of service resources. The rejection is reported by the error:

**ServiceError: ERROR [problem: ServiceProblem] = 4;**

**ServiceProblem: TYPE = {**  
    **cannotAuthenticate(0),**  
    **serviceFull(1),**  
    **serviceUnavailable(2),**  
    **mediumFull(3)};**

**cannotAuthenticate** indicates that necessary authentication could not be performed (e.g., because other services needed to perform the authentication were not available.)

**serviceFull** indicates that the service (not the server) has reached its maximum number of active clients and has rejected the requested operation.

**serviceUnavailable** indicates that the service is currently unavailable to network clients, even though the server is running, because the Mail program is not exported or because the server cannot provide the requested level of service.

**mediumFull** indicates that the limit on storage available to the service has been reached and the operation has been rejected because it would require additional storage.

---

### 5.5 Transfer errors

---

An operation may not complete due to failures associated with bulk data transfer. The failure is reported by the error:

**TransferError: ERROR [problem: TransferProblem] = 5;**

**TransferProblem: TYPE = {**  
    **aborted(0)};**

In some circumstances, the bulk data transfer is initiated but does not complete successfully. **aborted** indicates that the transfer was initiated but was subsequently aborted by either the source or the sink.

---

## 5.6 Session errors

---

An operation may be rejected because of incorrect session-related data. The rejection is reported by the error:

```
SessionError: ERROR [problem: SessionProblem] = 7;
SessionProblem: TYPE = {
    invalidHandle(0),
    wrongState(1)};
```

**invalidHandle** indicates that the session handle presented was not valid.

**wrongState** indicates that the requested operation was not legal given the state of the session (i.e., as determined by the preceding sequence of operations in that session). This error would be raised, for example, when a client attempts to do simultaneous calls with the same **Handle**.

---

## 5.7 Other errors

---

In unusual cases, an operation may be rejected because of an error other than one listed above. The rejection is reported by the error:

```
OtherError: ERROR [problem: OtherProblem] = 6;

OtherProblem: TYPE = {
    cantExpedite(0),
    malformedMessage(1),
    incorrectContentsSize(2),
    last(65535)};
```

**cantExpedite** indicates that an expedited call was made to a non-expedited procedure.

**malformedMessage** indicates that the message is malformed in some way. For example, the length of the **bodyPartTypesSequence** passed to **BeginPost** doesn't match the number of body parts in the message, or duplicate or illegal envelope items are specified in the **optionalPostingData** passed to **BeginPost**.

**incorrectContentsSize** is raised when the contents size specified by the client is outside the range [actualSize - 5000 bytes .. actualSize].

**last** exists so that new servers and procedures can be developed in a staged fashion; this value should never be reported by a correctly functioning server conforming to this protocol.

## 6. MailTransport conformance subsets

It is frequently useful to define subsets of a protocol by defining the minimum that must be done to provide a class of functionality. This chapter describes three conformance subsets: a subset for delivery slot clients, a subset for posting slot clients, and a subset for MailTransport service providers.

---

### 6.1 Subset for delivery slot clients

---

Delivery slot clients must implement the **BeginRetrieval**, **RetrieveEnvelope**, **RetrieveContents**, and **EndRetrieval** operations.

---

### 6.2 Subset for posting slot clients

---

Posting slot clients must implement the **BeginPost**, **PostOneBodyPart**, and **EndPost** operations. It is preferred, but not required, that the procedure given in Appendices G and H be used to locate a server for posting messages.

---

### 6.3 Subset for the service provider

---

MailTransport service providers must implement everything except Expedited Courier. It is recommended that Expedited Courier be implemented as well, to allow clients to use the procedure given in Appendices G and H for server location.



## **7. MailTransportEnvelopeFormat overview**

---

The MailTransport protocol defines the structure of a message envelope, and provides a facility for transporting arbitrary envelope data from originators to recipients. This approach to envelope transparency maximizes the flexibility of the message transfer service, but does not, in itself, provide for compatible exchange of message envelopes among a community of message transfer services.

To provide a message transfer system, it is necessary to specify, in addition, a standard format for the message envelope. This is one of the functions of the MailTransportEnvelopeFormat protocol.

The MailTransportEnvelopeFormat protocol also describes the collection of message body parts associated with delivery and nondelivery reports.

---

### **7.1 Envelope format**

---

In order for the Mail System to transport the components of an envelope transparently, each component is defined as an **EnvelopeItem**. The MailTransportEnvelopeFormat protocol is largely a set of **EnvelopeItem** definitions. Each **EnvelopeItem** has a **type** and a **value**, and the **value** is encoded as arbitrary data. The interpretation of the **value** of an **EnvelopeItem** is defined when the **type** is allocated. The Mail System does interpret the values of some **EnvelopeItems** in order to construct the envelope of a posted message. Figure 8.1 illustrates the way that **EnvelopeItems** are handled by the Mail System.

---

### **7.2 Relationship to X.400**

---

The MailTransportEnvelopeFormat standard is intended to allow systems adhering to this standard to implement all of the basic and essential optional service elements defined in CCITT recommendation X.401, with the exception of deferred delivery, deferred delivery cancellation, probe, and registered encoded information types. Other X.400 features may be supported in the future through the use of new envelope item types.

One of the **EnvelopeItems** defined in Section 8.3 of this standard, the **x400SpecificReportInformation**, is only used to carry information necessary for generating an X.400 report from the X.400 system to the XNS message transfer system. This information is simply reflected in all reports generated.

## 7.3 Reports

---

The message transfer system generates two kinds of reports, delivery reports and nondelivery reports. Unless the client requests that no report be returned, the message transfer system generates a nondelivery report when a message cannot be delivered. A delivery report, on the other hand, is generated only if a client requests it. If a client has requested a delivery report, the message transfer system generates a delivery report when the message has been added to the messages in the delivery slot or inbasket of the recipient. Note that the generation of the report is not connected in any way with the retrieval of the message by the client.

The information included in a report is contained in an **EnvelopeItem**, and consequently, the format of a report is described in detail in section 8.4.

---

## 7.4 ContentsType

---

The **ContentsType** defined in MailTransport is carried in the envelope as an **EnvelopeItem**, and distinguishes one message format from another. The numbers allocated for all **ContentsTypes** used in conjunction with published standards are recorded in the MailTransportEnvelopeFormat standard. For each **ContentsType**, a separate document exists to describe the collection of associated message body parts. For example, this protocol describes the collection of message body parts associated with the **ContentsType** used for delivery and nondelivery reports, and the MailFormat protocol describes the message body parts associated with the interpersonal message **ContentsType**.

## 8. MailTransportEnvelopeFormat definition

The XNS standard for envelope exchange is defined by the MailTransportEnvelopeFormat Courier program. This program, unlike most other Courier programs (e.g., MailTransport and Inbasket) consists entirely of data definitions, and defines no remote procedures or remote errors. In this sense, MailTransportEnvelopeFormat is not really a protocol, but rather a definition of the format of data carried by other protocols. It consists of a series of declarations of data types and constants, all in the standard Courier notation. A complete listing of the MailTransportEnvelopeFormat Courier program is given in Appendix D.

The following definition gives the version number of MailTransportEnvelopeFormat and lists all other Courier-based protocols which are referenced from this program.

```
MailTransportEnvelopeFormat: PROGRAM 1517 VERSION 1 =  
BEGIN  
    DEPENDS UPON  
        MailTransport (17) VERSION 5,  
        Time (15) VERSION 2;  
    ...  
END.
```

This document defines version 1 of MailTransportEnvelopeFormat, which references some types and constants that are defined in other protocols as shown in the above declaration. These protocols are documented in the Time Protocol [Time] and the MailTransport protocol (Chapters 3-6 and Appendix C of this document).

---

### 8.1 Common data types

---

MailTransportEnvelopeFormat depends on MailTransport for the naming of recipients, and the structure of the envelope.

```
RName: TYPE = MailTransport.RName;  
RecipientList: TYPE = MailTransport.RecipientList;  
  
Envelope: TYPE = MailTransport.Envelope;  
EnvelopeItem: TYPE = MailTransport.EnvelopeItemType;
```

Times are represented as specified in the Time protocol:

```
Time: TYPE = Time.Time;
```

Sections 8.2, 8.3 and 8.4 present envelope item definitions. As declared in the MailTransport protocol, an envelope item has two parts: its *type* field and its *value* field. The type field contains a 32-bit numeric tag. The value field is an uninterpreted block of data encoded as a SEQUENCE OF UNSPECIFIED, where the interpretation of the data (i.e., the UNSPECIFIED) as a specific Courier data type depends on the contents of the type field. Thus, the definition of a specific

envelope item consists of an assigned code for its type field (a 32-bit constant), and a specification of the Courier data type to be used in its value field. The procedure for allocating new envelope item types is given in Appendix B.

The body parts of a message are numbered from zero to the actual number of body parts minus one. The numbers are assigned to the body parts in the order that they appear in the **tableOfContents** (see Section 8.2). A **BodyPartIndex** can be used to refer to a particular message body part.

**BodyPartIndex:** TYPE = CARDINAL

---

## 8.2 Required envelope item types

---

This section specifies the envelope items that are required to appear in every envelope constructed by the Mail System.

The **postmark** item indicates the time and site at which the message was posted by the originator's user agent. It consists of the name of the Mail Service or foreign mail system at which posting occurred, and the time at which posting occurred. It is filled in by the Mail System.

**postmark:** EnvelopeItemtype = 0;

**Postmark:** TYPE = RECORD [postedAt: RName, time:Time];

The **messageID** identifies the message uniquely, distinguishing it from all other messages ever handled by the Mail System. It is filled in by the Mail System when the message is posted.

**messageID:** EnvelopeItemtype = 1;

**MessageID:** TYPE = MailTransport.MessageID;

The type of the message content is indicated by the **contentType**. The value for this item is supplied by the client to the **BeginPost** operation. Currently defined values for **contentType** are listed in Section 8.6.

**contentType:** EnvelopeItemtype = 2;

**ContentsType:** MailTransport.ContentsType;

The **tableOfContents** is a list of the body parts of a message. Each element in the list contains the type of the body part and its size in bytes. The **sizeInBytes** is filled in by the message transfer service when the message is posted. The **nullTableOfContents** is used when a message has zero body parts.

**tableOfContents:** EnvelopeItemtype = 3;

**TableOfContents:** TYPE = SEQUENCE 500 OF RECORD [  
    type: MailTransport.BodyPartType, sizeInBytes: LONG CARDINAL];

**nullTableOfContents:** TableOfContents = RECORD [ ];

The size of the message contents in bytes is indicated by the **contentsSize**, which is filled in by the Mail System. If the exact size of the message was supplied to the **BeginPost** operation then the contents size parameter to **BeginPost** and the **contentsSize** envelope item will have

the same value. Note that the **contentsSize** does not include the size of the message envelope.

**contentsSize: EnvelopeItemType = 4;**  
**ContentsSize: TYPE = LONG CARDINAL;**

The name of the originator is filled in by the Mail System from the originator's credentials when a message is posted. Consequently, when a message originates in the XNS mail system, the originator name is authenticated information, in the sense that its validity is not based on the trustworthiness of the originator. Its validity is, however, based on the strength of the authentication scheme used by the originator. The **authenticationLevelOfSender** envelope item described below is used to indicate the strength of the originator's authentication.

**originator: EnvelopeItemType = 5;**  
**Originator: TYPE = RName;**

The **authenticationLevelOfSender** indicates the form of authentication that was used to validate the identity of the message originator. A level of **strong** or **simple** means that the corresponding XNS authentication was done. A value of **foreign** means that the message originator was authenticated in a foreign system using the authentication scheme of the foreign system. This envelope item is usually generated by the Mail System, but the client can supply it, with a value of **foreign** to **BeginPost**.

**authenticationLevelOfSender: EnvelopeItemType = 16;**  
**AuthenticationLevelOfSender: TYPE = {strong (0), simple (1), foreign (2)};**

---

### 8.3 Other envelope item types

---

The envelope item types defined in this section may appear in envelopes constructed by the Mail System.

The **returnToName** indicates the user to whom reports should be sent. It is filled in and used by the Mail System and is not interesting to users.

**returnToName: EnvelopeItemType = 7;**  
**ReturnToName: TYPE = RName;**

The **previousRecipients** list is a list of recipients the Mail System has attempted to deliver the message to. It is filled in and used by the Mail System and is not interesting to users.

**previousRecipients: EnvelopeItemType = 8;**  
**PreviousRecipients: TYPE = RecipientList;**

The **gatewayPostmark** is filled in by a mail gateway when it submits a message to the Mail System.

**gatewayPostmark: EnvelopeItemType = 9;**  
**GatewayPostmark: TYPE = Postmark;**

When a message is delivered to a different recipient name than it was addressed to and both the name the message was addressed to and the name the message was delivered to refer to the same entity, the **addressChangeNotice** envelope item is present. The value of the **addressChangeNotice** item is the name of the recipient the message was addressed to.

**addressChangeNotice: EnvelopeItemType = 10;**  
**AddressChangeNotice: TYPE = RName;**

The client of MailTransport can request that the contents of a message be returned if the Mail System cannot deliver the message. This is done by setting **returnOfContents** in the **transportOptions** to **TRUE**. The **alternateRecipientAllowed** option determines whether the message can be delivered to an entity not named on the envelope. For example, if **alternateRecipientAllowed** is **FALSE**, the message cannot be delivered to the dead letter office. This field is generated by the client of the message transfer service.

**transportOptions: EnvelopeItemType = 11;**  
**TransportOptions: TYPE = RECORD [**  
    **returnOfContents: BOOLEAN,**  
    **alternateRecipientAllowed: BOOLEAN];**

The **x400SpecificReportInformation** is only used to carry information necessary for generating an X.400 report from the X.400 system to the Mail System. This information is reflected in all reports destined for X.400 recipients. This field is generated by gateways from the X.400 system. The X.400 specific information contained in this envelope item is the **mpdulIdentifier**, the **uaContentID**, the **PerRecipientFlag** and **ExtensionIdentifier** of each recipient, and the **intermediateTraceInformation**. The **mpdulIdentifier**, the **uaContentID**, the **PerRecipientFlag**, and **ExtensionIdentifier** of each recipient are always present. The **intermediateTraceInformation** is only present if an audit and confirmed report was requested.

**x400SpecificReportInformation: EnvelopeItemType = 12;**  
**X400SpecificReportInformation: TYPE = SEQUENCE 4 OF X400ReportItem;**  
**X400ReportItem: TYPE = SEQUENCE OF UNSPECIFIED;**

When **otherRecipients** is present, it contains all of the recipient names in the original envelope. This field is generated by gateways from the X.400 system.

**otherRecipients: EnvelopeItemType = 13;**  
**OtherRecipients: TYPE = RecipientList;**

The **priority** of a message determines the way it will be handled by the Mail System. Urgent messages will be processed before normal messages, and normal messages will be processed before nonUrgent messages. This envelope item is generated by the Mail System client.

**priority: EnvelopeItemType = 14;**  
**Priority: TYPE = {nonUrgent(0), normal(1), urgent(2)};**

The **converted** envelope item describes the conversions a message has undergone. This item applies to the entire message. That is, the entire message has been converted once for each element in the **SEQUENCE**. This envelope item can be generated by either the mail transfer service or the Mail System client.

**ConvertedItem: TYPE = {ia5TextToTeletex(0), teletexToTelex(1), teletexToIA5Text(2),  
    telexToTeletex(3)};**

converted: EnvelopeItemType = 15;  
Converted: TYPE = SEQUENCE 10 OF ConvertedItem;

It is possible that subsequent envelope item types may appear due to the inherent extensibility of the envelope item mechanism. These are beyond the scope of this standard, and should be ignored by clients of this standard. That is, when an envelope is displayed to an end user, these envelope items should be skipped.

---

## 8.4 Reports

---

The **report** is used to report when a message cannot be delivered, when a message was successfully delivered, when a delivery or nondelivery report could not be delivered, and when a nondelivery report was delivered but the return of contents service could not be provided.

### 8.4.1 Report envelopes

---

A **report** is implemented as an **EnvelopeItem** of a message with a **ContentsType** of **ctReport**.

**report**: EnvelopeItemType = 6;

**ctReport**: ContentsType = 6;

### 8.4.2 Report contents

---

The contents of reports generated by the Xerox Mail System vary according to the **ContentsType** of the original message. For interpersonal messages, the report always contains the heading body part of the original message. If return of contents is requested, and the Mail Service is able to comply, the contents of a nondelivery report will consist of exactly those message body parts that were in the original message. For non-interpersonal messages, the delivery report never has contents, and the nondelivery report has contents if and only if the client requested return of contents. The contents of the report in the latter case are the contents of the original message.

When a report has no contents, the **tableOfContents** in the envelope containing the **report** is the **nullTableOfContents** defined in MailTransportEnvelopeFormat.

Delivery and nondelivery reports generated by the X.400 Mail System will not contain enough information for the Xerox Mail System to reconstruct the heading information of the original message. Therefore, clients should expect to see reports for interpersonal messages which do not contain a heading body part.

### 8.4.3 The report envelope item

---

If a message cannot be delivered, it contains an indication of the recipients to whom the message could not be delivered and for each, the reason for nondelivery. This information is represented by the **report** item in the envelope. Note that the envelope containing the **report** item is a system-generated envelope used to carry the returned message, as distinct from the envelope used to transport the original message, which is included in the **report** in the **originalEnvelope** field.

**NonDeliveredRecipient:** TYPE = RECORD [  
    **recipient:** MailTransport.Recipient, **reason:** Reason];

**Reason:** TYPE = {noSuchRecipient(0), noMailboxForRecipient(1), illegalName(2),  
    timeout(3), reportNotAllowed(4), messageTooLong(5), ambiguousRName(6),  
    illegalCharacters(7), unsupportedBodyParts(8), unsupportedContentsType(9),  
    transientProblem(10), contentSyntaxError(11), tooManyRecipients(12),  
    protocolViolation(13), x400PragmaticConstraintViolation(14),  
    x400NoBilateralAgreement(15), accessRightsInsufficientForDL(16), other(17)};

The reasons that a message could not be delivered are interpreted as follows:

**noSuchRecipient:** the name doesn't exist. That is, the name is not known to the Mail Service.

**noMailboxForRecipient:** the name exists, but there is no mailbox registered for the recipient.

**illegalName:** the name does not have the syntax of an RName.

**timeout:** the message could not be delivered within some specified time period.

**reportNotAllowed:** the recipient is a distribution list, and the client requested a delivery report for the distribution list (delivery reports are not supported for distribution lists).

**messageTooLong:** the message was too long to be accepted at the destination or at a Mail Service en route to the destination.

**ambiguousRName:** the message was bound for a recipient in a foreign mail system and the name specified by the user was not unique.

**illegalCharacters:** the message was bound for a recipient in a foreign mail system and the message contained characters which could not be interpreted by the foreign system.

**unsupportedBodyParts:** the message was bound for a recipient in a foreign mail system, contained body parts which could not be interpreted by the foreign recipient, and this resulted in nondelivery of the entire message.

**unsupportedContentsType:** the message was bound for a recipient in a foreign mail system and the message format was not supported in the foreign system.

**transientProblem:** the message could not be delivered due to a problem which is likely to be transient. An example of a situation that could cause this error to be raised is the phone lines are down, If the message is sent again, it will probably be successfully delivered.

**contentSyntaxError:** there is something wrong with the structure of a body part. This reason is normally reported by a mail gateway that is attempting to convert the contents of the message to another format, or by a foreign mail system.

**tooManyRecipients:** the receiving mail system could not handle the message because it was sent to too many recipients.

**protocolViolation:** a mandatory protocol element is missing, or the message is improperly encoded (i.e., the Courier data types in the Xerox Mail System or the equivalent data types in a foreign mail system are incorrect).

**x400PragmaticConstraintViolation:** a message sent from the Xerox Mail System to the X.400 Mail System violated the X.400 pragmatic constraints [X.400 Implementor's Guide].

**x400NoBilateralAgreement:** a message containing a request for a service provided only by bilateral agreement was sent to an X.400 Mail System that had not made an agreement to provide the service.

**accessRightsInsufficientForDL:** the originator did not have access to send to one of the distribution lists the message was addressed to.

**other:** exists so that new servers and procedures can be developed in a staged fashion; this value should never be reported by a correctly functioning server conforming to this protocol.

If a message was successfully delivered, the **report** contains a list of the recipients the message was delivered to and the time the message was delivered. Note that a **report** can contain information about both recipients the message could not be delivered to and recipients the message was delivered to.

**DeliveredRecipient:** TYPE = RECORD [  
     **recipient:** MailTransport.Recipient, **when:** Time];

For both delivery and nondelivery reports, the **recipientID** in the **recipient** information should give the position of the recipient name in the recipient list on the original envelope. This information is provided to ease the correlation of reports with the original message. Note however, that **recipientID** is provided by the User Agent when the message is posted, and that the Mail System does not ensure that the **recipientIDs** given in the parameters to **BeginPost** match the names in the heading body part.

If a message could not be delivered to members of a distribution list, the report contains the name of the distribution list which immediately contains the members, a list of distribution list members that the message was not delivered to, and the reasons for nondelivery.

**Report:** TYPE = RECORD [  
     **originalEnvelope:** Envelope  
     **fateOfThisReport:** CHOICE OF {  
         **delivered** = > {**contentsTruncated**(0), **noProblem**(1)},  
         **notDelivered** = > RECORD [  
             **reason:** Reason  
             **reportCreationPostmark:** Postmark]],  
         **reportType:** CHOICE OF {  
             **dliMember**(0) = > RECORD [  
                 **dliName:** RName,  
                 **invalidDLRecipients:** SEQUENCE of NonDeliveredRecipient],  
             **other**(1) = > RECORD [  
                 **succeeded:** SEQUENCE of DeliveredRecipient,  
                 **failed:** SEQUENCE of NonDeliveredRecipient]]];

Note that a report about distribution list members does not contain information about successful or unsuccessful delivery to non-distribution list members. Note also that many reports may be generated for a single message. Finally, the **fateOfThisReport CHOICE** is used to note two situations that should occur infrequently: if a delivery or nondelivery report could not be delivered, (i.e., theMailSystem is no longer trying to deliver the report and has relegated it to the electronic equivalent of the dead letter office), the **notDelivered RECORD** indicates the reason for nondelivery and gives the name of the Mail Service that generated the report along with the time the report was generated; in a nondelivery report, **contentsTruncated** indicates that although it was requested to do so, the Mail System was unable to return the contents of the original message along with the report.

---

## 8.5 Envelope item usage

---

There are three phases in the life of an envelope: it is created when a message is posted, it travels through the Mail System, and then it is delivered. Different envelope items appear in the envelope in each of the phases. Figure 8.1 lists the envelope items that are required (R) to appear in each of the phases, those that are optional (O), and those that are not allowed (N). In addition, the table indicates whether an item is supplied by the Mail System (MS) or the Mail System client (C). Note that when new envelope item types are allocated, the client can associate them with a message by specifying them in the **optionalPostingData** of the **BeginPost** call.

| Envelope Item                 | optional<br>Envelope<br>Data in<br>Posting | Transit | Delivery | Supplied<br>By           |
|-------------------------------|--------------------------------------------|---------|----------|--------------------------|
| addressChangeNotice           | O                                          | O       | O        | MS                       |
| authenticationLevelOfSender   | O <sup>3</sup>                             | R       | O        | MS <sup>3</sup>          |
| contentsSize                  | N                                          | R       | R        | MS                       |
| contentsType                  | N                                          | R       | R        | C <sup>1</sup>           |
| converted                     | O                                          | O       | O        | MS or C                  |
| gatewayPostmark               | O                                          | O       | O        | MS                       |
| messageID                     | N                                          | R       | R        | MS                       |
| originator                    | N                                          | R       | R        | MS                       |
| otherRecipients               | N                                          | O       | O        | MS                       |
| postmark                      | N                                          | R       | R        | MS                       |
| previousRecipients            | O                                          | R       | O        | MS                       |
| priority                      | O                                          | O       | O        | C                        |
| report                        | O                                          | O       | O        | MS                       |
| returnToName                  | N                                          | R       | O        | MS                       |
| tableOfContents               | N                                          | R       | R        | MS and<br>C <sup>2</sup> |
| transportOptions              | O                                          | O       | O        | C                        |
| x400SpecificReportInformation | N                                          | O       | O        | MS                       |

Figure 8.1 Envelope item usage

N = Not Allowed O = Optional R = Required MS = Mail System C = Client

<sup>1</sup>The **contentsType** is supplied by the client via the **postingData** of the **BeginPost** operation.

<sup>2</sup>The body part types are supplied by the client via the **postingData** of the **BeginPost** call. The body part sizes are supplied by the Mail Service.

<sup>3</sup>A value of **foreign** can be supplied to **BeginPost** (e.g., by a client that is serving as a gateway to a foreign mail system).

## 8.6 Standard contents types

---

A list of contents types used in conjunction with published standards is given below. The procedure for obtaining new contents types is given in Appendix B.

The absence of message contents, that is, an envelope-only message is described by **ctNull**.

**ctNull: ContentType = 3777777777B;**

The collection of message body parts associated with the **ctStandardMessage** contents type is described in Section 11.5.

**ctStandardMessage: ContentType = 4;**

The collection of message body parts associated with the **ctReport** contents type is described in 8.4.2.

**ctReport: ContentType = 6;**

This chapter discusses subsets of the MailTransportEnvelopeFormat protocol. The simple client and the X.400 client can implement subsets of the MailTransportEnvelopeFormat protocol, while the service provider must implement the entire protocol and satisfy additional requirements.

---

### 9.1 Subset for the simple client

---

All clients must be able to receive all envelope items. However, many of these items need not be conveyed to the ultimate client in cases where it is a human user. The envelope items that must be conveyed to the user are given in Figure 9.1. It is recommended that the user have access to more than the minimal set of envelope items. The set of envelope items that are useful to users is also given in Figure 9.1.

---

### 9.2 Subset for the X.400 client

---

The X.400 client, just as any other client, must be able to receive all envelope items. In addition, the CCITT X.400 series of recommendations [X.400] requires that some of the information in the envelope be made accessible to the ultimate client when it is a human user. The envelope items that must be conveyed to the user by the X.400 client are given in Figure 9.1.

---

### 9.3 Requirements for the service provider

---

The service provider must provide the full MailTransportEnvelopeFormat protocol. In addition, the service provider must be able to accept newly defined envelope items and contents types without having to modify their implementation of the protocol. New envelope items and messages with new contents types must be conveyed to clients. Note that no validation of new envelope items and contents types is required.

| ENVELOPE ITEM                 | REQUIRED<br>FOR SIMPLE<br>CLIENT | REQUIRED<br>FOR X.400<br>CLIENT | USEFUL TO<br>END USERS |
|-------------------------------|----------------------------------|---------------------------------|------------------------|
| addressChangeNotice           | N                                | N                               | Y                      |
| authenticationLevelOfSender   | Y                                | N                               | Y                      |
| contentsSize                  | note 2                           | note 2                          | note 2                 |
| contentsType                  | note 1                           | note 1                          | note 1                 |
| converted                     | N                                | Y                               | Y                      |
| gatewayPostmark               | N                                | N                               | Y                      |
| messageID                     | Y                                | Y                               | Y                      |
| originator                    | Y                                | Y                               | Y                      |
| otherRecipients               | N                                | Y                               | N                      |
| postmark                      | Y                                | Y                               | Y                      |
| previousRecipients            | N                                | N                               | N                      |
| priority                      | N                                | Y                               | Y                      |
| report                        |                                  |                                 |                        |
| originalEnvelope              | note 3                           | note 3                          | note 3                 |
| fateOfThisReport              | N                                | N                               | Y                      |
| reportType                    | Y                                | Y                               | Y                      |
| returnToName                  | N                                | N                               | N                      |
| tableOfContents               | Y                                | Y                               | Y                      |
| transportOptions              | N                                | Y                               | Y                      |
| x400SpecificReportInformation | N                                | N                               | N                      |

Figure 9.1 Accessibility of envelope items to the user

**NOTE 1:** The **contentsType** should be accessible to the end user if it will help the user to display the message. For example, if a user receives a message with a new contents type which represents a form, there might be a user agent that can display the form, but the user won't know whether to use this user agent unless the contents type of the message is displayed.

**NOTE 2:** The **contentsSize** is the sum of the sizes in the **tableOfContents**, so it is not necessary to present both the **tableOfContents** and the **contentsSize** to the end user.

**NOTE 3:** When a report is presented to the end user, the **originalEnvelope** should be presented in the same way as an envelope is presented for a message.

The message transfer system, as accessed via the MailTransport protocol, defines a data-transparent communication facility capable of transporting arbitrary data from originators to recipients. The *envelope* of a message names the recipients and otherwise describes the handling of the message as it is transported. The envelope is thus the part of the message that is interpreted by the message transfer system, while the content is completely uninterpreted.

This approach to data transparency maximizes the flexibility of the message transfer system but does not, in itself, provide for compatible exchange of human-readable messages among users. To provide an interpersonal message system, it is necessary to specify, in addition, a standard format for one particular form of contents object that can be transported along with an envelope: the interpersonal message. The function of the MailFormat standard is to define the set of message body parts associated with an interpersonal message.

---

## 10.1 Heading and body

---

The fundamental structural division defined by MailFormat is between the heading and body of an interpersonal message. The heading body part corresponds to the familiar fields that would be found, for example, at the beginning of a memo in the "paper mail" environment. Such fields represent a set of standard attributes of the message, such as: who sent it, who is to receive it, what it is about, and so on. Following the heading is the body, which is made up of a set of message body parts, and contains the actual substance of the message. As described in 10.3, the body may contain data in a variety of formats.

---

## 10.2 Relationship to X.400

---

The MailFormat standard is intended to exhibit functional alignment with X.400—specifically with the P2 standard for interpersonal messages defined in X.420. However, there is a structural difference between X.420 and MailFormat. In MailFormat, the heading of a message is represented as a message body part, while in X.420, an interpersonal message consists of a structured heading followed by a sequence of message body parts. This difference arises because all contents types are modeled as a sequence of message body parts in the MailTransport protocol (see Section 8.2).

## 10.3 Data encapsulation and transparency

Together, the MailTransport and MailFormat standards define a two-level data encapsulation scheme, as depicted in Figure 10.1.

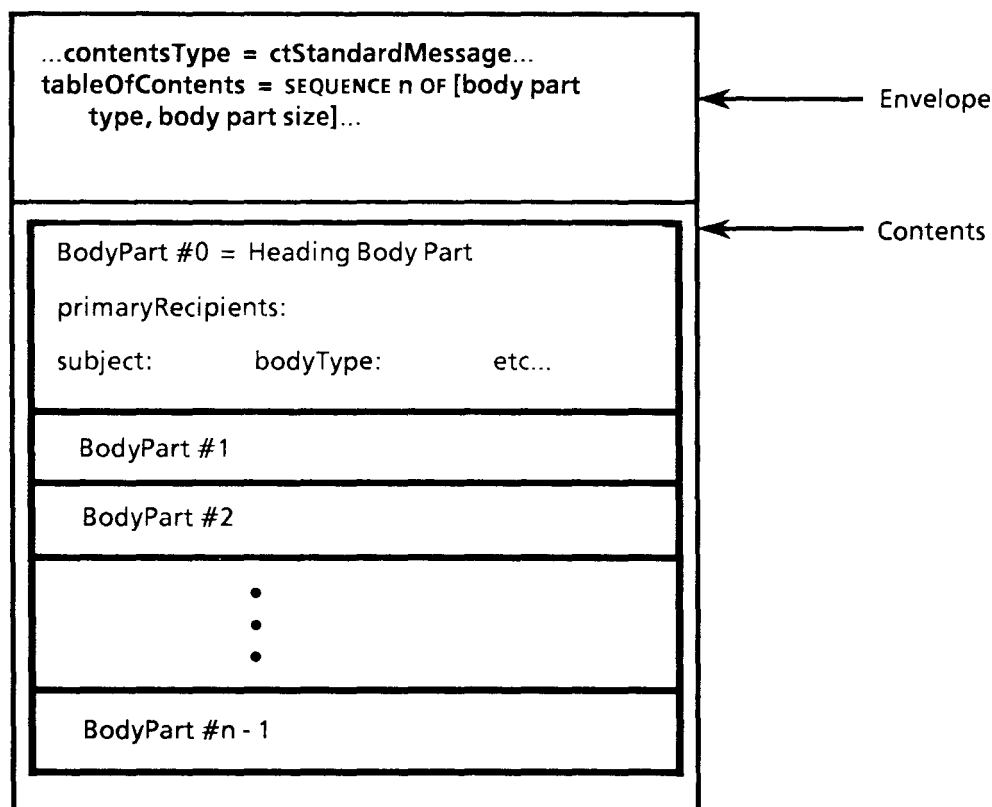


Figure 10.1

The two levels of data transparency thus defined are provided for substantially different purposes.

At the outer (MailTransport) level, the data transparency of the contents is intended to support other uses in addition to interpersonal message communication. For example, two computers might wish to mail each other binary data in an agreed-upon format. This would not constitute interpersonal messaging, and the MailFormat standard would therefore not apply.

At the inner (MailFormat) level, the data transparency of the body is intended to support multiple formats (e.g., multiple document types), all of which can be communicated as interpersonal messages using the standard heading information and one or more message body parts.

The XNS standard for interpersonal messages is defined by the MailFormat Courier program. This program, unlike most other Courier programs (e.g., MailTransport and Inbasket), consists entirely of data definitions, and defines no remote procedures or remote errors. In this sense, MailFormat is not really a protocol, but rather a definition of the format of data carried by other protocols. It consists of a series of declarations of data types and constants, all in the standard Courier notation. A complete listing of the MailFormat Courier program is given in Appendix E.

The following definition gives the version number of MailFormat and lists all other Courier-based protocols which are referenced from this program.

```
MailFormat: PROGRAM 1516 VERSION 3 =
BEGIN
    DEPENDS UPON

        Time (15) VERSION 2,
        MailTransport (17) VERSION 5,
        MailTransportEnvelopeFormat (1517) VERSION 1;

    ...
END.
```

This document defines version 3 of MailFormat. MailFormat references some types and constants that are defined in other protocols as shown in the above declaration. These protocols are documented in the Time Protocol [Time], the MailTransport protocol (Chapters 3-6 and Appendix C of this document), and the MailTransportEnvelopeFormat protocol (Chapters 7-9 and Appendix D of this document).

---

## 11.1 Interpersonal message format

---

The format of the content of a message is indicated by the value of its envelope's **contentsType** field. The value indicating the presence of an interpersonal message is **MailTransportEnvelopeFormat.ctStandardMessage**.

```
BodyPartType: TYPE = MailTransport.BodyPartType;
headingBodyPart: BodyPartType = 0;
HeadingBodyPart: TYPE = SEQUENCE 150 OF Attribute;
```

All messages consist of a sequence of message body parts. Interpersonal messages contain a **headingBodyPart** followed by zero or more body parts containing the actual substance of the message. The heading body part is made up of attributes which are described in Section 11.3. Message body parts used in interpersonal messages are described in 11.5.

---

## 11.2 Common data types

---

This section discusses data types that are used throughout the MailFormat standard.

An attribute has two parts: its *type field* and its *value field*. The *type field* contains a 32-bit numeric tag. The *value field* is an uninterpreted block of data encoded as a **SEQUENCE OF UNSPECIFIED**, where the interpretation of the data (i.e., the **UNSPECIFIED**) as a specific Courier data type depends on the contents of the *type field*. Thus, the definition of a new attribute consists of an assigned code for its *type field* (a 32-bit constant), and a specification of the Courier data type to be used in its *value field*. The procedure for allocating new attribute types is given in Appendix B.

**Attribute: TYPE = RECORD[type: AttributeType, value: SEQUENCE OF UNSPECIFIED];**

**AttributeType: TYPE = LONG CARDINAL;**

Times are represented as specified in the Time protocol:

**Time: TYPE = Time.Time;**

Originators and recipients are specified using **RNames**:

**RName: TYPE = MailTransport.RName;**

Where multiple users are to be specified, an **RNameList** is used:

**RNameList: TYPE = SEQUENCE OF RName;**

Identifiers of interpersonal messages are represented by **IPMessageIDs**:

**IPMessageID: TYPE = RECORD [originator: RName, uniqueString: STRING];**

The **uniqueString** is limited to 64 bytes. It is included to distinguish among multiple messages from the same originator. The suggested (although not mandatory) implementation of this is the use of the date and time, encoded as a character string. The **originator** name is optional in the X.400 series of recommendations, so it is possible for a client to receive two different messages with the same **IPMessageID** from the X.400 system, though this should occur infrequently.

Where multiple messages are to be specified, an **IPMessageIDList** is used:

**IPMessageIDList: TYPE = SEQUENCE 100 OF IPMessageID;**

## 11.3 Message heading attributes

---

This section presents a sequence of attribute definitions. Though none of the attributes is mandatory, it is strongly recommended that the subject and messageID be provided in all interpersonal messages. The attributes defined here make up the heading of an interpersonal message. Most of these attributes correspond with the identically named heading components defined in X.420.

All attributes are generated by the user agent and carried transparently by the message transfer system.

Each interpersonal message is assigned a unique message identifier. Note that this is *not* the same as the identifier assigned by the message transfer system (see 4.1.6).

**messageID: AttributeType = 1;**  
**MessageID: TYPE = IPMessageID;**

The sender of a message is indicated by the **originator** attribute. The originator's name also appears in the message envelope, so this attribute is frequently omitted from the heading.

**originator: AttributeType = 2;**  
**Originator: TYPE = RName;**

The originator may or may not correspond to the user(s) under whose authority the message contents are being communicated to the recipients. The users who authorize the sending of a message are indicated in the **authorizingUsers** attribute. If the message is authorized by the user who actually submits it, the **authorizingUsers** attribute is typically omitted.

**authorizingUsers: AttributeType = 3;**  
**AuthorizingUsers: TYPE = RNameList;**

The principle users to whom a message is sent are indicated by the **primaryRecipients** attribute:

**primaryRecipients: AttributeType = 4;**  
**PrimaryRecipients: TYPE = RNameList;**

Other users who are to receive a copy of the message are indicated by the **copyRecipients** attribute:

**copyRecipients: AttributeType = 5;**  
**CopyRecipients: TYPE = RNameList;**

The names of the copy recipients are disclosed to all recipients. Recipients who are to receive the message but whose reception is *not* to be disclosed to other recipients are indicated by the **blindCopyRecipients** attribute.

**blindCopyRecipients: AttributeType = 6;**  
**BlindCopyRecipients: TYPE = RNameList;**

The presence of a blind copy recipient is not revealed to any other recipients, including other blind copy recipients. When a user requests that a message be sent to blind copy recipients, the user agent generates several messages with the same IPMessageID that differ only in the appearance of the blind copy recipients attribute as follows: One of these messages has

no blind copy recipients attribute, and is sent to all of the primary and copy recipients. Another message is generated for each blind copy recipient. Each of these additional messages contains the names of all of the primary and copy recipients, but has the name of only one of the blind copy recipients in the blind copy recipients field, and is sent only to that recipient. The user agent controls which recipients the message is actually sent to through the parameters to the MailTransport **BeginPost** operation.

The identity of a previous message to which a given message is a response is indicated by the **inReplyTo** attribute. This attribute indicates the identifier of the message being replied to, and is omitted if the message is not a reply to an earlier message.

**inReplyTo: AttributeType = 7;**  
**InReplyTo: TYPE = IPMessageID;**

If a given message supersedes an earlier message, this is indicated by the **obsoletes** attribute. This attribute indicates the identifier of the message that is being superseded.

**obsoletes: AttributeType = 8;**  
**Obsoletes: TYPE = IPMessageIDList;**

A cross-reference to one or more other messages can be indicated by the **crossReferences** attribute. Each message being referenced is indicated by its identifier.

**crossReferences: AttributeType = 9;**  
**CrossReferences: TYPE = IPMessageIDList;**

The subject matter of a message is indicated by the **subject** attribute. The **subject** is limited to 256 bytes.

**subject: AttributeType = 10;**  
**Subject: TYPE = STRING;**

The date on which a message loses effect is indicated by the **expiryDate** attribute. The expiry date indicates the originator's statement of the point in time at which the content of the message is no longer meaningful. It does not control any action (e.g., automatic deletion of the message) by the message transfer system.

**expiryDate: AttributeType = 11;**  
**ExpiryDate: TYPE = Time;**

The date by which a reply is requested is indicated by the **replyBy** attribute:

**replyBy: AttributeType = 12;**  
**ReplyBy: TYPE = Time;**

The user(s) to whom a reply should be directed are indicated by the **replyToUsers** attribute:

**replyToUsers: AttributeType = 13;**  
**ReplyToUsers: TYPE = RNameList;**

The originator's estimate of the importance of a message is indicated by the **importance** attribute. If this attribute is missing, the importance of the message is assumed to be **normal**.

**importance: AttributeType = 14;**  
**Importance: TYPE = {low(0), normal(1), high(2)};**

The originator's estimate of the sensitivity of a message is indicated by the **sensitivity** attribute. This attribute is optional and no particular level of sensitivity is assumed if none is specified.

**sensitivity: AttributeType = 15;**  
**Sensitivity: TYPE = {personal(0), private(1), companyConfidential(2)};**

A message is considered to have been **autoforwarded** when a user agent has included it in the body of another message without any user intervention. In some systems, for example, people can set their user agents to forward all of their mail to someone else while they are on vacation. This feature will be implemented in the Xerox Mail System, but mail will be forwarded by the Mail System instead of by the User Agent. Consequently, **autoforwarding** is of marginal value in the Xerox Mail System. An autoforwarded message is indicated by the **autoforwarded** attribute. If this attribute is omitted, it is assumed that the message has not been autoforwarded.

**autoforwarded: AttributeType = 16;**  
**Autoforwarded: TYPE = BOOLEAN;**

The originator can indicate that the message content should not be modified through the **immutable** attribute. This attribute is optional, and if it is present, a mail gateway should not modify the message content in any way. That is, conversion should not be performed, nor should message text be rearranged.

**immutable: AttributeType = 17;**  
**Immutable: TYPE = ARRAY 0 OF UNSPECIFIED;**

The **replyRequest** attribute is used to indicate the recipients who are expected to reply to the message. This attribute is optional.

**replyRequest: AttributeType = 18;**  
**ReplyRequest: TYPE = RNameList;**

Some foreign mail systems allow users to generate arbitrary text heading items such as "word of the day:happiness". The **textAnnotation** attribute is intended to allow the use of arbitrary text heading items in the MailFormat standard. The length of this attribute is limited to 2000 characters, and only one of these attributes is allowed in a message.

**text-Annotation: AttributeType = 19;**  
**Text-Annotation: TYPE = STRING;**

When a message is forwarded as a part of another message, the **forwardedHeadings** attribute is present. Each element in the **SEQUENCE OF ForwardedMessageInfo** identifies a forwarded message. The **heading** contains the attributes in the heading body part of the forwarded message. The **associatedBodyParts** identify the message body parts of the forwarded message. And the **indexOfParentHeading** indicates whether this forwarded message was nested within another forwarded message, and if so, gives the index into the **SEQUENCE OF ForwardedMessageInfo** which identifies the parent message. This method of forwarding messages is similar to that used by X.400 but the **forwardedHeadings** attribute gives the user agent the option of using the tree structure of the forwarded message defined in X.400. See Figure 11.1 for a diagram of a message before and after it is forwarded.

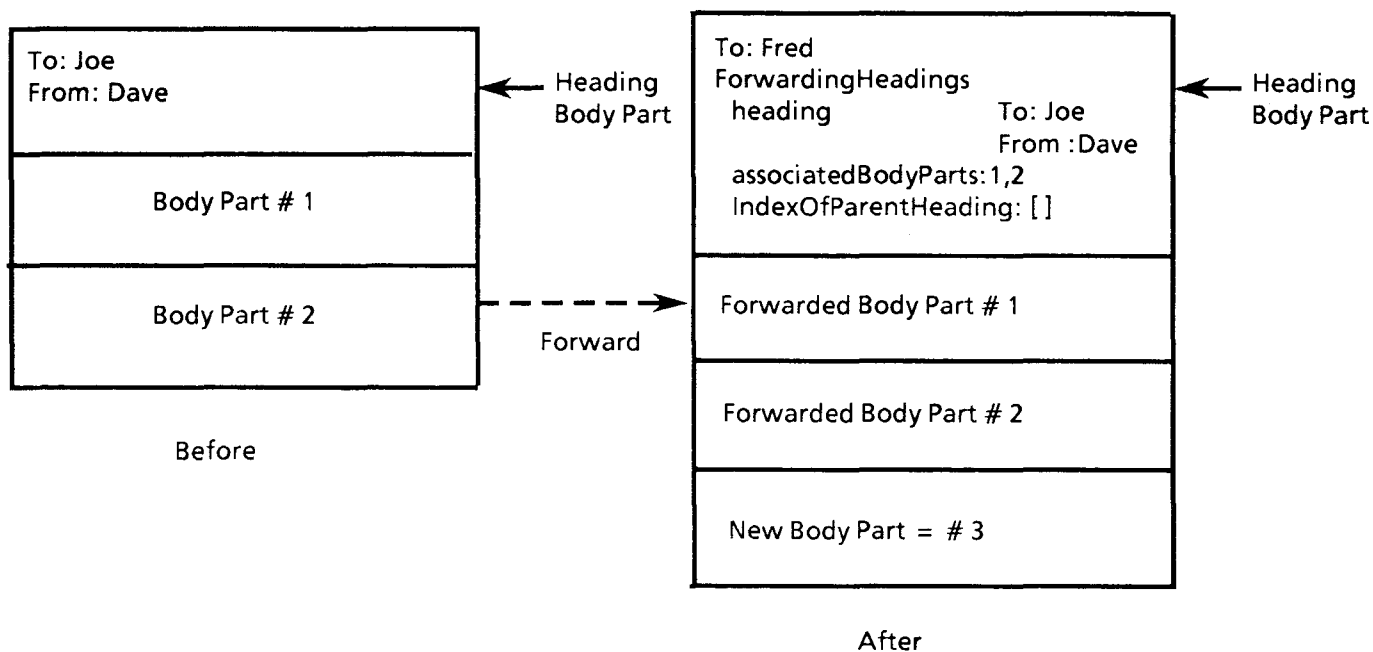


figure 11.1 Forwarding a message

```

forwardedHeadings: AttributeType = 20;
ForwardedHeadings: TYPE = SEQUENCE OF ForwardedMessageInfo;
ForwardedMessageInfo: TYPE = RECORD [
    heading: SEQUENCE 150 OF Attribute,
    associatedBodyParts: SEQUENCE 500 OF BodyPartIndex,
    indexOfParentHeading: IndexOfParentHeading];
IndexOfParentHeading: TYPE = CHOICE OF {
    null (0) = > RECORD [],
    nested (1) = > RECORD [index: CARDINAL]};

```

## 11.4 Other attributes

It is possible that additional attribute types may appear due to the inherent extensibility of the attribute mechanism. These are beyond the scope of this version of the standard, and should be ignored by clients of this standard. That is, if an unknown attribute type appears in a heading, the user agent should skip over it when displaying the rest of the attributes of the message.

## 11.5 Message body part types for interpersonal messages

---

This section specifies the message body part types that are used in interpersonal messages. Each message body part definition has two parts: its *type field*, and its English language description. The type field contains a 32-bit numeric tag. The English language description is a reference to the standard which describes the format of the body part.

It is possible that additional body part types may appear due to the inherent extensibility of the body part mechanism. These are beyond the scope of this version of the standard.

When a user agent notices that a message contains an unknown message body part or a message body part that the user agent cannot display, the user should be notified that the message contains the message body part type in question. The user agent need not retrieve or display such a message body part.

A **vpFolder** is encoded as a Xerox Filing Protocol **Filing.SerializedFile** [Filing] and represents any of the folders used by ViewPoint. This body part type is used to allow clients to notify their users that a message contains a folder, and to allow the client to decide whether to retrieve the body part instead of forcing them to retrieve it to decode the type information in the SerializedFile format that specifies that the file is a folder.

**vpFolder: BodyPartType = 1;**

An **nsTextFile** is encoded as a Xerox Filing Protocol **Filing.SerializedFile** [Filing] and represents any multinational text file which does not include specialized rendering or formality characters. This body part type is used to allow clients to decide whether to retrieve the body part instead of forcing them to retrieve it to determine the type information encoded in the SerializedFile format.

**nsTextFile: BodyPartType = 2;**

A **vpDocument** is encoded as a Xerox Filing Protocol **Filing.SerializedFile** [Filing] and represents any of the ViewPoint documents. This body part type is used to allow clients to notify their users that the body part is a ViewPoint document, and to allow clients to decide whether to retrieve the body part instead of forcing them to retrieve it to determine the type information encoded in the SerializedFile format.

**vpDocument: BodyPartType = 3;**

This body part type is encoded as a Xerox Filing Protocol **Filing.SerializedFile** [Filing] and represents any serializedFile which is not a **vpFolder**, **nsTextFile**, or a **vpDocument**. It is always acceptable to send a non-text SerializedFile as an **otherNSFile**, because the SerializedFile contains enough information for the recipient to identify the type of the file.

**otherNSFile: BodyPartType = 4;**

A **multinationalNote** is composed of characters encoded according to the Xerox Character Encoding Standard. A **multinationalNote** can have a maximum of 3777777777B characters.

**multinationalNote: BodyPartType = 5;**

An **ia5Note** is composed of characters encoded according to the International Alphabet number 5 [IA5]. This alphabet is very similar to ASCII. An **ia5Note** can have a maximum of 37777777778 characters. Note that every carriage return must be followed by a line feed according to the description of IA5.

**ia5Note: BodyPartType = 6;**

A **pilotFile** represents the file types used in the Xerox Development Environment.

**pilotFile: BodyPartType = 7;**

A **g3Fax** document is encoded as defined in X.420.

**g3Fax: BodyPartType = 8;**

A **teletex** document is encoded as defined in X.420.

**teletex: BodyPartType = 9;**

A **telex** document is encoded as defined in F.60 [Telex].

**telex: BodyPartType = 10;**

An **iso6937Note** is encoded in X.409 form according to the CEN/CENELEC implementation agreements [CEN/CENELEC]. It contains characters encoded according to the ISO 6937 character encoding standard [ISO 6937]. No conventions for switching between character sets have yet been adopted.

**iso6937Note: BodyPartType = 11;**

An **interpress** body part is encoded according to the Xerox Interpress Electronic Printing Standard [Interpress]

**interpress: BodyPartType = 12;**

This chapter discusses subsets of the MailFormat standard. User agents need not implement the entire MailFormat protocol to provide a useful service. In particular, the user agent need not generate all of the attributes given in 11.3. In addition, support for all of the message body parts given in 11.5 is not required.

---

## 12.1 Subset for the simple client

---

All clients must have the ability to convey any attribute type and value listed in 11.3 to the user. The attributes that must be generated to provide the minimum level of functionality are given in Figure 12.1. It is recommended that the User Agent be able to generate more attributes than those in the minimal set. The set of attributes that are frequently requested by users is also given in Figure 12.1.

The only message body parts that must be supported by all clients are the **ia5Note** and the **multinationalNote**. It is strongly recommended that clients also provide support for the **nsTextFile**.

---

## 12.2 Subset for the X.400 client

---

The X.400 client, just as any other client, must have the ability to convey any attribute type and value listed in 11.3 to the user. The attributes that must be generated by the X.400 client are given in Figure 12.1.

The X.400 client must support the **ia5Note**, the **multinationalNote**, and the **iso6937Note**.

| ATTRIBUTE           | REQUIRED<br>FOR SIMPLE<br>CLIENT | REQUIRED<br>FOR X.400<br>CLIENT | POPULAR<br>FEATURE |
|---------------------|----------------------------------|---------------------------------|--------------------|
| authorizingUsers    | N                                | N                               | Y                  |
| autoforwarded       | N                                | N                               | N                  |
| blindCopyRecipients | N                                | N                               | N                  |
| copyRecipients      | N                                | N                               | Y                  |
| crossReferences     | N                                | N                               | N                  |
| expiryDate          | N                                | N                               | Y                  |
| forwardedHeadings   | Y                                | Y                               | Y                  |
| immutable           | N                                | Y                               | N                  |
| importance          | N                                | N                               | Y                  |
| inReplyTo           | Y                                | Y                               | Y                  |
| messageID           | Y                                | Y                               | Y                  |
| obsoletes           | N                                | N                               | N                  |
| originator          | N                                | N                               | N                  |
| primaryRecipients   | Y                                | Y                               | Y                  |
| replyBy             | N                                | N                               | N                  |
| replyRequest        | N                                | N                               | N                  |
| replyToUsers        | Y                                | Y                               | Y                  |
| sensitivity         | N                                | N                               | Y                  |
| subject             | Y                                | Y                               | Y                  |
| text Annotation     | N                                | N                               | N                  |

Figure 12.1 Generation of attributes by the User Agent

The Inbasket protocol is an important enhancement to the basic functionality provided by the Mail Service via the MailTransport protocol. It is intended to enhance the functionality of the system when used for the transport of interpersonal messages; specifically it allows a given user to use a number of different user agents to access incoming mail by providing a medium-term repository for delivered mail that resides within the Mail Service and can be accessed by multiple clients.

---

### 13.1 Inbaskets and delivery slots

---

The basic paradigm of the inbasket is one in which the lowest level component of the user agent resides in the Mail Service, constantly polling the delivery slot and immediately taking delivery of any messages that arrive and storing them in an associated inbasket in the order in which they were delivered. The other components of the user agent reside on client machines and can access the inbasket to inspect the messages held there. Unlike the delivery slot, the inbasket does not require that each message be removed from the server before the client can inspect its contents. Instead, a set of random-access operations is provided for selective listing, retrieval and deletion of messages in the inbasket.

In most cases, the sharing of a given inbasket among multiple clients will tend to happen sequentially in time, but this is not required. The session facility described below allows multiple clients to access the same inbasket concurrently.

Inbaskets are associated with a specific recipient name in the same way as delivery slots. More information on this mechanism can be found in Section 3.4.1. and 14.1.2. A user can have either an inbasket or a delivery slot, but not both.

---

### 13.2 Message identification

---

Each message in an inbasket is identified by its numeric *index*, which is a unique identifier for the message in the inbasket. Indices are permanently assigned to messages, that is, the index of a message does not change across sessions. Indices are assigned as a function of time. The Mail Service can determine whether one message was put into the inbasket before another message by examining the indices of the messages.

A contiguous set of messages is identified by a *range* of indices in operations that apply to multiple messages.

### 13.3 Message status

---

The message status indicates whether the user has been made aware of the presence of the message, as mediated by the user agent. Since the high-level components of the user agent that effect reception reside in the client, the inbasket facility can provide only a supporting mechanism to record the status of each message. The status must be kept up to date by the client. The status is designed to reflect the normal progress of a message through two stages that occur following delivery:

- 1) The message is new: the user has not been made aware of its presence specifically. Typically, the user agent will provide some global signal alerting the user to the presence of one or more new messages in the inbasket.
- 2) The message is known: the user has been presented with a summary description of the message, often in the context of a larger listing of some or all of the messages in the inbasket. Typically, the user agent will include in this summary description some indication that the message has not yet been viewed in full.

It is important to note that the status of a message represents the extent to which *the user* has been made aware of the message. In particular, the transfer of partial or complete information about the message from the Mail Service to the client of the Inbasket protocol represents an internal event within the user agent and does *not* constitute a change in the message's logical status.

A user defined status is also provided for sophisticated user agents. The user defined status is not interpreted by the Mail Service. One application that might make use of this status is a program that notes messages of special interest to the user (e.g. all messages with high importance).

---

### 13.4 Body part status

---

The status of a body part serves as an indication of the actual reception of the body part, as mediated by the user agent. Only one status is defined for each body part, the *deletable* status. This status indicates whether the user has a need to keep the body part at the Mail Service. The inbasket facility provides a supporting mechanism to record whether a body part is *deletable*. A body part has this status in two different situations. Either the body part has been retrieved and stored by the user agent, or the user wishes to discard the body part. When all of the body parts of a message, including the heading body part, are marked as deletable, the message will be discarded by the Mail Service. Until all of the body parts of a message are marked deletable, any of the body parts may be accessed, including those that are marked as deletable.

The status of a body part represents the extent to which the message has been received *by the user*. In particular, the transfer of partial or complete information about the body part from the Mail Service to the user agent represents an internal event within the user agent, and does not constitute a change in the status of the body part.

---

## 13.5 MailTransport envelopes

---

As defined in the MailTransport protocol, a message consists of an envelope and its contents. The envelope of a delivered message is visible to the client, and represents immutable information regarding the pre-delivery history of the message. Operations for retrieving the MailTransport envelope are provided to the client via the Inbasket protocol.

---

## 13.6 Sessions

---

A client of the Inbasket protocol interacts with the inbasket via the various protocol operations. Most of these operations take place within the context of a *session*. An Inbasket session is established to access a specific inbasket, as identified by the name of the associated mail recipient. Multiple concurrent sessions are supported, but the operations within a given session must be invoked sequentially.

A session is identified by a session handle supplied as a parameter to each operation that takes place within the session. The session handle includes the necessary authentication information to validate the identity of the client on a continuing basis. The session handle loses its validity at the end of the session.

As new messages arrive in the inbasket, they are immediately visible to the session client. The range of valid indices can thus expand at any time during the course of the session. As messages are deleted from the inbasket, they are immediately invisible to the session client. In the case where the deletion was performed by another concurrent session, the client will notice that a previously valid index is no longer valid. In this case, the client should assume that the message represented by the index was deleted by another client.

---

## 13.7 Client good-citizenship considerations

---

By providing a server-resident facility for storage and examination of delivered messages, the Inbasket protocol raises the issue of appropriate usage of its facilities by clients. The loads imposed on the Mail Service by clients fall into two general categories: storage load and access load. Either of these loads can become excessive if clients do not exhibit "good citizenship".

*Storage load* is imposed on the Mail Service by its willingness to retain messages until they are voluntarily removed by the client, generally at the behest of the associated human user. It is important that clients provide facilities that encourage users to dispose of their messages promptly and not allow mail to accumulate in their inbaskets. In particular, it is important that clients provide convenient provisions for message filing to encourage users to save messages in storage facilities other than their inbaskets. It is not unreasonable for clients to be somewhat more aggressive in cases that constitute clear user abuse of the inbasket facility; for example, once the inbasket reaches a certain number of messages, the client might refuse to display any further new messages until some of the older messages have been disposed of.

*Access load* is imposed on the Mail System by virtue of client access to the Mail Service. In particular, the retention of inactive Inbasket sessions for long periods of time can result in a waste of Mail Service resources and inability of some users to access their mail. It is best if clients do not directly tie the initiation and termination of sessions to user controlled actions, (e.g. the opening of a mail-display window) since this can lead to the problems just described. A better strategy is the retention of a session for a generous but limited period of user inactivity, followed by termination of the session in a way that may or may not be explicitly visible to the user.

Observance of these guidelines by Inbasket clients can enable the Mail Service to provide the advantages of server-resident inbaskets while supporting the same level of user population that could be handled using the more economical delivery slot model.

Each Mail Service implements the set of Inbasket operations as a Courier [Courier] remote program. Each procedure description includes a declaration of the procedure in Courier's standard notation and a description of the procedure's arguments and results. Several types and constants are defined in the Inbasket program.

The following definition gives the program and version numbers of the Inbasket protocol and lists all other Courier-based protocols which are referenced from this program.

```
Inbasket: PROGRAM 18 VERSION 2 =
BEGIN
  DEPENDS UPON
    Authentication (14) VERSION 2,
    BulkData (0) VERSION 1,
    Clearinghouse (2) VERSION 3,
    MailFormat (1516) VERSION 3,
    MailTransport (17) VERSION 5,
    MailTransportEnvelopeFormat (1517) VERSION 1;
  ...
END.
```

This indicates that Inbasket is program number 18. This document defines version 2. Inbasket references some types and constants that are defined in other protocols as shown in the above declaration. These are documented in the Authentication Protocol [Authentication], the Bulk Data Transfer Protocol [Bulk Data], the Clearinghouse Protocol [Clearinghouse], and the MailTransport and MailFormat protocols (sections 3-6 and 10-12 of this document).

---

## 14.1 Frequently used data types

---

This section discusses data types that are used throughout the Inbasket protocol.

### 14.1.1 Authentication

---

Several Inbasket remote procedures require the client to provide data that unequivocally demonstrates the client's right to access or change the specified information. The qualifying information comprises two items, *credentials* and a *verifier*, as defined in the Authentication Protocol [Authentication]. The **Credentials** need not be strong, and the names used in them should be those of the user and the Mail Service.

```
Credentials: TYPE = Authentication.Credentials;
Verifier: TYPE = Authentication.Verifier;
```

### 14.1.2 Naming of inbaskets

---

When a message recipient has an inbasket, the name of the inbasket is the same as the name of the message recipient, which in the Xerox Mail System is identified by a 3-part name as defined in the Clearinghouse protocol [Clearinghouse]:

**Name:** TYPE = Clearinghouse.Name;

### 14.1.3 Message indices and ranges

---

A message within an inbasket is identified by a unique **Index** which is permanently assigned to the message. Indices are positive integers allocated from a 32-bit field. The value of zero is reserved to represent the **nullIndex**. The type **Range** is defined to allow specification of the sequence of messages having indices between **low** and **high** (inclusive). The distinguished value **nullIndex** is provided to allow specification of ranges that are open-ended, for example, **Range>nullIndex, 5** indicates all of the messages from the beginning of the inbasket through the message with an index of 5, and **Range>nullIndex, nullIndex** denotes the entire contents of the inbasket.

**nullIndex:** Index = 0;

**Index:** TYPE = LONG CARDINAL;

**Range:** TYPE = RECORD [low, high: Index];

### 14.1.4 Inbasket anchor

---

On rare occasions, the Mail Service has to change the association of an index with a message. For example, new associations might be made when an inbasket is moved from one Mail Service to another. Clients must be concerned about this when they remember message indices across sessions. Clients can use the **Anchor** to determine when the messages in an inbasket are no longer associated with the cached indices. The anchor is obtained when a session is initiated. The value of the anchor is only changed after events that invalidate indices cached by clients.

**Anchor:** TYPE = ARRAY 5 OF UNSPECIFIED;

### 14.1.5 Inbasket state

---

**State** describes the makeup of messages in an inbasket. **new** indicates the number of messages in the inbasket with **status** = **new** (see below). **total** indicates the total number of messages in the inbasket.

**State:** TYPE = RECORD [

**new:** CARDINAL,

**total:** CARDINAL];

### 14.1.6 Inbasket sessions

---

An inbasket session is represented by a session handle of type **Session**:

**Session: TYPE = RECORD [token: ARRAY 2 OF UNSPECIFIED, verifier: Verifier];**

The session handle contains a token that should be used unchanged in the session handles passed to all operations within the same session. The verifier, however, should be updated for each call, as described in the Authentication protocol [Authentication].

### 14.1.7 Message body parts

---

A message is composed of a sequence of body parts, and **BodyPartSequence** represents any subset of the body parts of a message. **allBodyParts** can be used to refer to all of the body parts of a message. See Section 8.1 for a description of **BodyPartIndex**.

**BodyPartSequence: TYPE =**  
**SEQUENCE 500 OF MailTransportEnvelopeFormat.BodyPartIndex;**

**allBodyParts: BodyPartSequence = [];**

### 14.1.8 Status

---

Some of the status information associated with a message applies to the message as a whole, and some applies to individual body parts of a message.

The **MessageStatus** applies to the message as a whole, and indicates one of two situations:

The message is **new**: The user has not yet been made aware of the delivery of this specific message.

The message is **known**: The user has been apprised of the delivery of this specific message, but has not yet received the message content.

The **userDefinedStatus** is uninterpreted by the Mail Service, and is intended to allow user agents to note messages of special interest to the user. Note that this information does not go from sending user agent to receiving user agent. It is set and inspected only by the receiving user agent.

**MessageStatus: TYPE = RECORD [**  
**userDefinedStatus: CARDINAL,**  
**existenceOfMessage {new(0), known(1)}];**

The **BodyPartsStatus** represents the status information for all of the body parts of a message. Only one status is defined for each body part, the **deletable** status. This status indicates whether the user has a need to keep the body part at the Mail Service. A body part with a **BodyPartIndex** of **x** is **deletable** if **BodyPartsStatus[x] = TRUE**. (See Section 8.1 for a description of **BodyPartIndex**.)

**BodyPartsStatus: TYPE = SEQUENCE 500 OF BOOLEAN;**

The **Status** of a message is the collection of the **MessageStatus** and the **BodyPartsStatus**. It is important to note that the **Status** serves as an indication of the reception of the message by

the *user*, as mediated by the Mail Service client. It does *not* reflect the delivery of the message to the client by the message transfer service nor the transfer of the information to the Inbasket client. The appropriate updating of the status is thus entirely the responsibility of the Mail Service client, and is not performed by the Mail Service itself. When a message is delivered to an inbasket, its status is initially **new**. When the user has been alerted to the presence of the message (e.g. by a brief display of selected message information), the client should update the **MessageStatus**. Similarly, when the user has received a body part (e.g., by storing the content) or when the user wants to discard the body part, the status of the body part should be updated to **deletable**. Typically, a client will use various means to alert the user to the presence of messages that are **new** or **known**, but not yet completely processed.

```
Status: TYPE = RECORD [  
    messageStatus: MessageStatus,  
    bodyPartsStatus: BodyPartsStatus] ;
```

---

## 14.2 Initiating and terminating sessions

---

Most Inbasket operations take place in the context of a session. Each session references a single inbasket specified when the session is initiated. When a message recipient has an inbasket, the name of the inbasket is the same as the name of the message recipient. This recipient may or may not be the user whose credentials are used to authorize the session. It is possible for two or more sessions to access the same inbasket at any given time (e.g. from two separate workstations). If multiple concurrent sessions access the same inbasket, each is cognizant of modifications made by other concurrent sessions. If a client notices that an index that was valid at the beginning of a session has become invalid while the session was open, the client should assume that the message represented by the index was deleted by another client.

---

### 14.2.1 Logon

---

**Logon** initiates a new Inbasket session with the Mail Service.

```
Logon: PROCEDURE [inbasket: Name, credentials: Credentials, verifier: Verifier]  
RETURNS [session: Session, state: State, anchor: Anchor]  
REPORTS [AccessError, AuthenticationError, InbasketInUse, OtherError, ServiceError] = 5;
```

Arguments: **credentials** and **verifier** are described in 14.1.1. The client cannot switch from strong to simple authentication or vice versa within a session. **inbasket** identifies the mail recipient whose inbasket will be accessed during the session (which is often the user identified in the credentials).

Results: **session** is a session handle. The **token** is an identifying bit string whose content is unspecified. It should be passed unchanged by the client from one operation to the next within this session. The **verifier** can be used to validate the identity of the Mail Service, according to the specification for returned verifiers in the Authentication protocol [Authentication]. **state** indicates the number of messages in the inbasket and the number of messages that are new. The **anchor** is used to determine whether any indices previously cached by the client have become invalid. If the anchor returned by **Logon** is different from

the anchor returned by the previous call to **Logon**, then all indices previously cached by the client are invalid.

### 14.2.2 Logoff

---

**Logoff** terminates an Inbasket session. The session handle is henceforth invalid.

**Logoff:** PROCEDURE [session: Session]  
REPORTS [AuthenticationError, OtherError, SessionError] = 4;

Arguments: **session** is a session handle.

---

## 14.3 Checking inbasket state

---

### 14.3.1 Mail poll

---

**MailPoll** allows a client to check the state of an inbasket without incurring the overhead of initiating a session. Based on the result returned, the client may or may not wish to initiate a session and examine the contents of the inbasket in more detail, using the operations in Section 14.4.

**MailPoll:** PROCEDURE [inbasket: Name, credentials: Credentials, verifier: Verifier]  
RETURNS [state: State]  
REPORTS [AccessError, AuthenticationError, OtherError, ServiceError] = 7;

Arguments: **credentials** and **verifier** are described in 14.1.1. **inbasket** identifies the mail recipient whose inbasket state is to be checked.

Results: **state** indicates the number of messages in the inbasket and the number of messages that are new. The numbers that are returned do not take into account status changes that have not yet been made permanent.

NOTE: The **MailPoll** operation can be expedited (see Section 17 for details on expediting calls).

### 14.3.2 Mail check

---

**MailCheck** provides the same function as **MailPoll** except that it is designed to be used within a session. This operation is provided to allow the client to make use of the authentication that was done to initiate the session. That is, the client need not start the authentication process anew to find out whether mail has arrived in the inbasket.

**MailCheck:** PROCEDURE [session: Session]  
RETURNS [state: State]  
REPORTS [AuthenticationError, OtherError, SessionError, ServiceError] = 6;

Arguments: **session** is a session handle.

Results: **state** indicates the number of messages in the inbasket and the number of messages that are new. The numbers returned do not take into account status changes that have not yet been made permanent.

NOTE: The **MailCheck** operation can be expedited (See Section 17 for details expediting calls).

---

## 14.4 Examining inbasket contents

---

### 14.4.1 Retrieve envelopes

---

**RetrieveEnvelopes** is used either to retrieve the envelope of a particular message, or to sequentially retrieve the envelopes of several messages in the inbasket. In order to retrieve a particular envelope, **whichMsg** is set to **this**. If there is no message in the inbasket corresponding to the specified index, **IndexError** is raised. Envelopes are enumerated by specifying an **index** after which the enumeration should start, and repeatedly calling **RetrieveEnvelopes** with **whichMsg** set to **next**, and **index** set to the value of **index** returned by the previous call. An **index** of **nullIndex** with **whichMsg** set to **next** is used to begin an enumeration of all of the messages in the inbasket. The **index** of a message that the user agent has previously seen can be used to start the enumeration at the first message that the User Agent has not seen. The **nullIndex** is returned when there are no more envelopes to enumerate. The returned **index** corresponds to the returned envelope.

**RetrieveEnvelopes:** PROCEDURE [**index:** Index, **whichMsg:** {**this**(0), **next**(1)},  
    **session:** Session]

**RETURNS** [**transportEnvelope:** MailTransport.Envelope, **status:** Status, **index:** Index]

**REPORTS** [**AuthenticationError**, **IndexError**, **OtherError**, **SessionError**,  
    **ServiceError**] = 2;

Arguments: **session** is a session handle. **index** in conjunction with **whichMsg** determines which message the procedure will operate on. **this** indicates that the procedure will operate on the message designated by **index**, while **next** indicates that the procedure will operate on the next message in the inbasket after the message designated by **index**. The messages in the inbasket can be enumerated by calling this procedure repeatedly, each time asking for the **next** message. If **this** is specified, and no message with an index of **index** exists, an error will be raised.

Results: **transportEnvelope** is the MailTransport envelope which represents information regarding the pre-delivery history of the message. **status** shows the MessageStatus and BodyPartStatus of the message. The returned **index** corresponds to the returned envelope. When messages are enumerated, the **nullIndex** is returned when there are no more messages in the inbasket.

---

### 14.4.2 Retrieve body parts

---

**RetrieveBodyParts** copies the contents of the specified body parts from the inbasket to the client.

**RetrieveBodyParts:** PROCEDURE [**message:** Index, **bodyParts:** BodyPartSequence,  
    **contents:** BulkData.Sink, **session:** Session]

**REPORTS** [AuthenticationError, IndexError, OtherError, SessionError, ServiceError, TransferError] = 8;

**Arguments:** **session** is a session handle. **message** identifies the message that body parts are to be retrieved from. **bodyParts** specifies the body parts to be retrieved. The constant **allBodyParts** may be used to specify all of the body parts of the message. Body parts will be returned in the order specified in **BodyPartSequence**. That is, the first body part returned will be that identified by **bodyParts[0]**, and so on. The body parts are concatenated in **contents** without any structuring information between them, and the client can determine where each body part begins by using the body part sizes given in the **tableOfContents**.

---

## 14.5 Modifying inbasket contents

---

### 14.5.1 Change message status

---

**ChangeMessageStatus** updates the status of a **Range** of messages to **known**. Typically, a message is delivered, advances from **new** to **known**, and is later deleted from the inbasket. In addition to marking the range of messages **known**, this procedure can be used to update the **userDefinedStatus**. Note that a message cannot be made **new** again. Repeated calls to **ChangeMessageStatus** with **changeUserDefinedStatus** set to **FALSE** have no effect once the message has been made **known**.

**ChangeMessageStatus:** PROCEDURE [range: Range, changeUserDefinedStatus: BOOLEAN, newUserDefinedStatus: CARDINAL, session: Session]  
**REPORTS** [AuthenticationError, IndexError, OtherError, SessionError, ServiceError] = 0;

**Arguments:** **session** is a session handle. **range** specifies a set of messages. **changeUserDefinedStatus** indicates whether the **userDefinedStatus** should be changed, and if so, **newUserDefinedStatus** is the new value for user defined status.

### 14.5.2 Change body parts status

---

**ChangeBodyPartsStatus** updates the status of one or more body parts of a message. When all of the body parts of a message have been made **deletable**, the message will be deleted by the Mail Service. Therefore, if the client wants the data in the body part, it is important to store the data before the status change to **deletable** takes effect. Note that when a body part status changes to **deletable** this change is irreversible. Note also that all of the body parts of a message are accessible until the entire message is deleted. This procedure also has the effect of updating the **MessageStatus** to **known**.

**BodyPartStatusChangeSequence:** TYPE = SEQUENCE 500 OF **BodyPartStatusChange**;  
**BodyPartStatusChange:** TYPE = RECORD [  
    **bodyPartIndex:** MailTransportEnvelopeFormat.BodyPartIndex,  
    **deletable:** {true(0), noChange(1)}];  
**ChangeBodyPartsStatus:** PROCEDURE [  
    **index:** Index, **setStatusTo:** BodyPartStatusChangeSequence, **session:** Session]  
**RETURNS** [deleted: BOOLEAN]  
**REPORTS** [AuthenticationError, IndexError, OtherError, SessionError, ServiceError] = 3;

Arguments: **session** is a session handle. **index** identifies the message that will be updated, **setStatusTo** identifies the body parts which will be updated.

Results: **deleted** indicates if all of the body parts of a message have been marked as deletable.

---

### 14.5.3 Delete

---

**Delete** deletes one or more messages from an inbasket. If there are no messages in the **range**, no error is raised.

**Delete:** PROCEDURE [**range:** Range, **session:** Session]  
**REPORTS** [AuthenticationError, OtherError, ServiceError] = 1;

Arguments: **session** is a session handle. **range** specifies a set of messages.

---

## 14.6 Get size

---

**GetSize** reports the amount of space occupied by all of the messages in an inbasket in bytes. This could be used to warn users when their inbasket is taking up too much space on the Mail Service.

**GetSize:** PROCEDURE [**inbasket:** Name, **credentials:** Credentials, **verifier:** Verifier]  
**RETURNS** [sizeInBytes: LONG CARDINAL]  
**REPORTS** [AuthenticationError, AccessError, OtherError, ServiceError] = 10;

Arguments: **inbasket** names the inbasket for which the size will be obtained. **credentials** and **verifier** are described in 14.1.1.

Results: **sizeInBytes** is the size of the inbasket.

---

## 14.7 Bulk data formats

---

Bulk data is used as defined in the Addenda to Standards, XSIS 138301. That is, the transfer simply consists of a series of packets, the last of which has end-of-message set to TRUE (if the transfer was successful). No additional structuring of the data is needed. The body parts are concatenated in **contents** without any structuring information between them, and the client can determine where each body part begins by using the body part sizes given in the **tableOfContents**

When a remote procedure completes successfully, it returns results as specified in the definition of the procedure. However, conditions can arise before or during execution of the procedure that make successful completion of the request impossible. For example, the client may have specified incorrect arguments in a remote procedure call, or some required resource may be unavailable.

When such conditions occur, an error is reported to communicate to the client the nature of the problem. Each error encompasses an entire class of possible conditions and the specific problem is further described by the arguments of the error. For example, **AccessError** indicates that the Mail Service rejected the requested operation because of some sort of access problem. The particular problem is specified by the argument to **AccessError**, which is of type **AccessProblem**.

When an exceptional condition arises during execution of a remote procedure, the Mail Service makes every effort to undo the effects of the partial execution so that it appears to the client as though the procedure had never been called. However, the Mail Service does not guarantee that such effects can always be reversed. Therefore, when an error is reported, the client must be prepared for the possibility that the procedure was partially executed.

All **Inbasket** errors are defined below. Each error definition includes a declaration of the error in Courier notation, and a description of its arguments. For Remote errors generated by the MailTransport protocol, see Section 5.

---

## 15.1 Authentication errors

---

A Mail Service may refuse to perform an Inbasket operation if the client's credentials are invalid or too weak. The refusal is reported by the error:

**AuthenticationError: ERROR [problem: Authentication.AuthenticationProblem] = 1;**

The reader should consult [Authentication] for the values and meanings of the various authentication problem codes.

## 15.2 Access errors

---

Even if the caller's credentials are valid, the caller may not be allowed to invoke the requested operation, due either to lack of access privileges or to the state of the service at the time the access is requested. This situation is reported by the error:

**AccessError: ERROR [problem: AccessProblem] = 0;**

The nature of the problem is indicated by the value of **problem**:

**AccessProblem: TYPE = {**  
    **accessRightsInsufficient(0),**  
    **accessRightsIndeterminate(1),**  
    **noSuchInbasket(2),**  
    **inbasketIndeterminate(3),**  
    **wrongService(4)};**

**accessRightsInsufficient** indicates that the caller did not have sufficient access rights to perform the operation in question such as access rights to a mailbox.

**accessRightsIndeterminate** indicates that it was not possible to determine whether the caller had sufficient access rights (e.g., some other service that was needed to make the determination was unavailable).

**noSuchInbasket** indicates that the Mail Service determined that the specified recipient name was invalid. For example, this problem code is used to indicate that an inbasket was specified that did not correspond to a valid recipient.

**inbasketIndeterminate** indicates that the validity of the specified recipient name could not be determined (e.g., some other service that was needed in making the determination was unavailable).

**wrongService** indicates that the specified inbasket does not reside on this service.

---

## 15.3 Session errors

---

An operation may be rejected because of incorrect session-related data. The rejection is reported by the error:

**SessionError: ERROR [problem: SessionProblem] = 5;**

**SessionProblem: TYPE = {**  
    **tokenInvalid(0)};**

**tokenInvalid** indicates that the session handle presented contained an invalid token.

## 15.4 Service errors

---

An operation may be rejected because of a lack of service resources or a server crash. The rejection is reported by the error:

```
ServiceError: ERROR [problem: ServiceProblem] = 6;  
ServiceProblem: TYPE = {  
    cannotAuthenticate(0),  
    serviceFull(1),  
    serviceUnavailable(2)};
```

**cannotAuthenticate** indicates that necessary authentication could not be performed (e.g. because other services such as an Authentication Service needed to perform the authentication were not available).

**serviceFull** indicates that the service (not the server) has reached its maximum number of active clients and has rejected the requested operation.

**serviceUnavailable** indicates that the service is not currently available to network clients, even though the server is running.

---

## 15.5 Transfer errors

---

An operation may fail due to failures associated with bulk data transfer. The failure is reported by the error:

```
TransferError: ERROR [problem: TransferProblem] = 7;  
TransferProblem: TYPE = {  
    aborted(0)};
```

**aborted** indicates that the bulk data transfer was initiated but was subsequently aborted by one end (the source or the sink).

---

## 15.6 Index errors

---

An operation may be rejected because a specified index does not identify a message or body part. The rejection is reported by the error:

```
IndexError: ERROR [problem: IndexProblem] = 9;  
IndexProblem: TYPE = {  
    invalidIndex(0),  
    invalidBodyPartIndex(1)};
```

**invalidIndex** indicates that the specified index does not identify a message.

**invalidBodyPartIndex** indicates that the specified index does not identify a body part.

## 15.7 Inbasket in use

---

Although version 2 of the Inbasket protocol allows clients to access the inbasket concurrently, version 1 allowed the client to request exclusive access to the inbasket. Consequently, as long as version 1 of the Inbasket protocol is supported, the Mail Service will raise **InbasketInUse** when a version 1 Mail Service client has exclusive access to the inbasket at the time a version 2 call is made.

**InbasketInUse: ERROR [user: Name] = 10;**

---

## 15.8 Other errors

---

In unusual cases, an operation may be rejected because of an error other than one listed above. The rejection is reported by the error:

**OtherError: ERROR [problem: OtherProblem] = 8;**

**OtherProblem: TYPE = {**  
    **cantExpedite(0),**  
    **malformedMessage(1),**  
    **invalidOperation(2),**  
    **last(65535)};**

**cantExpedite** indicates that an expedited call was made to a non-expedited procedure, or that an expedited call was made to a service which doesn't support expedited calls.

**malformedMessage** indicates that the number of body parts specified in the table of contents does not match the number of body parts in the message or that the data is not formatted according to the BulkData protocol [Bulk Data].

**invalidOperation** indicates that the sequence of operations is invalid.

**last** exists so that new servers and procedures can be developed in a staged fashion; this value should never be reported by a correctly functioning server conforming to this protocol.

It is frequently useful to define subsets of a protocol by defining the minimum that must be done to provide a class of functionality. This chapter defines two classes of functionality: the class needed by Inbasket clients, and the class supported by the Inbasket service provider.

---

## 16.1 Subsets for the client

---

The client can use the inbasket in a number of different ways, many of which will be unique to a particular application. Consequently, the minimum that must be supported depends on the application. The following is an indication of what different clients might implement.

All clients must implement the **Logon**, **Logoff**, **RetrieveEnvelopes**, and **Delete** procedures. Clients need not implement the **GetSize** procedure. Client implementation of the remaining procedures in the Inbasket protocol depends on the type of interaction the client would like to have with the Mail Service. For example, a client that does nothing but list the messages in an inbasket does not need to implement the **RetrieveBodyParts** procedure, but any client that retrieves messages would implement this procedure. As another example, clients need not implement either **MailPoll** or **MailCheck** if the inbasket is inspected infrequently (e.g., once a day). Implementation of both **MailPoll** and **MailCheck** is advised when the client keeps sessions to the Mail Service open for long periods of time (e.g., one hour).

When a client will be sharing inbaskets with other clients, it should be the user who decides which message body parts of a message to retrieve. This can be specified in a variety of ways, for example, upon each mail retrieval, or in a user's profile. In addition, the user should have the option of inspecting a message body part without making it **deletable**.

---

## 16.2 Subset for the service provider

---

Inbasket service providers must implement the entire Inbasket Courier program. It is strongly recommended, but not required, that Expedited Courier be implemented.



The Mailing Protocols utilize the Packet Exchange Protocol defined in the Internet Transport Protocols [Internet] to call some Mail Service remote procedures [Internet]. Remote procedure calls made in this way are called *expedited procedure calls*. An expedited procedure call is frequently more efficient than the corresponding Courier call. Four characteristics of a remote procedure call make the use of such a low-level protocol appropriate:

- the calls are usually confined to the local network;
- the amount of data allows the argument and result lists to be less than a maximum Packet Exchange packet size; and
- minimum delays are important for performance reasons;
- idempotency (the operation may be repeated many times with no ill effects).

Because of the restriction on the amount of data that can be conveyed in a single packet, not all Mail Service operations can be invoked as expedited calls. However, for every operation that can be expedited, the function operates in exactly the same way as the corresponding Courier operation; it produces the same effects, returns the same results, and reports the same errors.

The expedited procedure call mechanism described here is identical to that used by the Clearinghouse [Clearinghouse].

---

## 17.1 Expedited procedure call protocol

---

Not all Mail Service operations are eligible for expediting. Only four expedited operations are allowed: two from MailTransport, (**MailPoll** and **ServerPoll**) and two from Inbasket, (**MailPoll** and **MailCheck**). All other Mail Service operations are ineligible for expediting.

As noted above, expedited procedure calls are accomplished with the Packet Exchange Protocol. There are two kinds of packets involved, *request packets*, whereby the client invokes a Mail Service operation, and *response packets*, by which means the client receives the results of the operation.

The client sends a request packet to the desired mail server using the well-known Mail Service socket number. The client should then wait for a response packet coming from the Mail Service and using the same socket number. If a client wishes to re-transmit the request packet, it should use the same level 2 ID it used on the first request. The Mail Service uses this information to suppress duplicate (re-transmitted) requests. Eventually, the Mail

Service will return a response packet containing the results if the operation was successful, or error information if it was not.

---

## 17.2 Packet formats

---

The following descriptions of packets assume familiarity with the Packet Exchange Protocol packet type defined in the Internet Transport Protocol [Internet]. The formats of the request and response packets are given relative to the level 3 portion of the Packet Exchange packet. It is the intent that the content of the level 3 portion of the packet be the same as the corresponding Courier call and return messages (see Courier [Courier]). The content of the various fields will be described by reference to the Courier document.

---

### 17.2.1 Data in the Packet Exchange header

---

The Destination Socket field of the level 1 portion of the packet contains the value of the Mail Service socket, 26<sub>10</sub>. The Client Type field of the level 2 portion of the packet has a value of 2.

---

### 17.2.2 Request packet

---

The request packet, shown in Figure 17.1, consists of two parts: the *request header*, and the *message body*. The first two fields of the header contain the lowest and highest Courier version numbers with which the client is willing to deal. In particular, the highest version number determines the encoding of the message body. Each is 16 bits long and is encoded identically to a Courier **CARDINAL**. These version numbers are to be used in the same fashion as the version number exchange described in Courier. The third word (word 2) contains the Courier Message type; for a request packet, it has the value 0 encoded identically to a Courier **CARDINAL**.

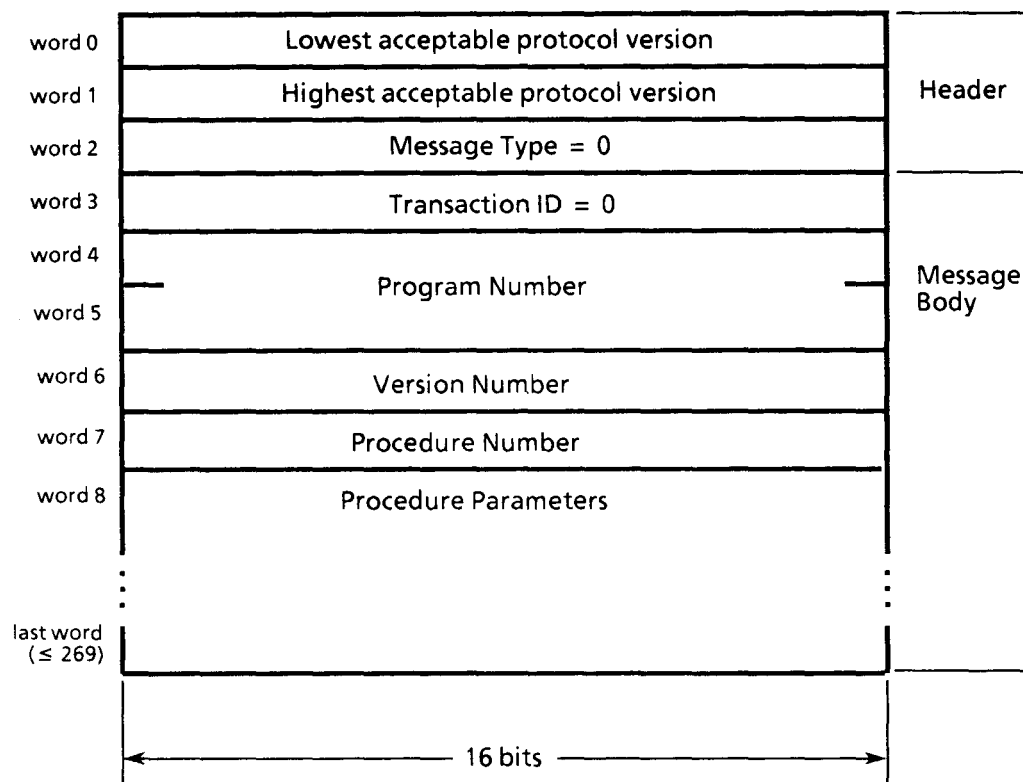
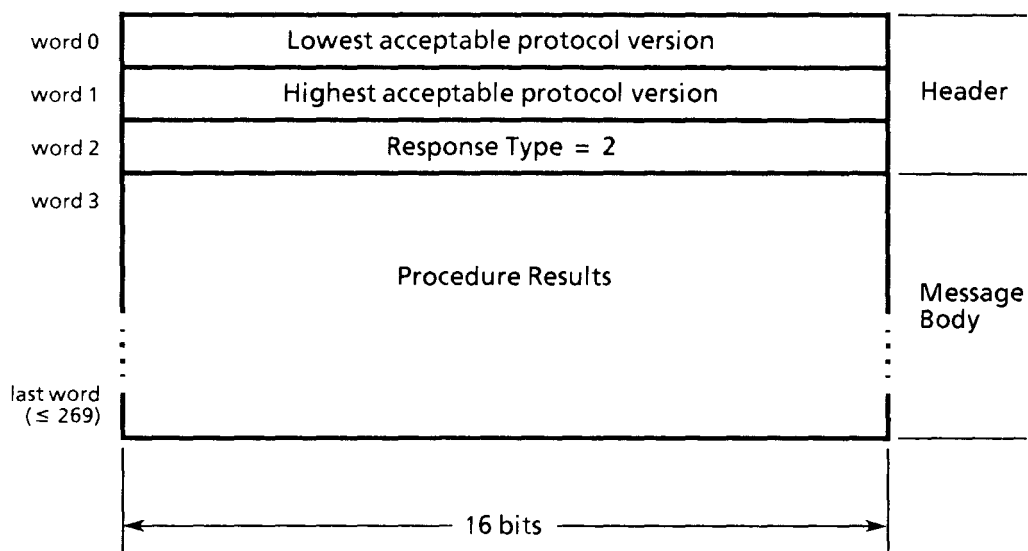
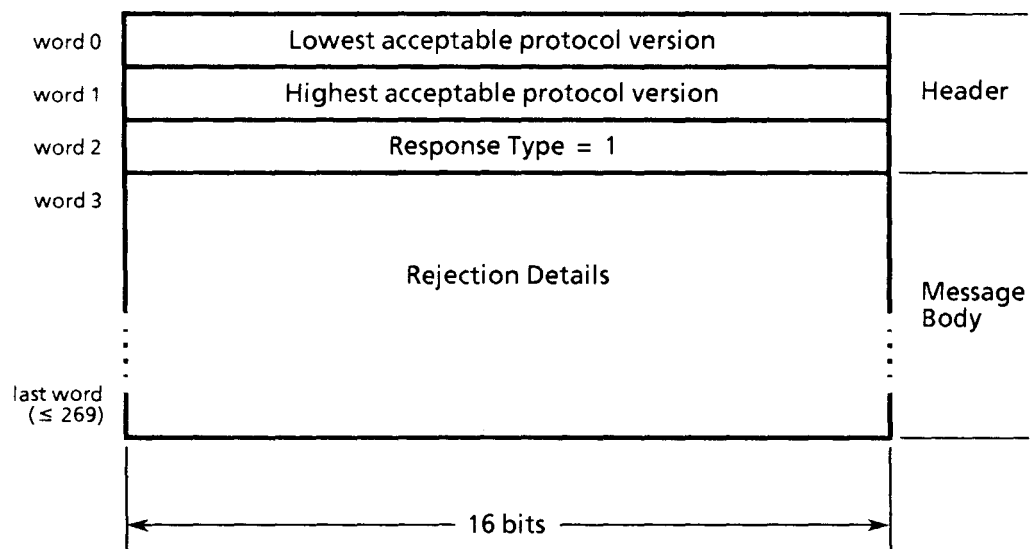


Figure 17.1 Request packet format

The rest of the packet is the message body. It is encoded the same as the Courier **CallMessageBody**. In the message body, the **transactionID** should have the value 0. For MailTransport operations, the **programNumber** and **versionNumber** should have the values 17 and 5, respectively, and the **procedureValue** field (i.e., the procedure number that is being called) must be either 0 or 7. For Inbasket operations, the **programNumber** and **versionNumber** should have the values 18 and 2, respectively, and the **procedureValue** field must be either 6 or 7.

### 17.2.3 Response packet

The response packets shown in Figures 17.2, 17.3 and 17.4 also consist of two parts, the *response header* and the *message body*. The response header consists of three fields, the highest and lowest acceptable Courier version numbers at the server, and the response message type corresponding to and having the same meaning as the Courier Message type. All three fields are encoded the same as a Courier **CARDINAL**. A response type of 1 indicates that the call has been rejected. A response type of 2 indicates that the call has succeeded. A response type of 3 indicates the procedure is reporting an error.

Figure 17.2 Response packet format for `ResponseType = return`Figure 17.3 Response packet format for `ResponseType = reject`

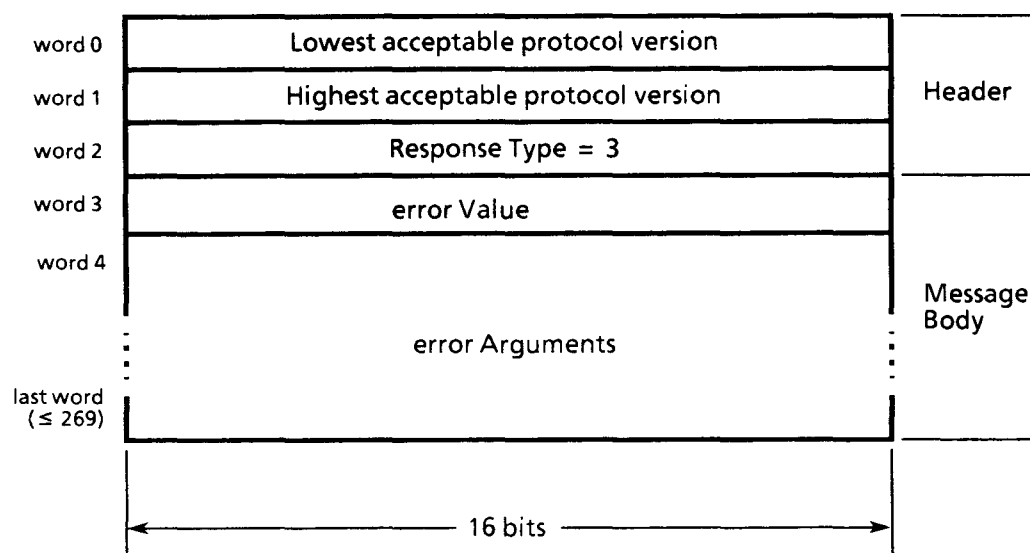


Figure 17.4 Response packet format for ResponseType = abort

The Message Body part of the packet contains the Courier message body that corresponds to the response type. The appropriate message body is encoded the same as it is in Courier.

### 17.3 When expediting fails

An expedited call is not guaranteed to succeed. The Mail Service currently signals the appropriate (**Inbasket** or **MailTransport**) **OtherError[cantExpedite]** when it will not accept an expedited attempt. The Mail Service may report this error for one of the following reasons:

- the particular procedure cannot be expedited (for example, most bulk data operations cannot be expedited);
- a Mail Service may insist at any time that any operation use the Courier protocol (for example, because it does not support expedited calls).

When this error (**OtherError** with problem **cantExpedite**) occurs, the client should invoke the desired operation using Courier.



- [Filing] Xerox Corporation. Filing Protocol. Xerox System Integration Standard. Stamford, Connecticut. October 1982. XSIS 108210.  
This reference defines the serialized file format used by many mail system clients.
- [IA5] International Organization for Standardization. International Alphabet No. 5. ISO 646 (also known as T.50 and V.3). Geneva, Switzerland 1984.  
This reference defines the format of the **ia5**Note body part.
- [Internet] Xerox Corporation. Internet Transport Protocols. Xerox System Integration Standard. Stamford, Connecticut. December 1981. XSIS 028112.  
This reference defines packet formats for layers 1 and 2. These formats are used in conjunction with Expedited Courier.
- [Interpress] Xerox Corporation. Interpress Electronic Printing Standard. Xerox System Integration Standard. Stamford Connecticut. April 1984. XSIS 048404.  
This reference defines the digital representation of material that is to be transmitted to and printed on an electronic printer.
- [ISO 6937] International Organization for Standardization. Information processing - Coded character sets for text communication - Part 2: Latin alphabetic and non-alphabetic graphic characters. Geneva, Switzerland 1983. ISO 6937/2-1983 (E).  
This reference defines the characters that make up the **iso6937** body part.
- [Telex] CCITT (The International Telegraph and Telephone Consultative Committee), Red Book, Volume II, Fascicle II.4. Telegraph Services-Operations and Quality of Service. Recommendations F.1-F.150 Geneva, Switzerland 1985. International Telecommunication Union.  
This reference defines the format of the **telex** body part.
- [Time] Xerox Corporation. Time Protocol. Xerox System Integration Standard. Stamford, Connecticut; October 1982. XSIS 088210.  
This reference defines the Time Standard upon which the MailTransport and Inbasket protocols rely for the definition of the format of time and date quantities.
- [X.400 Implementor's Guide] CCITT - The International Telegraph and Telephone Consultative Committee. X.400-Series Implementor's Guide. Geneva, Switzerland. October 1986.  
This reference defines the pragmatic constraints for X.400 protocol elements.
- [X.400] CCITT - The International Telegraph and Telephone Consultative Committee. Data Communication Networks Message Handling Systems. Geneva, Switzerland 1985. Recommendations X.400-X.430.  
The X.400 recommendations describe a multinational standard for message exchange. This document is made up of eight separate recommendations: X.400, X.401, X.408, X.409, X.410, X.411, X.420, and X.430.

The following documents are referenced by this specification.

- [Authentication] Xerox Corporation. Authentication Protocol. Xerox System Integration Standard. Stamford, Connecticut; October 1982. XSIS 098210.  
This reference defines the Authentication Protocol upon which the MailTransport and Inbasket protocols rely for authentication.
- [Bulk Data] Xerox Corporation. Bulk Data Transfer. Xerox System Integration Standard. Stamford, Connecticut; October 1982. XSIS 038112; Addendum 1. Augments [6].  
This reference defines the Bulk Data Transfer Protocol upon which the MailTransport and Inbasket protocols rely for bulk data transfer.
- [CEN/CENELEC] The Joint European Standards Institution. Private Message Handling System: User Agent and Message Transfer Agent: Private Management Domain to Private Management Domain. Brussels, Belgium. June 1986.  
This reference defines the format of the **iso6937** body part.
- [Char Code] Xerox Corporation. Character Code Standard. Xerox System Integration Standard. Stamford, Connecticut; October 1982. XSIS 058303.  
This reference defines the character set and the string format which provide the basis for Courier's string data type.
- [Clearinghouse] Xerox Corporation. Clearinghouse Protocol. Xerox System Integration Standard. Stamford, Connecticut; April 1984. XSIS 078404.  
This reference defines the protocol for accessing the Clearinghouse directory database.
- [CH Entries] Xerox Corporation. Clearinghouse Entry Formats. Xerox System Integration Standard. Stamford, Connecticut; April 1984. XSIS 168404.  
This reference defines the structure of entries that appear in the Clearinghouse directory database.
- [Courier] Xerox Corporation. Courier: The Remote Procedure Call Protocol. Xerox System Integration Standard. Stamford, Connecticut; 1981 December. XSIS 038112.  
This reference defines the Courier language, in which terms the MailTransport and Inbasket protocols are defined.
- [Ethernet] Digital Equipment Corporation, Intel Corporation, and Xerox Corporation. The Ethernet: A Local Area Network; Data Link Layer and Physical Layer Specifications. Stamford, Connecticut; September 1980.  
This reference defines socket numbers, which are used by Expedited Courier.

The **ContentType**, **EnvelopeItem**, and **BodyPartType** defined in **MailTransport**, and the **AttributeType** defined in **MailFormat** are assigned 32-bit numbers that are unique throughout the distributed system. These number spaces are administered by Xerox Corporation.

The **type** field of the **gatewaySpecificInformation** in an **RName** (see 4.1.2) is also administered by Xerox Corporation. To obtain a block of numbers, or a **type** for **gatewaySpecificInformation**, submit a written request to:

Xerox Corporation  
Document Systems Business Unit  
Network Systems Administration Office  
475 Oakmead Parkway  
Sunnyvale, California 94086

When requesting blocks of numbers for **MailTransport.BodyPartType**, the **MailTransport.ContentType** that the body part types will be associated with should be specified.

Protocol implementors are encouraged to apply for unique blocks of numbers for their particular applications. Uniqueness allows systems to freely interconnect without having to worry about overlapping values for critical fields.

Type numbers should be used with economy as the total number of blocks is limited. If an implementation is using a large quantity of either of these type numbers, the designer has probably misunderstood their use.

The following blocks of **MailFormat.AttributeType** are reserved for use by specialized Xerox client programs. They are not intended for use by all user agents and do not correspond to any X.400 heading components. They are not a part of the standard and are recorded here for bookkeeping purposes only.

**firstInterlispType: AttributeType = 4910;**  
**lastInterlispType: AttributeType = 4919;**  
**firstGWType: AttributeType = 4920;**  
**lastGWType: AttributeType = 4927;**

When requesting a **type** for **gatewaySpecificInformation**, the **value** for the requested type should be described. Note that **value** is limited to 128 bytes, and that type is limited to 8 bytes when it is intended to be sent to the X.400 system as an X.400 Domain Defined Attribute (see Appendix J). Special arrangements need to be made to send such attributes to the X.400 system, because the X.400 mail gateway must translate from **gatewaySpecificInformation** to Domain Defined Attributes. **types** currently in use are:

|                                |                                                                                                              |
|--------------------------------|--------------------------------------------------------------------------------------------------------------|
| <b>type:</b> "X121Address"     | <b>value:</b> a string of digits with maximum length = 15                                                    |
| <b>type:</b> "TerminalID"      | <b>value:</b> a string of ASCII characters with maximum length = 24                                          |
| <b>type:</b> "UAIIdentifier"   | <b>value:</b> a string of digits with maximum length = 32                                                    |
| <b>type:</b> "FreeFormName"    | <b>value:</b> a string of characters from the Xerox Character Code Standard with a maximum length = 64 bytes |
| <b>type:</b> "TelephoneNumber" | <b>value:</b> a string of digits with maximum length = 32                                                    |

The complete declaration of the MailTransport protocol is given below.

```
MailTransport: PROGRAM 17 VERSION 5 =
BEGIN
```

```
    DEPENDS UPON
        Authentication (14) VERSION 2,
        BulkData (0) VERSION 1,
        Clearinghouse (2) VERSION 3;
```

```
-- TYPES AND CONSTANTS --
```

```
Credentials: TYPE = Authentication.Credentials;
ContentsType: TYPE = LONG CARDINAL;
Verifier: TYPE = Authentication.Verifier;
Session: TYPE = RECORD [token: ARRAY 2 OF UNSPECIFIED, verifier: Verifier];
```

```
EnvelopeItem: TYPE = LONG CARDINAL;
EnvelopeItem: TYPE = RECORD [type: EnvelopeItem, value: SEQUENCE OF UNSPECIFIED];
Envelope: TYPE = SEQUENCE 100 OF EnvelopeItem;
-- there can be up to 100 envelope items in an envelope.
```

```
InvalidName: TYPE = RECORD[id: CARDINAL, invalidReason: InvalidReason];
InvalidNameList: TYPE = SEQUENCE OF InvalidName;
InvalidReason: TYPE = {
    noSuchRecipient(0),           -- recipient does not exist
    noMailboxForRecipient(1),
    illegalName(2),               -- specified name is not a valid RName
    noDLsAllowed(3),              -- occurs only when allowDLRecipients = FALSE
    reportNotAllowed(4)};         -- reports are not allowed on DLs
```

```
Name: TYPE = Clearinghouse.Name;
ReportType: TYPE = {none(0), nonDeliveryOnly(1), all(2)};
Recipient: TYPE = RECORD[name: RName, recipientID: CARDINAL, report: ReportType];
RecipientList: TYPE = SEQUENCE OF Recipient;
RName: TYPE = CHOICE OF {
    xns(0) => Name,
    gateway(1) => RECORD[
        country: STRING,
        adminDomain: STRING,
        privateDomain: STRING,
        organization: STRING,
        organizationalUnits: SEQUENCE 20 OF STRING,
        -- there can be up to 20 organizationalUnits in a name.
        personal: CHOICE OF {
            whole(0) => STRING,
            broken(1) => RECORD[given, initials, family, generation: STRING]},
```

```

        gatewaySpecificInformation: SEQUENCE 10 OF Attribute]];
        -- there can be up to 10 pieces of gatewaySpecificInformation in a name
Attribute: TYPE = RECORD[type, value: STRING];

-- The maximum length of the strings in a gateway variant of an RName are given by
-- the X.400 Implementors Guide and are as follows:

-- country: maximum length of 3 bytes
-- adminDomain: maximum length of 16 bytes
-- privateDomain: maximum length of 16 bytes
-- organization: maximum length of 64 bytes
-- organizationUnits: maximum length of each element is 32 bytes
-- personal whole: maximum length of 40 bytes
-- personal given: maximum length of 16 bytes
-- personal initials: maximum length of 5 bytes
-- personal family: maximum length of 40 bytes
-- personal generation: maximum length of 3 bytes

-- The gateway SpecificInformationField lengths are defined in Appendix B.

--The following Attributes are defined to allow the use of non-standard X.400 name forms:
-- type: 'X121Address'    value: a string of digits, maximum length = 15
-- type: 'TerminalID'     value: a string of ASCII characters, maximum length =
--                          24
-- type: 'UAIentifier'    value: a string of digits, maximum length = 32
-- type: 'FreeFormName'   value: a string of characters from the Xerox Character
--                          Code Standard, maximum length = 64 bytes
-- type: 'TelephoneNumber' value: a string of digits, maximum length = 32

Willingness: TYPE = SEQUENCE 12 OF WillingnessMetric;
-- Willingness[i] gives the service's ability to accept a message of size  $8^i$  bytes to
-- size  $8^{(i+1)-1}$  bytes.
-- The last element in the sequence gives willingness from  $8^{(\text{index of last element})}$  bytes out to
-- infinity. This sequence will typically contain fewer than 12 elements.
WillingnessMetric: TYPE = CARDINAL; -- Ranges from 1 (unwilling) to 10 (very willing);

BodyPartType: TYPE = LONG CARDINAL; -- types of constituent message pieces

PostingData: TYPE = RECORD [
    recipients: RecipientList,
    contentsType: ContentsType,
    contentsSize: LONG CARDINAL,
    bodyPartTypesSequence: SEQUENCE 500 OF BodyPartType];

MessageID: TYPE = ARRAY 5 OF UNSPECIFIED;

-- REMOTE PROCEDURES --

-- Posting Slot Procedures --

ServerPoll: PROCEDURE []
RETURNS [willingness: Willingness, address: Clearinghouse.NetworkAddressList,
        serverName: Name] = 0;

```

**BeginPost:** PROCEDURE [postingData: PostingData, postIfInvalidNames: BOOLEAN,  
allowDLRecipients: BOOLEAN, optionalEnvelopeData: SEQUENCE OF EnvelopeItem,  
credentials: Credentials, verifier: Verifier]  
RETURNS [session: Session, invalidNames: InvalidNameList]  
REPORTS [AuthenticationError, InvalidRecipients, OtherError, ServiceError] = 1;  
-- If postIfInvalidNames = FALSE and LENGTH[invalidNames] > 0, then the session  
-- that's returned may not be used to post body parts.

**PostOneBodyPart:** PROCEDURE [session: Session, bodyPartType: BodyPartType,  
contents: BulkData.Source]  
REPORTS [AuthenticationError, OtherError, SessionError, ServiceError,  
TransferError] = 8;  
-- Called once per body part.

**EndPost:** PROCEDURE [session: Session, abortPost: BOOLEAN]  
RETURNS [messageID: MessageID]  
REPORTS [AuthenticationError, OtherError, SessionError, ServiceError,  
TransferError] = 9;  
-- Terminates a session initiated by BeginPost. If abortPost = TRUE, the message is  
-- discarded by the Mail Service.  
  
-- Delivery Slot Procedures --

**BeginRetrieval:** — PROCEDURE [deliverySlot: Name, credentials: Credentials,  
verifier: Verifier]  
RETURNS [session: Session]  
REPORTS [AccessError, AuthenticationError, OtherError, ServiceError] = 2;  
-- Initiates retrieval of one or more messages via the delivery slot.  
-- The returned session is to be used in a series of calls to the operations below.

**RetrieveEnvelope:** PROCEDURE [session: Session]  
RETURNS [empty: BOOLEAN, envelope: Envelope]  
REPORTS [AuthenticationError, OtherError, SessionError, ServiceError] = 5;  
-- Begins retrieval of the message at the head of the delivery slot queue identified  
-- by 'handle'. This operation is legal after BeginRetrieval or RetrieveContents,  
-- AbortRetrieval, or another call to RetrieveEnvelope.  
-- If 'empty' is TRUE, the delivery slot has no messages available.

**RetrieveContent:** PROCEDURE [content: BulkData.Sink, session: Session]  
REPORTS [AuthenticationError, OtherError, SessionError, ServiceError, TransferError] = 4;  
-- Receives the content of the message at the head of the slot queue implied  
-- by 'handle'. This operation is legal only directly after RetrieveEnvelope.

**AbortRetrieval :** PROCEDURE [session: Session]  
REPORTS [AuthenticationError, OtherError, SessionError, ServiceError, TransferError] = 3;  
-- Aborts retrieval of the message at the head of the  
-- delivery slot queue implied by 'session'. This operation is legal following either  
-- RetrieveContent or RetrieveEnvelope

**EndRetrieval:** PROCEDURE [session: Session]  
REPORTS [AuthenticationError, OtherError, SessionError, ServiceError, TransferError] = 6;  
-- Terminates a delivery session initiated by BeginRetrieval. This operation is legal  
-- directly after BeginRetrieval, or AbortRetrieval.

```

MailPoll: PROCEDURE [deliverySlot: Name, credentials: Credentials, verifier: Verifier]
RETURNS [mailPresent: BOOLEAN]
REPORTS [AccessError, AuthenticationError, OtherError, ServiceError] = 7;
-- Checks for mail at the delivery slot.

-- REMOTE ERRORS --

AccessError: ERROR [problem: AccessProblem] = 0;
AccessProblem: TYPE = {
    accessRightsInsufficient(0),      -- the user doesn't have access --
    accessRightsIndeterminate(1), -- cannot determine whether the user has access --
    deliverySlotInUse(2),             -- delivery slot in use ... sharing not allowed --
    noSuchDeliverySlot(3),            -- no delivery slot for this recipient at this server --
    deliverySlotIndeterminate(4),     -- cannot resolve name --
    wrongService(5)};                -- the wrong service was asked about that delivery slot --

AuthenticationError: ERROR [problem: Authentication.AuthenticationProblem] = 1;

InvalidRecipients: ERROR [nameList: InvalidNameList] = 3;

ServiceError: ERROR [problem: ServiceProblem] = 4;
ServiceProblem: TYPE = {
    cannotAuthenticate(0),    -- generally, an Authentication.CallProblem on the server --
    serviceFull(1),          -- this operation cannot be accepted --
    serviceUnavailable(2),   -- the service is unavailable to network clients --
    mediumFull(3)};         -- insufficient space to accept posted message --

TransferError: ERROR [problem: TransferProblem] = 5;
TransferProblem: TYPE = {
    aborted(0)};            -- the transfer was aborted by the source or sink --

OtherError: ERROR [problem: OtherProblem] = 6;
OtherProblem: TYPE = {
    cantExpedite(0),        -- the operation cannot be invoked by ExpeditedCourier --
    malformedMessage(1),
    -- this error is raised if, for example, the message body parts don't correspond to the
    -- length of the body PartTypes sequence passed to BeginPost or illegal envelope items are
    -- specified in the optionalPostingData passed to BeginPost
    incorrectContentsSize(2),
    -- the contentsSize specified was off by more than the allowable error
    -- (currently 5000 bytes)
    last(65535)};

SessionError: ERROR [problem: SessionProblem] = 7;
SessionProblem: TYPE = {
    invalidHandle(0),      -- the handle presented was not valid --
    wrongState(1)};
    -- the requested operation is not legal following the preceding operations.

END.

```

# MailTransportEnvelopeFormat D. program declaration

The complete declaration of the Mail Transport Envelope Format protocol is given below.

**MailTransportEnvelopeFormat: PROGRAM 1517 VERSION 1 =**

**BEGIN**

**DEPENDS UPON**

**MailTransport (17) VERSION 5,**

**Time (15) VERSION 2;**

*-- TYPES AND CONSTANTS --*

**Time: TYPE = Time.Time;**

**RName: TYPE = MailTransport.RName;**

**RecipientList: TYPE = MailTransport.RecipientList;**

**Envelope: TYPE = MailTransport.Envelope;**

**EnvelopeItem: TYPE = MailTransport.EnvelopeItem;**

**DeliveredRecipient: TYPE = RECORD [**  
**recipient: MailTransport.Recipient, when: Time];**

**NonDeliveredRecipient: TYPE = RECORD [**  
**recipient: MailTransport.Recipient, reason: Reason];**

**Reason: TYPE = {noSuchRecipient(0), noMailboxForRecipient(1), illegalName(2),**  
**timeout(3), reportNotAllowed(4), messageTooLong(5), ambiguousRName(6),**  
**illegalCharacters(7), unsupportedBodyParts(8), unsupportedContentsType(9),**  
**transientProblem(10), contentSyntaxError(11), tooManyRecipients(12),**  
**protocolViolation(13), x400PragmaticConstraintViolation(14),**  
**x400NoBilateralAgreement(15), accessRightsInsufficientForDL(16), other(17)};**

*-- if corrective action can be taken by remailing, 'transientProblem' is used.*

*-- EnvelopeItemTypes*

**postmark: EnvelopeItem = 0;      -- Time and site of posting.**

**Postmark: TYPE = RECORD [postedAt: RName, time: Time];**

**messageID: EnvelopeItem = 1;**

**MessageID: TYPE = MailTransport.MessageID;**

*-- Unique ID within the mail system for this message.*

**contentsType: EnvelopeItem = 2;      -- indicates format of message content.**

**ContentsType: TYPE = MailTransport.ContentsType;**

*-- indicates the format of the contents of a message.*

ctNull: ContentsType = 3777777777B -- no content  
ctStandardMessage: ContentsType = 4;  
ctReport: ContentsType = 6; -- A Mail Service-generated delivery or non-delivery report.

tableOfContents: EnvelopeItemtype = 3;  
TableOfContents: TYPE = SEQUENCE 500 OF RECORD [  
    type: MailTransport.BodyPartType,  
    sizeInBytes: LONG CARDINAL];  
    --There can be up to 500 body parts in a message.

BodyPartIndex: TYPE = CARDINAL;  
    -- Used as an index into the tableOfContents.

nullTableOfContents: TableOfContents = RECORD [];

contentsSize: EnvelopeItemtype = 4; -- size of message content in bytes  
ContentsSize: TYPE = LONG CARDINAL;

originator: EnvelopeItemtype = 5; -- authenticated sender of the message  
Originator: TYPE = RName;

report: EnvelopeItemtype = 6;  
Report: TYPE = RECORD [  
    originalEnvelope: Envelope,  
    fateOfThisReport: CHOICE OF {  
        delivered = > {contentsTruncated(0), noProblem(1)},  
        notDelivered = > RECORD [  
            reason: Reason,  
            reportCreationPostmark: Postmark]],  
    reportType: CHOICE OF {  
        dlMember(0) = > RECORD [  
            dlName: RName,  
            invalidDLRecipients: SEQUENCE OF NonDeliveredRecipient],  
        other(1) = > RECORD [  
            succeeded: SEQUENCE OF DeliveredRecipient,  
            failed: SEQUENCE OF NonDeliveredRecipient]]];

returnToName: EnvelopeItemtype = 7;  
ReturnToName: TYPE = RName; -- default = originator  
    -- Generated and used by the Mail System. This information is not interesting to users

previousRecipients: EnvelopeItemtype = 8;  
PreviousRecipients: TYPE = RecipientList;  
    -- Generated and used by the mail system. This information is not interesting to users.

gatewayPostmark: EnvelopeItemtype = 9;  
GatewayPostmark: TYPE = Postmark;

addressChangeNotice: EnvelopeItemtype = 10;  
AddressChangeNotice: TYPE = RName; -- original recipient name  
    -- Set on messages forwarded by the Mail Service.

**transportOptions: EnvelopeItem** = 11; -- default = all TRUE

**TransportOptions: TYPE** = RECORD[  
    **returnOfContents: BOOLEAN**,  
    **alternateRecipientAllowed: BOOLEAN**];

**x400SpecificReportInformation: EnvelopeItem** = 12;

**X400SpecificReportInformation: TYPE** = SEQUENCE 4 OF **X400ReportItem**;

**X400ReportItem: TYPE** = SEQUENCE OF UNSPECIFIED;

-- This envelope item contains the X.400 mpduIdentifier, uaContentID,  
-- the PerRecipientFlag plus ExtensionIdentifier of each recipient  
-- and the TraceInformation of messages originating in the X.400 mail  
-- system. The mpduIdentifier, the uaContentID and the  
-- PerRecipientFlag plus ExtensionIdentifier for each recipient  
-- are always present. The TraceInformation is only present if an X.400  
-- audit and confirmed report was requested. This information is simply reflected by the  
-- Xerox mail system in reports to the X.400 mail system, and users should not be  
-- confronted with it.

**otherRecipients: EnvelopeItem** = 13;

**OtherRecipients: TYPE** = RecipientList; -- when present, contains all the original recipients

**priority: EnvelopeItem** = 14;

**Priority: TYPE** = {nonUrgent(0), normal(1), urgent(2)}; -- If absent, defaults to normal.

**ConvertedItem: TYPE** = {ia5TextToTeletex(0), teletexToTelex(1), teletexToIA5Text(2),  
    telexToTeletex(3)};

**converted: EnvelopeItem** = 15;

-- Indicates the conversions a message has been through.

**Converted: TYPE** = SEQUENCE 10 OF **ConvertedItem**;

**authenticationLevelOfSender: EnvelopeItem** = 16;

**AuthenticationLevelOfSender: TYPE** = {strong(0), simple(1), foreign(2)};

-- Indicates the level of authentication with which the sender has been authenticated to the  
-- Xerox mail system.

END.



The complete declaration of the MailFormat protocol is given below. This protocol describes the collection of message body parts associated with the **MailTransport.ctStandardMessage** contents type, which is used for interpersonal messages.

**MailFormat: PROGRAM 1516 VERSION 3 =**

**BEGIN**

**DEPENDS UPON**

**Time(15) VERSION 2,**

**MailTransport(17) VERSION 5,**

**MailTransportEnvelopeFormat(1517) VERSION 1;**

*-- BULK DATA FORMAT --*

*-- All messages consist of a sequence of message body parts.*

*-- TYPES AND CONSTANTS --*

*-- A standardMessage has a heading body part, followed by a sequence of message body parts*

**BodyPartType: TYPE = MailTransport.BodyPartType;**

**headingBodyPart: BodyPartType = 0;**

**HeadingBodyPart: TYPE = SEQUENCE 150 OF Attribute;**

*-- There can be up to 150 attributes in a heading.*

**AttributeType: TYPE = LONG CARDINAL;**

**Attribute: TYPE = RECORD[type: AttributeType, value: SEQUENCE OF UNSPECIFIED];**

**RName: TYPE = MailTransport.RName;**

**RNameList: TYPE = SEQUENCE OF RName;**

**Time: TYPE = Time.Time;**

**IPMessageID: TYPE = RECORD[originator: RName, uniqueString: STRING];**

*-- Note: The originator is optional in X.400, so clients cannot assume [during ID comparisons] that other clients will supply this field.*

**IPMessageIDList: TYPE = SEQUENCE 100 OF IPMessageID;**

*-- There can be up to 100 message ids in a list.*

*-- Mail Attributes of an interpersonal message. Each user agent must construct a collection of attributes to associate with a message.*

**messageID: AttributeType = 1;**

**MessageID: TYPE = IPMessageID;**

*-- Contains the unique ID for the message.*

**originator: AttributeType = 2;**  
**Originator: TYPE = RName;**  
*-- The name of the user who actually submitted the message. The contents of this field*  
*-- are normally the same as the contents of the originator field of the envelope.*

**authorizingUsers: AttributeType = 3;**  
**AuthorizingUsers: TYPE = RNameList;**  
*-- The name(s) of the user(s) who authorized the message, to be used when different from*  
*-- the name in the originator field.*

**primaryRecipients: AttributeType = 4;**  
**PrimaryRecipients: TYPE = RNameList;**  
*-- The names of the primary recipients of the message.*

**copyRecipients: AttributeType = 5;**  
**CopyRecipients: TYPE = RNameList;**  
*-- The names of the secondary recipients for the message.*

**blindCopyRecipients: AttributeType = 6;**  
**BlindCopyRecipients: TYPE = RNameList;**  
*-- Identifies recipients for "blind" copies of the message. The members of this list will*  
*-- get copies of the message, but their names will not appear in any of the fields seen by*  
*-- other recipients. Other recipients will get a special version of the message with the*  
*-- BlindCopyRecipients omitted.*

**inReplyTo: AttributeType = 7;**  
**InReplyTo: TYPE = IPMessageID;**  
*-- Designates previous correspondence to which this message is a reply. The usual*  
*-- contents of this field would be the IPMessageID field of the message being replied to.*

**obsoletes: AttributeType = 8;**  
**Obsoletes: TYPE = IPMessageIDList;**  
*-- Identifies one or more messages that this one supplants. Should contain one or more*  
*-- message IDs.*

**crossReferences: AttributeType = 9;**  
**CrossReferences: TYPE = IPMessageIDList;**  
*-- Identifies other correspondence that this message references.*

**subject: AttributeType = 10;**  
**Subject: TYPE = STRING;**  
*-- Whatever information the originator provided to summarize or indicate the nature of*  
*-- the message.*

**expiryDate: AttributeType = 11;**  
**ExpiryDate: TYPE = Time;**  
*-- The date on which a message loses effect.*

**replyBy: AttributeType = 12;**  
**ReplyBy: TYPE = Time;**  
*-- The date by which replies to this message are requested.*

**replyToUsers: AttributeType = 13;**  
**ReplyToUsers: TYPE = RNameList;**  
*-- The names to be used as recipients for replies to this message.*

**importance: AttributeType = 14;**  
**Importance: TYPE = {low(0), normal(1), high(2)};**  
*-- Indicates the significance of the message.*

**sensitivity: AttributeType = 15;**  
**Sensitivity: TYPE = {personal(0), private(1), companyConfidential(2)};**  
*-- Indicates the sensitivity of the message.*

**autoforwarded: AttributeType = 16;**  
**Autoforwarded: TYPE = BOOLEAN;**  
*-- Indicates that the message was auto-forwarded.*

**immutable: AttributeType = 17;**  
**Immutable: TYPE = ARRAY 0 OF UNSPECIFIED;**  
*-- If present, this attribute prevents a mail system from modifying the contents of a message.*

**replyRequest: AttributeType = 18;**  
**ReplyRequest: TYPE = RNameList;**  
*-- If present, this attribute indicates who is expected to reply to the message.*

**textAnnotation: AttributeType = 19;**  
**TextAnnotation: TYPE = STRING;**  
*-- Plain text heading component. A maximum of 2000 characters are allowed.*

**forwardedHeadings: AttributeType = 20;**  
**ForwardedHeadings: TYPE = SEQUENCE OF ForwardedMessageInfo;**  
**ForwardedMessageInfo: TYPE = RECORD [**  
    **heading: SEQUENCE 150 of Attribute,**  
    **associatedBodyParts: SEQUENCE 500 OF BodyPartIndex,**  
    **indexOfParentHeading: IndexOfParentHeading];**

**IndexOfParentHeading: TYPE = CHOICE OF {**  
    **null (0) = > RECORD [],**  
    **nested (1) = > RECORD [index: CARDINAL];**  
*-- This attribute will appear in messages containing a forwarded message. Note that*  
*-- headings may be nested several levels deep.*

*-- STANDARD MESSAGE BODY PART TYPES --*

*-- Clients are encouraged to use these predefined types to identify messages that*  
*-- have the specified characteristics.*

*-- heading body part = 0;*

**vpFolder: BodyPartType = 1;**  
*-- A Xerox Filing Protocol Filing.SerializedFile representing a viewpoint folder.*

**nsTextFile: BodyPartType = 2;**  
*-- A Xerox Filing Protocol Filing.SerializedFile representing a text file.*

**vpDocument: BodyPartType = 3;**

-- *A Xerox Filing Protocol Filing.SerializedFile representing a viewpoint document.*

**otherNSFile: BodyPartType = 4;**

-- *A Xerox Filing Protocol Filing.SerializedFile other than vpFolder, nsTextFile, or vpDocument.*

**multinationalNote: BodyPartType = 5;**

-- *A maximum of 3777777777B characters from the Xerox Character Code Standard.*

**ia5Note: BodyPartType = 6;**

-- *A maximum of 3777777777B characters from international Alphabet No 5. Very similar to ASCII.*

**pilotFile: BodyPartType = 7;**

-- *The file type used in the Xerox Development Environment.*

**g3Fax: BodyPartType = 8;**

-- *A g3Fax document encoded as defined in X.420 [X.400].*

**teletex: BodyPartType = 9;**

-- *A teletex document encoded as defined in X.420 [x.400].*

**telex: BodyPartType = 10;**

-- *A telex document encoded as defined in F.60 [Telex].*

**iso6937Note:BodyPartType = 11;**

-- *An ISO6937 body part X.409 encoded according to the CEN/CENELEC implementation agreements. [CEN/CENELEC]*

**interpress:BodyPartType = 12;**

-- *An Interpress body part encoded according to the Xerox Interpress Electronic Printing Standard [Interpress].*

**END.**

The complete declaration of the Inbasket protocol is given below.

**Inbasket: PROGRAM 18 VERSION 2 =**

**BEGIN**

**DEPENDS UPON**

**Authentication (14) VERSION 2,**  
**BulkData (0) VERSION 1,**  
**Clearinghouse (2) VERSION 3,**  
**MailFormat (1516) VERSION 3,**  
**MailTransport (17) VERSION 5,**  
**MailTransportEnvelopeFormat (1517) VERSION 1;**

*-- TYPES AND CONSTANTS --*

**Credentials: TYPE = Authentication.Credentials;**

**Verifier: TYPE = Authentication.Verifier;**

**Name: TYPE = Clearinghouse.Name;**

**Index: TYPE = LONG CARDINAL;**      *-- Names a message; unique until the mailbox is moved.*

**nullIndex: Index = 0;**

**Range: TYPE = RECORD[low, high: Index];**

**MessageStatus: TYPE = RECORD [**

**userDefinedStatus: CARDINAL,**

**existenceOfMessage: {new(0), known(1)}];**

**BodyPartSequence: TYPE = SEQUENCE 500 OF**

**MailTransportEnvelopeFormat.BodyPartIndex;**

**allBodyParts: BodyPartSequence = [];**

*-- Represents all the message body parts.*

**BodyPartsStatus: TYPE = SEQUENCE 500 OF BOOLEAN;**

*-- For each body part of a message, indicates whether the body part is deletable. A value*

*-- of TRUE means that the body part is deletable. A message can have up to 500 body parts.*

**BodyPartStatusChangeSequence: TYPE = SEQUENCE 500 OF BodyPartStatusChange;**

**BodyPartStatusChange: TYPE = RECORD[**

**bodyPartIndex: MailTransportEnvelopeFormat.BodyPartIndex,**

**deletable: {true(0), noChange(1)}];**

*-- Status changes are unidirectional - there's no going back.*

**Status: TYPE = RECORD[**

**messageStatus: MessageStatus,**

**bodyPartsStatus: BodyPartsStatus];**

**Session: TYPE = RECORD[token: ARRAY 2 OF UNSPECIFIED, verifier: Verifier];**

**Anchor: TYPE = ARRAY 5 OF UNSPECIFIED;** *-- Stored by clients which cache info across sessions.*

```

State: TYPE = RECORD[
    new: CARDINAL,
    total: CARDINAL];

```

-- State of the mailbox, used for polling.  
-- Number of "new" messages in the mailbox.  
-- Total number of messages in the mailbox.

-- REMOTE PROCEDURES --

```

Login: PROCEDURE [inbasket: Name, credentials: Credentials, verifier: Verifier]
RETURNS [session: Session, state: State, anchor: Anchor]
REPORTS [AccessError, AuthenticationError, InbasketInUse, OtherError, ServiceError] = 5;

```

-- Creates a session with the mail service. Returns the Inbasket.State. Clients which  
-- retain indices between sessions should store a copy of the old Anchor. If the returned  
-- Anchor is different, then the message indices have been renumbered, and clients MUST  
-- flush the old indices from their cache to operate correctly.

```

Logout: PROCEDURE [session: Session]
REPORTS [AuthenticationError, OtherError, SessionError] = 4;

```

-- Ends the session at the mail service, and invalidates the session handle.

```

MailPoll: PROCEDURE [inbasket: Name, credentials: Credentials, verifier: Verifier]
RETURNS [state: State]
REPORTS [AccessError, AuthenticationError, OtherError, ServiceError] = 7;

```

-- MailPoll returns the state of the inbasket. A non-zero state.new implies that new mail  
-- exists. This is for use outside a session. This call can be expedited.

```

MailCheck: PROCEDURE [session: Session]
RETURNS [state: State]
REPORTS [AuthenticationError, OtherError, SessionError, ServiceError] = 6;

```

-- MailCheck returns the state of the inbasket. A non-zero state.new implies that new mail  
-- exists. For use within a session. MailCheck should not be used to keep a session open.  
-- This call can be expedited.

```

RetrieveEnvelopes: PROCEDURE [index: Index, whichMsg: {this(0), next(1)}_
    Session: Session]
RETURNS [transportEnvelope: MailTransport.Envelope, status: Status, index: Index]
REPORTS [AuthenticationError, IndexError, OtherError, SessionError,
    ServiceError] = 2;

```

-- Returns the envelope, the inbasket status, and the message index of whichMsg. To start  
-- a stateless enumeration, whichMsg should be set to "next", pass in either a nullIndex  
-- or an index saved from the last time the User Agent inspected the mailbox.  
-- Returns "nullIndex" when there are no more  
-- envelopes. For random access to the inbasket, whichMsg should be set to "this", and  
-- IndexError will be raised if the index does not exist.

```

RetrieveBodyParts: PROCEDURE [message: Index, bodyParts: BodyPartSequence,
    contents: BulkData.Sink, session: Session]
REPORTS [AuthenticationError, IndexError, OtherError, SessionError, ServiceError,
    TransferError] = 8;

```

-- Retrieves body parts of a message named via "Index", to the local workstation.  
-- Either individual parts, or the whole message may be specified (see the allBodyParts  
-- constant above.) The RetrieveBodyParts procedure returns the body parts in the order  
-- that they were requested. The body parts are concatenated without any structuring

- *information between them, so that the client can determine where each body part begins*
- *by using the body part sizes given in the tableOfContents.*

**ChangeMessageStatus:** PROCEDURE [range: Range, changeUserDefinedStatus: BOOLEAN, newUserDefinedStatus: CARDINAL, session: Session]

**REPORTS** [AuthenticationError, IndexError, OtherError, SessionError, ServiceError] = 0;

- *Makes a message known, and changes the userDefinedStatus if the caller wants to do*
- *so. A message cannot be made new again.*

**ChangeBodyPartsStatus:** PROCEDURE [index: Index, setStatusTo: BodyPartStatusChangeSequence, session: Session]

**RETURNS** [deleted: BOOLEAN]

**REPORTS** [AuthenticationError, IndexError, OtherError, SessionError, ServiceError] = 3;

- *Makes a message known and changes its status. Individual message body parts cannot*
- *be deleted. A body part that is marked deletable may be retrieved by a User Agent. If all*
- *of the body parts of a message are marked deletable, then the message will be deleted by*
- *the Mail Service.*

**Delete:** PROCEDURE [range: Range, session: Session]

**REPORTS** [AuthenticationError, OtherError, SessionError, ServiceError] = 1;

- *Deletes a range of messages from the inbasket.*
- *If there aren't any messages in the range, no error is raised.*

**GetSize:** PROCEDURE [inbasket: Name, credentials: Credentials, verifier: Verifier]

**RETURNS** [sizeInBytes: LONG CARDINAL]

**REPORTS** [AuthenticationError, AccessError, OtherError, ServiceError] = 10;

- *Retrieves the sizeInBytes of a mailbox. Suggested use: to warn users when an inbasket*
- *grows excessively large.*

-- REMOTE ERRORS --

**AccessError:** ERROR [problem: AccessProblem] = 0;

**AccessProblem:** TYPE = {

- accessRightsInsufficient(0), -- *the user doesn't have access* --
- accessRightsIndeterminate(1), -- *cannot determine if the user has access* --
- noSuchInbasket(2), -- *no inbasket for this recipient* --
- inbasketIndeterminate(3), -- *cannot resolve potential alias* --
- wrongService(4)}; -- *mailbox does not reside on this service* --

**AuthenticationError:** ERROR [problem: Authentication.AuthenticationProblem] = 1;

**SessionError:** ERROR [problem: SessionProblem] = 5;

**SessionProblem:** TYPE = {

- tokenInvalid(0)}; -- *the Inbasket.Session is invalid* --

**ServiceError:** ERROR [problem: ServiceProblem] = 6;

**ServiceProblem:** TYPE = {

- cannotAuthenticate(0), -- *generally, an Authentication.CallProblem on the server* --
- serviceFull(1), -- *no more operations of that type can be accepted* --
- serviceUnavailable(2)}; -- *operations of that type are currently disabled* --

```
TransferError: ERROR [problem: TransferProblem] = 7;
TransferProblem: TYPE = {
    aborted(0);           -- the transfer was aborted by the source or sink --

OtherError: ERROR [problem: OtherProblem] = 8;
OtherProblem: TYPE = {
    cantExpedite(0),       -- the operation cannot be expedited --
    malformedMessage(1),   -- the message bodyparts or other portions are malformed --
    invalidOperation(2),   -- the sequence of operations is invalid --
    last(65535)};

IndexError: ERROR [problem: IndexProblem] = 9;
IndexProblem: TYPE = {
    invalidIndex(0),       -- the index doesn't name an existing message --
    invalidBodyPartIndex(1)}; -- the index doesn't name an existing body part --

InbasketInUse: ERROR [user: Name] = 10;      -- inbasket is currently in use by "user". --

END.
```

An *expanding ring broadcast* is an algorithm used to locate a (particular kind of) system element in the internet. It is particularly useful for locating Mail Services for posting messages since a list of nearby Mail Services is not necessarily obtainable in any other way. It is not mandatory that this technique be used, but it is strongly recommended.

When describing the expanding ring broadcast, it is necessary to refer to "distance" in the internet. The metric that will be used is the *hop*. (This term is defined in Internet Transport Protocols [Internet].) The number of hops away that a network is from the local network (the one to which the client is attached) is defined as the number of internetwork routers a packet must traverse to get to that network. Note that this notion of distance is only approximately related to transit times since close-by networks may be connected via slower lines than more distant networks. The *radius* of a broadcast is defined to be the number of hops from the local network to the target network.

To do an expanding ring broadcast the client must acquire the routing tables describing the internet topology. This can be done using the Routing Information Protocol defined in Internet Transport Protocols [Internet].

An expanding ring broadcast begins when the client makes a directed broadcast (as defined in Internet Transport Protocols [Internet]) of an operation on the local network. The Mail Service uses expanding ring broadcast to locate Mail Servers, so the operation that would be broadcast in this case is the **ServerPoll** procedure (see Section 17.2 for a description of the level 3 packet format used). The client should wait for a reply for a reasonable amount of time, and possibly retry. If the client receives no reply then it should repeat the broadcast on networks that are one hop away. The client should continue its broadcast on networks that are increasingly farther away until it locates a system element it is looking for. It is recommended that clients limit the radius of their search to six hops.

See Appendix H for a method of choosing among the responses to the expanding ring broadcast of the **ServerPoll** operation.



During the course of a client's operation, it may locate one or more Mail Services for purposes of message delivery. It is not necessary, however, for these same Mail Services to be used for posting messages. Indeed, *any* nearby Mail Service is a candidate to receive posted messages. This section discusses an appropriate way to locate such Mail Services and choose among them. It is strongly recommended that this approach be used.

---

## **H.1 Locating nearby Mail Services**

---

To locate nearby Mail Services, the client does an expanding ring broadcast (see Appendix G) of the **BroadcastServerPoll** operation (see Section 4.2.2). Every Mail Service on a given net will respond to a **BroadcastServerPoll** operation with a package of useful information, as defined by the results of the **ServerPoll** operation.

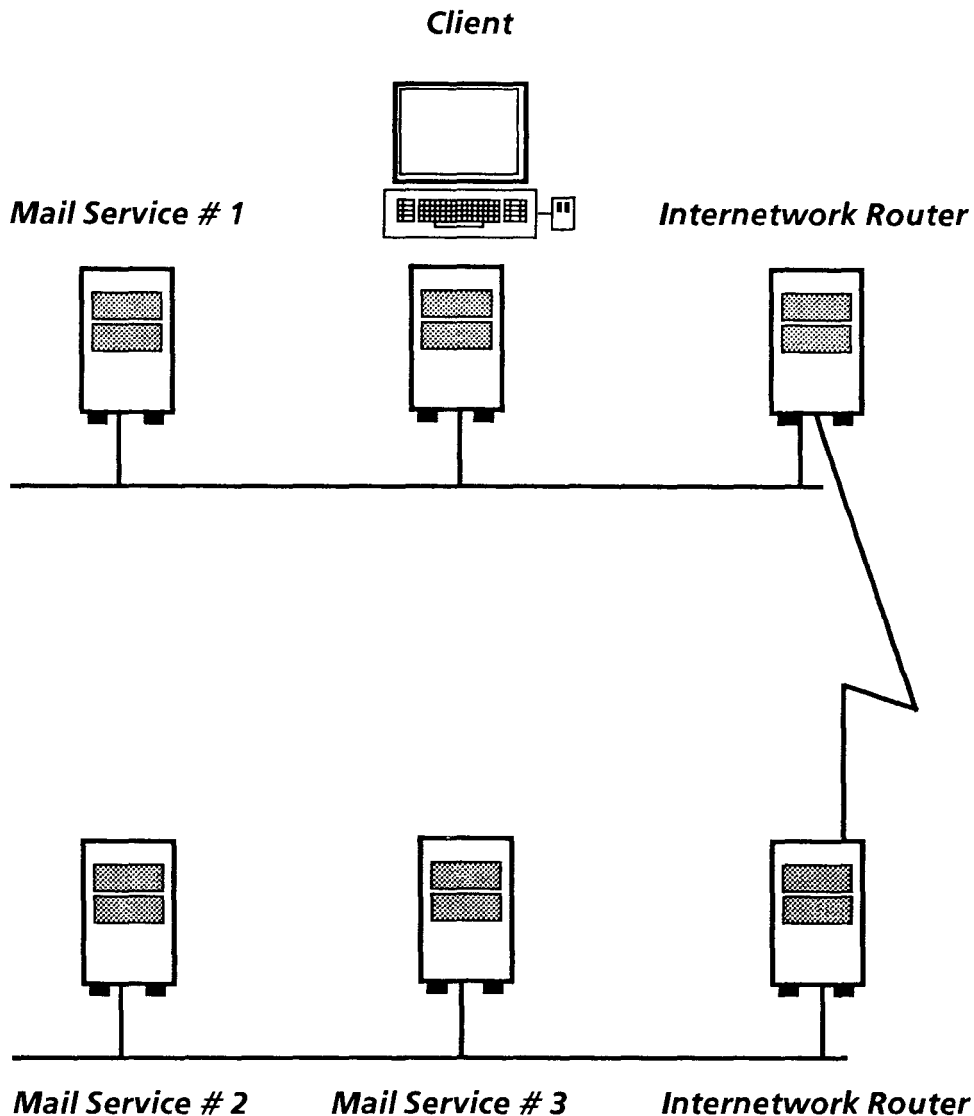
The **BroadcastServerPoll** operation is expensive and so should not be done before each posting call. Instead, useful addresses obtained through the procedure should be cached and kept for some time. The address for a server could be kept until the server fails to respond. If the server is not on the local net, or if it could have been more willing to receive mail, the client should occasionally try to find a different server for posting.

---

## **H.2 Choosing among the available Mail Services**

---

Once the client has a set of Mail Services to choose from, it should select a Mail Service to post messages to for each message size it is likely to send. Considerations in choosing a Mail Service should be: round trip delay time to the server and the willingness of the Mail Service to accept a message in the desired size range (this information is returned by the **ServerPoll** call). An example of how this could be done follows.



If Mail Service # 1 is down when the client tries to locate a Mail Service for posting, and Mail Service # 2 and Mail Service # 3 respond with willingnesses of [10, 10, 8, 5, 1] and [8, 8, 1] respectively, then Mail Service # 2 should be used to post messages less than or equal to 4096 bytes. Since Mail Service # 2 is not on the local net, the client should try periodically to reach a closer Mail Service.

If all of the Mail Services are available when the client tries to locate a Mail Service for posting, and Mail Service # 1, Mail Service # 2, and Mail Service # 3 respond with willingnesses of [10, 10, 1], [10, 10, 1], and [10, 10, 8, 1] respectively, then Mail Service # 1 should be chosen to post messages that are less than or equal to 64 bytes because the round trip distance from the client to Mail Service # 1 is less than that to Mail Service # 2, while Mail Service # 3 should be used to post messages from 65 bytes to 512 bytes.

If Mail Service # 3 is down when the client tries to locate a Mail Service for posting, and Mail Service # 1 and Mail Service # 2 respond with willingnesses of [8, 8, 1] and [10, 10, 1] respectively, the choice of Mail Service depends on whether the client is more concerned

about making calls to the local net or about sending messages to the Mail Service that can most readily process them.



In the following simple examples of protocol exchanges between a Mail Service client and a Mail Service, it is assumed that the client has already obtained valid credentials and verifiers via the Authentication Protocol [Authentication].

---

## I.1 Posting mail

---

First the Mail Service client must find a Mail Service to post to. This is described in Appendix H. Having found a Mail Service to post to the client performs the following Mail Transport protocol actions:

```
BeginPost[
  postingData: [
    recipients: [xns [organization: "Xerox", domain: "DSBU", object: "George"]],
    contentsType: ctStandardMessage, contentsSize: 200,
    bodyPartTypeSequence: [headingBodyPart, nsTextFile]],
  postIfInvalidNames: FALSE, allowDLRecipients: FALSE, optionalEnvelopeData: [],
  credentials: userCredentials, verifier: someVerifier]
```

The **postingData** contains the recipients of the message ("George"), the contents type (**ctStandardMessage** which is a simple message) and size (200 bytes), together with information about the type of the body parts of the message (one part of type **headingBodyPart** and one part of type **nsTextFile**). This data must always be given to the Mail Service for every message. There was no **optionalEnvelopeData** data. **postIfInvalidNames** is FALSE so the operation will report an error if invalid name is encountered. No list of invalid names will be returned in **invalidNames**. In this example none of the recipients of the message is an XNS group name so **allowDLRecipients** is FALSE. **userCredentials** and **someVerifier** represent the credentials and verifier to be used for this operation.

```
RETURNS[session: sessionData, invalidNames: []]
```

The name of the session returned by the call to **BeginPost** is used during subsequent calls to the Mail Service to identify the client. It will be called **sessionData** in this example. Because **postInvalidNames** is FALSE in the call, the **invalidNames** return value will be nil.

The client now calls the following procedure once per body part of the message. The first part is the heading body part:

```
PostOneBodyPart [session: sessionData, bodyPartType: headingBodyPart,
  contents: bulkDataSource1]
```

For a contents type of **ctStandard Message** the heading is contained in body part zero. The data is sent to the Mail Service via a single bulk data transfer from **bulkDataSource1**. The

client calls **PostOneBodyPart** again to post the other and final part of the message, the text file part.

**PostOneBodyPart** [session: sessionData, bodyPartType: nsTextFile,  
contents: bulkDataSource2]

To finish posting a message to the Mail Service the client calls:

**EndPost** [session: sessionData, abortPost: FALSE]  
**RETURNS** [messageID: [1, 2, 3, 4, 5]]

The Mail Service returns a message ID ([1,2,3,4,5]) that uniquely identifies the message from this point on. Note that the client cannot assume that the Mail Service has all or any part of the message until the successful return of **EndPost**. At this point the Mail Service assumes responsibility for the message.

---

## I.2 Retrieving mail via the delivery slot

---

There is a delivery slot associated with each XNS recipient. The delivery slot for a particular recipient is found by looking in the Clearinghouse database as described in Section 3.4.1.

All protocol transactions on the delivery slot use the MailTransport protocol. The client views the mail delivery slot as a first-in-first-out (FIFO) queue of messages.

The client periodically polls the delivery slot for the recipient in question via:

**MailPoll** [deliverySlot: [organization: "Xerox", domain: "DSBU", object: "George"],  
credentials: differentCredentials, verifier: differentVerifier]  
**RETURNS** [mailPresent: TRUE]

This is a lightweight method of determining if **deliverySlot** has any new mail. The **mailPresent** value of **TRUE** indicates that one or more messages have arrived.

Once there is new mail available at the Mail Service the client may begin protocol actions to retrieve the message.

**BeginRetrieval** [  
deliverySlot: [organization: "Xerox", domain: "DSBU", object: "George"],  
credentials: differentCredentials, verifier: thirdVerifier]  
**RETURNS** [session: newSession]

**BeginRetrieval** starts the delivery process for the recipient **deliverySlot**. The Mail Service returns a **newSession** for use in subsequent calls to uniquely identify the client.

The client now calls:

**RetrieveEnvelope** [session: newSession]  
**RETURNS** [empty: FALSE,  
envelope: [  
[type: postmark,value:  
[postedAt: [xns  
[organization: "Xerox", domain: "DSBU", object: "MailServer1"]],

```
time: 2745446400]],  
[type: messageId, value: [1, 2, 3, 4, 5]],  
[type: contentsType, value: ctStandardMessage],  
[type: tableOfContents, value:  
  [[type: headingPodyPart, sizeInBytes: 100],  
   [type: nsTextFile, sizeInBytes: 100]]],  
[type: contentsSize, value: 200],  
[type: originator, value: [xns  
  [organization: "Xerox", domain: "DSBU", object: "John"]]],  
[type: authenticationLevelOfSender, value: strong]]]
```

**RetrieveEnvelope** returns the **envelope** of the message at the "head" of the queue implied by **newSession**. If there are no messages in the queue then **empty** will be **TRUE**. In this case **envelope** contains:

the postmark: the message was posted to MailServer1 at 8:00AM on February 28, 1988;

the message ID: a unique sequence of 5 numbers;

the type of the contents: **ctStandardMessage**;

the table of contents of the message: a heading body part of length 100 bytes, and a **nsTextFile** part of length 100 bytes;

the contents size: 200 bytes;

the originator of the message: a person identified as John:DSBU:Xerox;

the security level of the sender: strong authentication was used.

The client now transfers the content of the message implied by **newSession** via:

**RetrieveContent** [content: **bulkDataSink**, session: **newSession**]

The content is sent via a bulk data transfer to the location described by **bulkDataSink**. All the content (i.e., all the body parts) is delivered via a single bulk data operation. It is the client's responsibility, by examining the table of contents field within the envelope of the message, to construct the body parts as separate entities for use by users or other client side software.

It is possible for the client to call **AbortRetrieval** after either **RetrieveEnvelope** or **RetrieveContent**. This aborts the retrieval of this message. Note, however, that as the client views the delivery slot as a first-in, first-out queue, it cannot retrieve any further messages from the delivery slot until this message has been retrieved or deleted. (A message may be deleted from the delivery slot by a call to **RetrieveEnvelope** immediately followed by a second call to **RetrieveEnvelope**.)

The client continues to call **RetrieveEnvelope**, **RetrieveContent**, until there are no more messages at the delivery slot.

**RetrieveEnvelope** [session: **newSession**] RETURNS [empty: **TRUE**, envelope: []]

There are no more messages (empty = **TRUE**). At this point the client calls:

**EndRetrieval** [session: **newSession**]

### I.3 Retrieving mail via the Inbasket

An alternative, lightweight method for clients to periodically poll an inbasket is provided in the Inbasket protocol.

The client representing "George" calls:

**MailPoll** [inbasket: [organization: "Xerox", domain: "DSBU", object: "George"],  
credentials: credentialsData, verifier: verifierData]

This call returns.

**RETURNS** [state: [new: 1, total: 1]]

The **state.new**, since it is non-zero, indicates that a message is awaiting delivery. The client then initiates an Inbasket session with the Mail Service by calling:

**Logon** [inbasket: [organization: "Xerox", domain: "DSBU", object: "George"],  
credentials: credentialsData, verifier: verifierData]

This call returns the values:

**RETURNS** [session: sessionData, state: [new: 1, total: 1], anchor: anchorData]

The **sessionData** contains the name of this session, **state** has the same values and meanings as it did in the **MailPoll** call, and **anchor** returns values to be used if the message indices were to be retained across sessions. In this case it is assumed that the indices are of no interest and the anchor data is ignored. The session data is an identifier the client uses in subsequent calls to the Mail Service to identify this session.

The client decides which message is of interest and calls:

**RetrieveEnvelopes** [index: nullIndex, whichMsg: next, session: sessionData]

where **nullIndex** and **whichMsg = next** indicates that the first message is to be retrieved. (If the client were performing random access to the inbasket, the **whichMsg** would be set to **this**, and **index** would indicate the index of the message. But in the current example a stateless enumeration of the inbasket is being performed by setting **whichMsg** to **next**, and repeatedly calling **RetrieveEnvelopes** with the **index** returned by the previous call. The **nullIndex** can be used to start an enumeration of all of the messages in the inbasket. The Mail Service will return **nullIndex** when there are no more messages in the inbasket.

The envelope of the message is returned in **transportEnvelope**.

**RETURNS** [  
  **transportEnvelope**: [  
    [type: postmark, value:  
      [postedAt: [xns  
        [organization: "Xerox", domain: "DSBU", object: "MailServer1"]],  
        time: 2745446400]],  
    [type: messageId, value: [1, 2, 3, 4, 5]],  
    [type: contentsType, value: ctStandardMessage],  
    [type: tableOfContents, value:  
      [[type: headingPodyPart, sizeInBytes: 100],

```

        [type: nsTextFile, sizeInBytes: 100]]],
    [type: contentsSize, value: 200],
    [type: originator, value: [xns
        [organization: "Xerox", domain: "DSBU", object: "John"]]],
    [type: authenticationLevelOfSender, value: strong]],
    status: [messageStatus: [userDefinedStatus: 0, existenceOfMessage: new],
        bodyPartsStatus:[FALSE, FALSE]],
    index: 1]

```

The envelope data has the same meaning as the previous example. The **userDefinedStatus** is not used in this example. The **existenceOfMessage** indicates the message has not been inspected before, and **bodyPartsStatus** indicates that there are two body parts. The message index of 1 is returned in **index**.

The client must now decide which body parts of the message to retrieve. The client then calls:

```

RetrieveBodyParts [message: 1, bodyParts: [allBodyParts], contents: bulkDataSink,
    session: sessionData]

```

This requests the retrieval of the body parts of message 1. **bodyParts** contains a list of the body parts that the client wishes to retrieve, in this case all of them. The body parts are returned via a bulk data transfer as specified by **bulkDataSink** in the order the client specified. It is the client's responsibility to reconstruct the individual body parts as separate entities for use by users or by client side software.

The client now updates the status of the inbasket on the Mail Service by calling:

```

ChangeBodyPartsStatus [index: 1,
    setStatusTo: [[bodyPartIndex: 0, deletable: TRUE]
        [bodyPartIndex: 1, deletable: true]],
    session: sessionData]

```

where **setStatusTo** describes which body parts are to be marked as being read. If all the body parts are marked as being deletable then **deleted** is returned **TRUE** as in this case.

**RETURNS [deleted: TRUE]**

Any messages that have all their body parts marked as deletable will be deleted.

The Inbasket protocol provides a method of checking within a session if there is any new mail. The client calls:

```

MailCheck [session: sessionData]
RETURNS [state: [new:0, total: 0]]

```

**state** shows that there is no new mail and so the client terminates the inbasket session with:

```

Logoff [session: sessionData]

```

---

## I.4 Handling Errors

---

Although the example didn't show any, nearly all the procedures in MailTransport and Inbasket can return remote errors. In particular the session a client has with a Mail Service may time out. Client implementations are responsible for handling these errors and are responsible for ensuring that the client exits from its protocol interaction with the Mail Service in a graceful manner.

The RName is closely related to the X.400 ORName. The definitions of both forms of name are given below, along with a description of how to translate from one to the other. Note: the definition of an ORName is given in X.409 notation.

```
RName: TYPE = CHOICE OF [
  xns(0) = > Clearinghouse.Name,
  gateway(1) = > RECORD [
    country: STRING,
    adminDomain: STRING,
    privateDomain: STRING,
    organization: STRING,
    organizationalUnits: SEQUENCE 20 OF STRING;
    personal: CHOICE OF {
      whole (0) = > STRING,
      broken (1) = > RECORD [given, initials, family, generation: STRING]}
    gatewaySpecificInformation: SEQUENCE 10 OF Attribute];
```

```
Attribute: TYPE = RECORD [type, value: STRING];
```

```
ORName :: = [APPLICATION 0] IMPLICIT SEQUENCE {
  StandardAttributeList,
  DomainDefinedAttributeList OPTIONAL}
```

```
StandardAttributeList :: = SEQUENCE {
  CountryName OPTIONAL,
  AdministrationDomainName OPTIONAL,
  [0] IMPLICIT X121Address OPTIONAL,
  [1] IMPLICIT TerminalID OPTIONAL,
  [2] PrivateDomainName OPTIONAL,
  [3] IMPLICIT OrganizationName OPTIONAL,
  [4] IMPLICIT UniqueUAIIdentifier OPTIONAL,
  [5] IMPLICIT PersonalName OPTIONAL,
  [6] IMPLICIT SEQUENCE OF OrganizationalUnit OPTIONAL}
```

**DomainDefinedAttributeList** :: = SEQUENCE OF **DomainDefinedAttribute**  
**DomainDefinedAttribute** :: = SEQUENCE {  
    type **PrintableString**,  
    value **PrintableString**}  
**CountryName** :: = [APPLICATION 1] CHOICE {  
    **NumericString**,  
    **PrintableString**}  
**AdministrationDomainName** :: = [APPLICATION 2] CHOICE {  
    **NumericString**,  
    **PrintableString**}  
**X121Address** :: = **NumericString**  
**TerminalID** :: = **PrintableString**  
**UniqueUAIentifier** :: = **NumericString**  
**PersonalName** :: = SET {  
    surName [0] IMPLICIT **PrintableString**,  
    givenName [1] IMPLICIT **PrintableString** OPTIONAL,  
    initials [2] IMPLICIT **PrintableString** OPTIONAL,  
    generationQualifier [3] IMPLICIT **PrintableString** OPTIONAL,  
**OrganizationalUnit** :: = **PrintableString**  
**PrivateDomainName** :: = CHOICE {  
    **NumericString**,  
    **PrintableString**}

Note that **NumericString** and **PrintableString** contain digits and characters respectively from the international alphabet number 5 [IA5].

To specify the name of an XNS recipient as an **ORName**, the **CountryName**, **AdministrationDomainName**, and **PrivateDomainName** are assigned by the recipient's systems administrator. The **OrganizationName** is the organization part of the recipient's 3-part Clearinghouse name. The SEQUENCE OF **OrganizationalUnits** contains one element, the domain part of the recipient's 3-part Clearinghouse name. The **surName** field of the **PersonalName** contains the local part of the recipient's 3-part Clearinghouse name.

To specify the name of an X.400 recipient as an **RName**, the gateway variant is used, and the **country**, **adminDomain**, **privateDomain**, **organization**, and **organizationalUnits** are the recipient's **CountryName**, **AdministrationDomainName**, **PrivateDomainName**, **Organization**, and **OrganizationalUnits** respectively. If the **PersonalName** contains only a **surName**, then the **whole** variant is used. Otherwise, the **broken** variant is used in the obvious way. An **X121Address**, **TerminalID**, or **UniqueUAIentifier** is translated into an **Attribute** in the **gatewaySpecificInformation** (see Appendix B).

Xerox Corporation  
Xerox Systems Institute  
475 Oakmead Parkway  
Sunnyvale, CA 94086

XEROX® is a trademark of XEROX CORPORATION.

Printed in U.S.A.