IEN: 85
RFC: 753

INTERNET MESSAGE PROTOCOL

Jonathan B. Postel

March 1979

Information Sciences Institute
University of Southern California
4676 Admiralty Way
Marina del Rey, California 90291

(213) 822-1511

< INC-PROJECT, MAIL-MAR-79.NLS.38, >, 31-Mar-79 19:50 JBP ;;;;

                         TABLE OF CONTENTS

                                PREFACE

This is the first edition of this specification and should be treated as
a request for comments, advice, and suggestions. A great deal of prior
work has been done on computer aided message systems and some of this is
listed in the reference section. This specification was shaped by many
discusions with members of the ARPA research community, and others
interested in the development of computer aided message systems. This
document was prepared as part of the ARPA sponsored Internetwork
Concepts Research Project at ISI, with the assistance of Greg Finn, Alan
Katz, Paul Mockapetris, and Mamie Chew.

                                                        Jon Postel

INTERNET MESSAGE PROTOCOL

1. INTRODUCTION

This document describes an internetwork message system. The system is
designed to transmit messages between message processing modules
according to formats and procedures specified in this document. The
message processing modules are processes in host computers. Message
processing modules are located in different networks and together
constitute an internetwork message delivery system.

This document is intended to provide all the information necessary to
implement a compatible cooperating module of this internetwork message
system.

1.1. Motivation

   As computer supported message processing activities grow on individual
   host computers and in networks of computers, there is a natural desire
   to provide for the interconnection and interworking of such systems.
   This specification describes the formats and procedures of a general
   purpose internetwork message system, which can be used as a standard
   for the interconnection of individual message systems, or as a message
   system in its own right.

   We also provide for the communication of data items beyond the scope
   of contemporary message systems. Messages can include typed segments
   which could represent drawings, or facsimile images, or digitized
   speech. One can imagine message stations equipped with speakers and
   microphones (or telephone hand sets) where the body of a message or a
   portion of it is recorded digitized speech. The output terminal could
   include a graphics display, and the message might present a drawing on
   the display, and verbally (via the speaker) describe certain features
   of the drawing. This specification provides basic data elements for
   the transmission of structured binary data, as well as providing for
   text transmission.

1.2. Scope

   The Internet Message Protocol is intended to be used for the
   transmission of messages between networks. It may also be used for
   the local message system of a network or host. This specification was
   developed in the context of the ARPA work on the interconnection of
   networks, but it is anticipated that it has a more general scope.

The focus here is on the internal mechanisms to transmit messages,
rather than the external interface to users. It is assumed that a
number of user interface programs will exist. These will be both new
programs designed to work with system and old programs designed to
work with earlier systems.

1.3. The Internetwork Environment

The internetwork message environment consists of processes which run
in hosts which are connected to networks which are interconnected by
gateways. Each individual network consists of many different hosts.
The networks are tied together through gateways. The gateways are
essentially hosts on two (or more) networks and are not assumed to
have much storage capacity or to "know" which hosts are on the
networks to which they are attached [5].

1.4. Operation

The model of operation is that this protocol is implemented in a
process. Such a process is called a Message Processing Module or MPM.
The MPMs exchange messages by establishing full duplex communication
and sending the messages in a fixed format described in this document.
The MPM may also communicate other information by means of commands
described here.

A message is formed by a user interacting with a User Interface
Program or UIP. The user may utilize several commands to create
various fields of the message and may invoke an editor program to
correct or format some or all of the message. Once the user is
satisfied with the messages it is "sent" by placing it in a data
structure shared with the MPM.

The MPM discovers the unprocessed input data (either by a specific
request or by a general background search), examines it, and using
routing tables determines which outgoing link to use. The destination
may be another user on this host, a user on another host in this
network, or a user in another network.

In the first case, another user on this host, the MPM places the
message in a data structure shared with the destination user, where
that user's UIP will look for incoming messages.

In the second case, the user on another host in this network, the MPM
transmits the message to the MPM on that host. That MPM then repeats
the routing decision, and discovering the destination is local to it,
places the messages in the data structure shared with the destination
user.

In the third case, the user on a host in another network, the MPM
transmits the messages to an MPM in that network if it knows how to
establish a connection directly to it, otherwise the MPM transmits the
message to an MPM that is "closer" to the destination. An MPM might
not know of direct connections to MPMs in all other networks, but it
must be able to select a next MPM to handle the message for each
possible destination network.

A MPM might know a way to establish direct connections to each of a
few MPMs in other nearby networks, and send all other messages to a
particular big brother MPM that has a wider knowledge of the internet
environment.

A individual network's message system may be quite different from the
internet message system. In this case, intranet messages will be
delivered using the network's own message system. If a message is
addressed outside the network, it is given to a MPM which then sends
it through the appropriate gateways via internet procedures and format
to (or toward) the MPM in the destination network. Eventually, the
message gets to a MPM on the network of the recipient of the message.
The message is then sent via the local message system to that host.

When local message protocols are used, special conversion programs are
required to transform local messages to internet format when they are
going out, and to transform internet messages to local format when
they come into the local environment. Such transformations are
potentially information lossy. The internet message format attempts
to provide features to capture all the information any local message
system might use. However, a particular local message system is
unlikely to have features equivalent to all the possible features of
the internet message system. Thus, in some cases the transformation
of an internet message to a local message discard of some of the
information. For example, if an internet message carrying mixed text
and speech data in the body is to be delivered in a local system which
only carries text, the speech data may be replaced by the text string
"There was some speech here". Such discarding of information is to be
avoided when at all possible, and to be defered as long as possible,
still the possibility remains, that in some cases, it is the only
reasonable thing to do.

1.5. Interfaces

The MPM calls on a reliable communication procedure to communicate
with other MPMs. This is a Transport Level protocol such as the TCP
[20]. The interface to such a procedure conventionally provides calls
to open and close connections, send and receive data on a connection,
and some means to signal and be notified of special conditions (i.e.,
interrupts).

   The MPM receives input and produces output through data structures
   that are produced and consumed respectively by user interface (or
   other) programs.

2. FUNCTIONAL DESCRIPTION

2.1. Terminology

  The basic unit transferred between networks is called a message. A
  message is made up of a transaction identifier (a number which
  uniquely identifies the message), a command list (which contains the
  necessary information for delivery), and the document list. The
  document list consists of a header and a body, which contains the
  actual data of the message.

  For a personal letter the document body corresponds to the contents
  the a letter, the document header corresponds to the the address and
  return address on the envelope.

  For an inter-office memo the document body corresponds to the text,
  the document header corresponds to the header of the memo.

  The commands correspond to the information used by the Post Office or
  the mail room to route the letter or memo.

  The messages are routed by a process called the message processing
  module or MPM. Messages are created and consumed by User Interface
  Programs (UIPs) in conjunction with users.

  Please see the Glossary section for a more complete list of
  terminology.

2.2. Assumptions

  The following assumptions are made about the internetwork environment:

  It is in general not known what format intranet addresses will assume.
  Since no standard addressing scheme would suit all networks, it is
  safe to assume there will be several and that they will change with
  time. Thus, frequent software modification throughout all internet
  MPMs would be required if such MPMs were to know about the formats on
  many networks. Therefore, each MPM which handles internet messages is
  required to know only the minimum necessary to deliver them.

  We require each MPM to know completely only the addressing format of
  its own network. In addition, the MPM must be able to select an
  output link for each message addressed to another network or host.
  This does not preclude more intelligent behavior on the part of a
  given MPM, but at least this minimum is necessary. Each network has a
  unique name and number.

  Each MPM will have a unique internet address. This feature will

enable every MPM to place a unique "handling-stamp" on a message which
passes through it en-route to delivery.

2.3. General Specification

There are several aspects to a distributed service to be specified.
First there is the service to be provided, that is, the
characteristics of the service as seen by its users. Second there is
the service it uses, that is, the characteristics it assumes to be
provided by some lower level service. And, third there is the
protocol used between the modules of the distributed service.
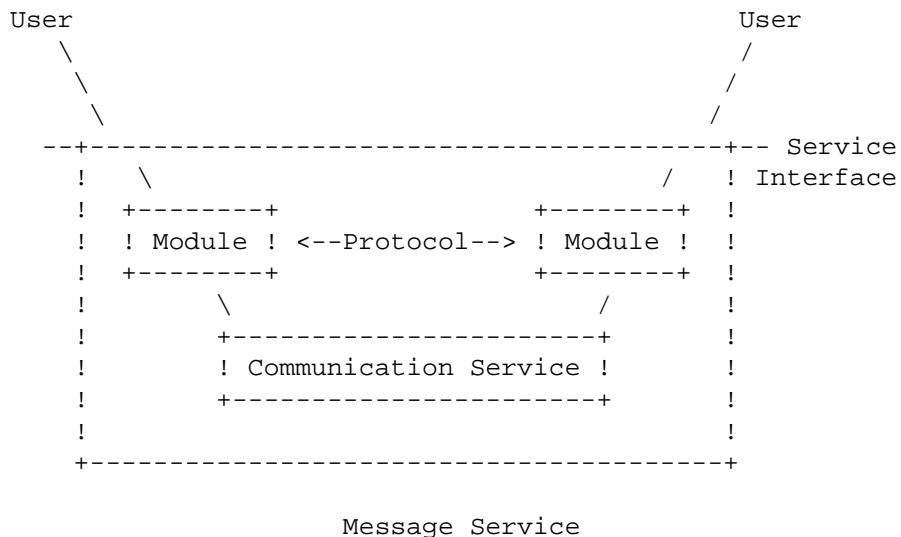
```
        User                                        User
          \                                        /
           \                                      /
            \                                    /
          --+------------------------------------+-- Service
            !   \                            /   ! Interface
            !  +--------+              +--------+ !
            !  ! Module ! <--Protocol--> ! Module ! !
            !  +--------+              +--------+ !
            !        \                    /      !
            !         +--------------------+     !
            !         ! Communication Service !  !
            !         +--------------------+     !
            !                                    !
            +------------------------------------+

                      Message Service

                         Figure 1.
```

The User/Message Service Interface

   The service the message delivery system provides is to accept
   messages conforming to a specified format and to attempt to deliver
   those messages, and to report on the success or failure of the
   delivery attempt. This service is provided in the context of an
   interconnected system of networks, and may involve relaying a
   message through several intermediate MPMs utilizing different
   communication services.

The Message/Communication Service Interface

   The message delivery system calls on a communication service to
   transfer information from one MPM to another. There may be
   different communication services used between different pairs of

      MPMs, though all communication services must meet the following
      service characteristics.

      It is assumed that the communication service provides a reliable two
      way data stream. Such a data stream can usually be obtained in
      computer networks from the transport level protocol, for example,
      the Transmission Control Protocol (TCP) [20]. In any case the
      properties the communication service must provide are:

         o Logical connections for two way simultaneous data flow of
            arbitrary data (i.e., no forbidden codes). Data is delivered
            in the order sent with no gaps.

         o Simple commands to open and close the connections, and to send
            and receive data on the connections.

         o A way to signal and be notified "out-of-band" (such as TCP's
            urgent) is available so that some messages can be labeled "more
            important" than others.

         o Controlled flow of data so that data is not transmitted faster
            that the receiver chooses to consume it (on the average).

         o Transmission errors are corrected without user notification or
            involvement. Complete breakdown on communication is reported
            to the user.

   The Message-Message Protocol

      The protocol used between the distributed modules of the message
      delivery system, that is, the MPMs is a small set of commands which
      convey requests and replies. These commands are encoded in a highly
      structured and rigidly specified format.

2.4. Mechanisms

   MPMs are processes which use some communication service. A pair of
   MPMs which can communicate reside in a common interprocess
   communication environment. A MPM might exist in two (or more)
   interprocess communication environments, and such an MPM might act to
   relay messages between MPMs in the environments.

```
      User                                              User
        \                                               /
         \                                             /
          \                                           /
      +-------------------------------------------------------+
      !    \                                     /     !
      ! +-----+              +-----+              +-----+   !
      ! ! MPM ! <--Protocol--> ! MPM ! <--Protocol--> ! MPM !   !
      ! +-----+              +-----+              +-----+   !
      !    !                 /   \                  !      !
      ! +--------------------+   +--------------------+   !
      ! !Communication Service A!   !Communication Service B!  !
      ! +--------------------+   +--------------------+   !
      !                                               !
      +-------------------------------------------------------+
```
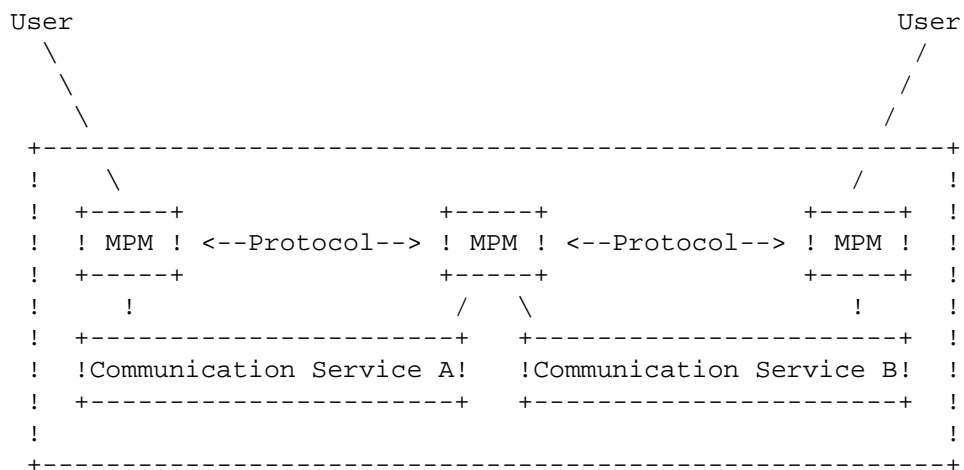
                 Message Service with Internal Relaying

                              Figure 2.

The transfer of data between UIPs and MPMs is conceived of as the
exchange of data structures which encode messages. The transfer of
data between MPMs is also in terms of the transmission of structured
data.

```
             +-----+       DATA          +-----+
       USER-->! UIP !-->STRUCTURES-->! MPM !-->other
             +-----+     +-----+       +-----+       MPMs
                           !      !
                           !  +-----+
                          +--!      !
                             !  +-----+
                            +--!      !
                               !      !
                               +-----+


              +-----+       DATA          +-----+
       other-->! MPM !-->STRUCTURES-->! UIP !-->USER
       MPMs    +-----+     +-----+       +-----+
                             !      !
                             !  +-----+
                            +--!      !
                               !  +-----+
                              +--!      !
                                 !      !
                                 +-----+
```
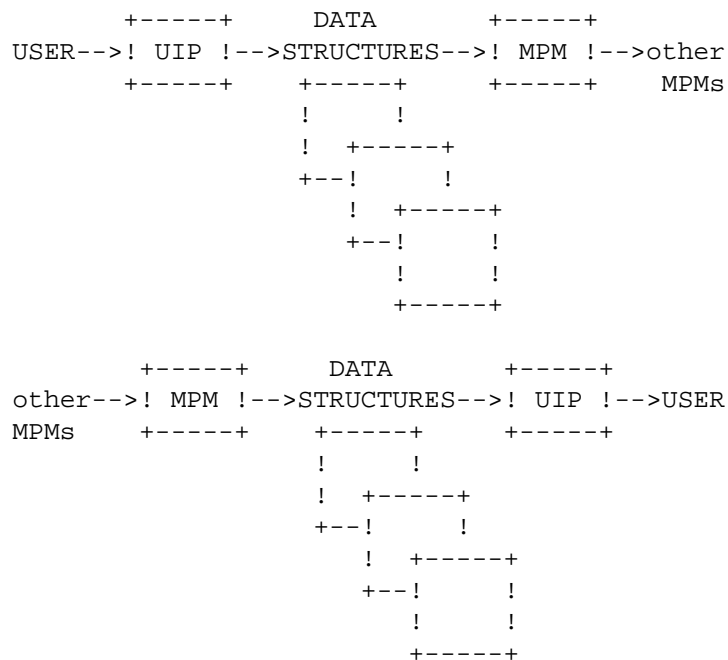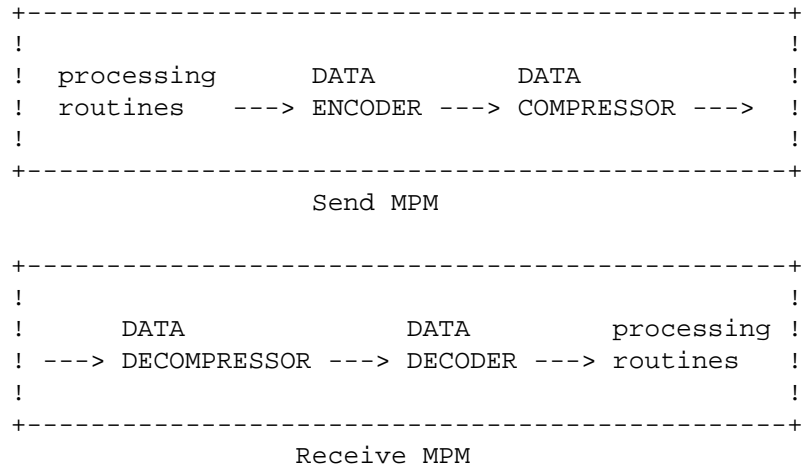
                         Message Flow

                          Figure 3.

   In the following, a message will be described as a structured data
   object represented in a particular kind of typed data elements. This
   is how a message is presented when transmitted between MPMs or
   exchanged between an MPM and a UIP. Internal to a MPM (or a UIP), a
   message may be represented in any convenient form. As the following
   figure shows, when a message is ready for transmission, it moves from
   the processing routines to be encoded in the typed data elements and
   then to a data compression routine, and is finally transmitted. On
   the receiving side, the message is first decompressed then decoded
   from the data element representation to the local representation for
   the processing routines.

```
      +-----------------------------------------------+
      !                                               !
      !  processing        DATA          DATA         !
      !  routines    ---> ENCODER ---> COMPRESSOR --->  !
      !                                               !
      +-----------------------------------------------+
                        Send MPM


      +-----------------------------------------------+
      !                                               !
      !       DATA             DATA        processing !
      ! ---> DECOMPRESSOR ---> DECODER ---> routines    !
      !                                               !
      +-----------------------------------------------+
                      Receive MPM

                      Detailed View

                       Figure 4.
```
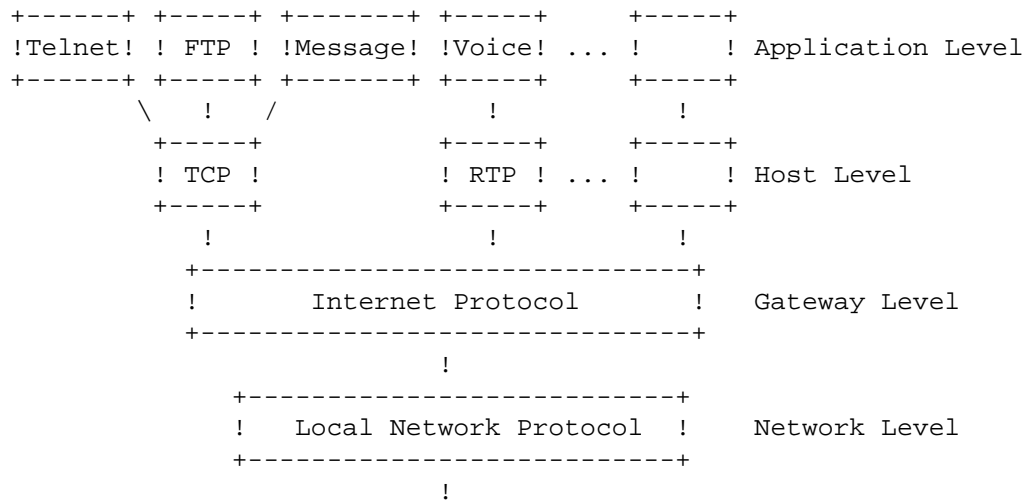
   2.5. Relation to Other Protocols

      The following diagram illustrates the place of the message protocol in
      the protocol hierarchy:

       +------+ +-----+ +-------+ +-----+   +-----+
       !Telnet! ! FTP ! !Message! !Voice! ... !   ! Application Level
       +------+ +-----+ +-------+ +-----+   +-----+
              \   !   /   !   !
               +-----+   +-----+   +-----+
               ! TCP !   ! RTP ! ... !   ! Host Level
               +-----+   +-----+   +-----+
                 !   !   !
               +------------------------------+
               !   Internet Protocol   ! Gateway Level
               +------------------------------+
                              !
                 +--------------------------+
                 !   Local Network Protocol !   Network Level
                 +--------------------------+
                              !

                        Protocol Relationships

                            Figure 5.

      The message protocol interfaces on one side to user interface programs
      and on the other side to a reliable transport protocol such as TCP.

                        3. DETAILED SPECIFICATION

The presentation of the information in this section is difficult since
everything depends on everything, and since this is a linear media it
has to come in some order. In this attempt, a very brief overview of
the message structure is given, then a radical switch is made to
defining the basic building blocks, and finally using the building
blocks to reach the overall structure again.

3.1. Overview of Message Structure

   In general a message is composed of three parts: the identification,
   the command, and the document. Each part is in turn composed of
   message objects.

   The identification part is composed of a transaction number assigned
   by the originating MPM, and the internet host number of that MPM.

   The command part is composed of an operation type, an operation code,
   an argument list, an error list, the destination mailbox, and a stamp.
   The stamp is a list of the MPMs that have handled this message.

   The document part is composed of a header and a body. The message
   delivery system does not depend on the contents of the document part,
   but this specification does make some recommendations for the document
   header.

   The following sections define the representation of a message as a
   structured object composed of other objects. Objects in turn are
   represented using a set of basic data elements.

3.2. Data Elements

   The data elements defined here are similar to the data structure and
   encoding used in NSW [18].

   Each of the diagrams which follow represent a sequence of octets.
   Field boundaries are denoted by the "!" character, octet boundaries by
   the "+" character. The diagrams are presented in left to right order.
   Each element begins with a one octet code.

```
        Code  Type           Representation
        ----  ----           --------------


                             +------+
          0  No Operation    !  1   !
                             +------+


                             +------+------+------+------+------
          1  Padding         !  0   !    octet count    ! Data ...
                             +------+------+------+------+------


                             +------+------+
          2  Boolean         !  2   ! 1/0  !
                             +------+------+


                             +------+------+------+
          3  Index           !  3   !    Data    !
                             +------+------+------+


                             +------+------+------+------+------+
          4  Integer         !  4   !           Data          !
                             +------+------+------+------+------+


                             +------+------+------+------+------
          5  Bit String      !  5   !     bit count     ! Data ...
                             +------+------+------+------+------


                             +------+------+------+------+------
          6  Text String     !  6   !    octet count    ! Data ...
                             +------+------+------+------+------


                             +------+------+------+------+------+------+-----
          7  List            !  7   !    octet count    ! item count ! Data
                             +------+------+------+------+------+------+-----


                             +------+------+------+------+------
          8  Proplist        !  8   !    octet count    ! Data ...
                             +------+------+------+------+------
```

Element code 0 (NOP) is an empty data element used for padding when it is necessary. It is ignored.

Element code 1 (PAD) is used to transmit large amounts of data with a message for test or padding purposes. No action is taken with this data but the count of dummy octets must be correct to indicate the next element code.

Element code 2 (BOOLEAN) is a boolean data element which has the value 1 for True and 0 for False.

Element code 3 (INDEX) is a 16-bit unsigned integer datum. Element code 3 occupies only 3 octets.

Element code 4 (INTEGER) is a signed 32-bit integer datum. This will always occupy five octets. Representation is two's complement.

Element code 5 (BITSTR) is a bit string element for binary data. The bit string is padded on the right with zeros to fill out the last octet if the bit string does not end on an octet boundary. This data type must have the bit-count in the two octet count field instead of the number of octets.

Element code 6 (TEXT) is used for the representation of text. Seven bit ASCII characters are used, right justified in the octet. The high order bit in the octet is zero.

Element code 7 (LIST) can be used to create structures composed of other elements. The item-count contains the number of elements which follow. Any element may be used including List itself. The octet count specifies the number of octets in the whole list. A null or empty List, one with no elements, has an item-count of zero (0).

   Element code 8 (PROPLIST) is the Property-List element. It has the
   following form:

      +------+------+------+------+------+
      ! 8 !   octet   ! pair !
      !   !   count   ! count!
      +------+------+------+------+------+
                     +------+------+------+---------+---------+
                     ! name !   value   ! name   ! value   !
            repeated ! count!   count   !   ...! !   ...! !
                     +------+------+------+---------+---------+

   The Property-List structure consists of a set of unordered name/value
   pairs. The pairs are a one octet name count and a two octet value
   count followed by the name and value strings. The counts specify the
   length in octets of the name and value strings. Each string has a
   length in octets which agrees with its respective count. The count of
   octets until the next pair in the property list is 1 + 2 + name count
   + value count octets. The entire Property-List is of course equal in
   length to the octet count of the element itself. Immediately
   following the octet count for the entire element is a one octet pair
   count field which contains the total number of name/value pairs in the
   Proplist.

3.3. Message Objects

   In the composition of messages we use a set of objects such as
   address, or date. These objects are encoded in the basic data
   elements. The message objects are built of data elements.

   While data elements are typed, message objects are not. This is
   because messages are structured to the extent that only one kind of
   message object may occur in any position of a message structure.

   The following is a list of some of the objects used in messages. The
   object descriptions are grouped by the section of the message in which
   they normally occur.

   Identification

     Internet Host Number (ihn)

       This identifies a host in the internetwork environment. When used
       as a part of tid, it identifies the originating host of a message.
       The ihn is a 32 bit number, the higher order 8 bits identify the
       network, and the lower order 24 bits identify the host on that
       network.

          INTEGER

     Transaction Identifier (tid)

       This is the transaction identifier associated with a particular
       command. It is a list of the transaction number and the internet
       host number of the originating host.

          LIST (tn , ihn)

     Transaction Number (tn)

       This is a number which is uniquely associated with this
       transaction by the originating host. It identifies the
       transaction. (A transaction is a message and acknowledgment, this
       is discussed in more detail in later sections.) A tn must be
       unique for the time which the message (a request or reply)
       containing it could be active in the network.

          INDEX

   Command

     Address

       This is very similar to Mailbox in that it also is the "address"
       of a user. However, Address is intended to contain the minimum
       information necessary for delivery, and no more.

          PROPLIST (---)

     Answer

       A yes (true) or no (false) answer to a question.

          BOOLEAN

Arguments

   This is the argument to many of the operations. It consists of a
   List of different data types. The List will have form and data
   relevant with the particular operation.

   LIST (---)

Command-Type

   Gives the type of a command (e.g., request, reply, alarm).

   INDEX

Error-List

   The error list contains information concerning an error which has
   occured. It is a List comprised of the two objects error-class
   and error-string.

   LIST (error class, error string)

Error-Class

   A code for the class of the error.

   INDEX

Error-String

   A text string explaining the error.

   TEXT

How-Delivered

   A comment on the delivery of a messages, for instance a message
   could be delivered, forwarded, or turned over to general delivery.

   LIST (TEXT)

       Mailbox

         This is the "address" of a user of the internetwork mail system.
         Mailbox contains information such as net, host, location, and
         local user-id of the recipient of the message. Some information
         contained in Mailbox may not be necessary for delivery.

         As an example, when one sends a message to someone for the first
         time, he may include many items which are not necessary simply to
         insure delivery. However, once he gets a reply to this message,
         the reply could contain an Address (as opposed to Mailbox) which
         the user will use from then on.

           A mailbox is a PROPLIST. A mailbox might contain the following
           name-value pairs:

             name   element description
             ----   ------- -----------
             IA   INTEGER internet address
             NET   TEXT   network name
             HOST   TEXT   host name
             USER   TEXT   user name
             CITY   TEXT   city
             COUNTRY   TEXT   country
             STATE   TEXT   state
             ZIP   TEXT   zip code
             PHONE   TEXT   phone number

         PROPLIST (---)

       Operation

         This names the operation or procedure to be performed.

           TEXT

       Options

         REGULAR for normal delivery, FORWARD for message forwarding,
         GENDEL for general delivery, or other options which may be defined
         later.

         LIST (TEXT, ...)

Reasons

These could be mailbox does not exist, mailbox full, etc.

LIST (TEXT)

Stamp

Each MPM that handles the message must add a unique identifier (ihn, see above) to the list. This will prevent messages from being sent back and forth through the internet mail system without eventually either being delivered or returned to the sender.

LIST (ihn, ihn, ...)

Trail

When a message is sent through the internetwork environment, it acquires a list of MPMs that have handled the message in "Stamp". This list is then carried as "Trail" upon reply or acknowledgment of that message. More simply, requests and replies always have a "Stamp" and each MPM adds its ihn to this "Stamp." Replies, in addition, have a "Trail" which is the complete "Stamp" of the original message.

LIST (ihn, ihn, ...)

Type

The command type, e.g., request or reply.

INDEX

Document

In this section, we define some objects useful in message document headers. The ones we use are taken from the current ARPANET message syntax standard [6,8].

CC

When copies of a message are sent to others in addition to the addresses in the To object, those to whom the copies are sent will have their addresses recorded here. CC will be a single TEXT element.

TEXT

   Date

      The date and time are represented according to the International
      Standards Organization (ISO) recommendations [13,14,15]. Taken
      together the ISO recommendations 2014, 3307, and 4031 result in
      the following representation of the date and time:

        yyyy-mm-dd-hh:mm:ss,fff+hh:mm

      Where yyyy is the 4 digit year, mm is the two digit month, dd is
      the two digit day, hh is the two digit hour in 24 hour time, mm is
      the two digit minute, ss is the two digit second, and fff is the
      decimal fraction of the second. To this basic date and time is
      appended the offset from Greenwich as plus or minus hh hours and
      mm minutes.

      TEXT

   Document-Body

      The document body will contain that portion of the message
      commonly thought of as the text portion. It will be composed of a
      list of elements. This will allow transmission of data other than
      pure text if such capabilities are needed. We can, for instance,
      envision digital voice communication through the transmission of
      BITSTR element, or transmission of graphic data, etc. Information
      regarding control of such features could be included in the header
      for cooperating sites, or in the body itself but such protocols
      would depend upon agreement among those sites involved. It is
      expected of course that the majority of messages will contain body
      portions comprised of TEXT elements.

      LIST (---)

   Document-Header

      The document header contains the memo header presented to the
      user. In principle this may be of any style or structure. In
      this specification it is recommended that a PROPLIST be used and
      that the name-value pairs correspond to the header fields of
      RFC 733 [6].

      PROPLIST (---)

From

   The From is meant to be the name of the author of a document. It
   will be one TEXT element.

      TEXT

Reply-To

   Sometimes it will be desired to direct the replies of a message to
   some address other than the From or the Sender. In such a case
   the Reply-To object can be used.

      TEXT

Sender

   The Sender will contain the address of the individual who sent the
   message. In some cases this is NOT the same as the author of the
   message. Under such a condition, the author should be specified in
   the From object. The Sender is a single TEXT element.

      TEXT

Subject

   The subject of the message.

      TEXT

To

   To identifies the addressees of the message. The To object is one
   TEXT element.

      TEXT

   3.4. Command

     This section describes the commands which processes in the internet
     message system can use to communicate. Several aspects of the command
     structure are based on the NSW Transaction Protocol [19]. The
     commands come in pairs, with each request having a corresponding
     reply.

      A command is a list:

        LIST (mailbox, stamp, type, operation, arguments, error-list)

     The arguments are described generally here and more specifically, if
     necessary, in the description of each command.

        mailbox: PROPLIST

          This is the "to" specification of the message. Mailbox takes the
          form of a property list of general information, some of which is
          the essential information for delivery, and some of which could be
          extra information which may be helpful for delivery. Mailbox is
          different from address in that address is a very specific list
          without extra information.

        stamp: LIST (INTEGER, ...)

          This is a list of the MPMs that have handled the message. Each
          MPM must add its 32 bit Internet Host Number (ihn) to the LIST.

        type: INDEX

          type=1 a REQUEST operation.

          type=2 a REPLY operation.

          type=3 an ALARM operation. (A high priority message.)

          type=4 a RESPONSE to an alarm operation.

        operation: TEXT

          Operation is the name of the operation or procedure to be
          performed. This string must be interpreted in an upper/lower case
          independent manner.

     arguments: LIST

        This is a list of arguments to the above operation.

     error-list: LIST

        If message is type 1 or 3 (a request or an alarm):

          LIST ( ) (a zero length list)

        If message is a type 2 or 4 (a response or response to alarm)

          LIST (error-class, error-string) indicates what,if any, error
          occured

        error-class: INDEX

          =0: indicates success, no error
          =1: partial results returned.
            This error class is used when several steps are performed by
            one operation and some of them fail.
          =2: failure, resources unavailable.
          =3: failure, user error.
          =4: failure, MPM error. Recoverable.
          =5: failure, MPM error. Fatal.
          =6: User abort requested

        error-string: TEXT

          This is a human readable character string describing the error.

      Possible errors:

               error-string   error-class

        No errors   0
        Command not implemented   2
        Syntax error, command unrecognized   3
        Syntax error, in arguments   3
        Server error, try again later   4
        No service available   5
        User requested abort   6

   command: DELIVER

      type: 1

      function: Sends message to a mailbox

      reply: The reply is ACKNOWLEDGE

      arguments: LIST (options)

        options: one or more of the following

           "REGULAR" regular delivery

           "FORWARD" message forwarding

           "GENDEL" general delivery

           other options which may be defined later

      argument structure:

        LIST (LIST (TEXT, ...))

  command: ACKNOWLEDGE

    type: 2

    function: reply to DELIVER

    arguments: LIST (tid, trail, answer, reasons, how-delivered)

      tid: tid of the originating message

      trail: the stamp from the deliver command

      answer: yes if delivered successfully,
               no if error in delivery.

      reasons: if the answer is yes, the reason is "ok", if the answer
      is no the reason could be one of "no such user", "no such host",
      "no such network", "address ambiguous", or a similar response

      how-delivered: one or more of the following:

        "FORWARD" message was accepted for forwarding

        "GENDEL"   message was accepted for general delivery

        "ACCEPT"   message was accepted for normal delivery

        other types of delivery may be defined later

    argument structure:

      LIST (LIST (INDEX, INTEGER),
             LIST (INTEGER, ...),
             BOOLEAN,
             LIST (TEXT),
             LIST (TEXT))

command: PROBE

   type: 1

   function: finds out if specified mailbox (specified in mailbox of
   the command) exists at a host

   reply: the reply is RESPONSE

   arguments: LIST (--none--)

   argument structure:

     LIST ( )

     command: RESPONSE

        type: 2

        function: reply to PROBE

        arguments: LIST (tid, trail, answer, address OR reasons)

           tid: the tid which came from the originating PROBE

           trail: the stamp which came from the originating PROBE

           answer: Yes if mailbox found, or no for invalid mailbox

           if answer is yes the fourth argument is address
           if answer is no it is reasons

           address: a specific address in the network

           reasons: a reason why mailbox is invalid

             Possible reasons include:

                "Mailbox doesn't exist"

                "Mailbox full"

                "Mailbox has moved, try this new location", address

                  address is a new address to try

        argument structure:

           if answer is yes

              LIST (LIST (INDEX, INTEGER),
                     LIST (INTEGER, ...),
                     BOOLEAN,
                     PROPLIST)

           if answer is no

              LIST (LIST (INDEX, INTEGER),
                     LIST (INTEGER, ...),
                     BOOLEAN,
                     LIST (TEXT))

       command: CANCEL

         type: 3

         function: abort request for specified transaction

         reply: The reply is CANCELED

         arguments: LIST (tid)

            tid of transaction to be cancelled

         argument structure:

            LIST (LIST (INDEX, INTEGER))

    command: CANCELED

       type: 4

       function: reply to CANCEL

       arguments: LIST (tid, trail, answer)

          tid: tid of transaction to be cancelled

          trail: the stamp of the CANCEL command

          answer: yes if the command was canceled, no if not.

       argument structure:

          LIST (LIST (INDEX, INTEGER),
                 LIST (INTEGER, ...),
                 BOOLEAN)

   To summarize again, a command consists of a LIST of the following
   objects:

```
   name          element
   ----          -------
   mailbox       PROPLIST
   stamp         LIST ( INTEGER, ... )
   type          INDEX
   operation     TEXT
   arguments     LIST ( --- )
   error         LIST ( INDEX, TEXT )
```

3.5. Document

  The actual document follows the command list. It contains a header
  which usually contains such information as From, To, Date, CC, etc.;
  and the actual body of the message. The message delivery system does
  not depend on the document. The following section should be taken as
  a recommendation for common practice, not as a requirement.

  Document Header

    For the same reason that it is impossible to for see the many forms
    that intranet addresses will take, standardizing of document headers
    would also be a mistake. The approach we suggest is to lay the
    groundwork for a set of basic document header functions and provide
    for enough extensibility to allow nets to add whatever header
    features they desire. Features added in this fashion, however, may
    not be understood by other networks. It is suggested that subset
    defined here be implemented by all networks.

    This subset is taken from the current ARPANET standard for message
    headers in the text oriented computer message system [6,8].

    The document header will precede the document body portion of the
    message and will consist of a proplist data element. The document
    header is meant to be used by individual networks to tailor the
    header to suit their individual needs. As an example, consider the
    ARPA network. Typically, the receiver's name is taken to be his
    network address. It often prints in the document header in just
    that form: Frank@SITEX. Such a salutation is unacceptable in some
    more formal modes of communication. Some network might choose to
    place into header proplist the name-value pair ("SALUTATION:", "Mr.
    Frank Hacker"). Upon receipt of the message, the document handling
    program would then be able to scan the header proplist looking for
    such a pair and so be able to correctly address the recipient by
    name instead of by network address. However, other networks or

sites within the network may not understand such specific
information. Under such a condition it should be ignored.

The minimum header is a PROPLIST of the following name-value pairs:

```
   Name       Value
   ----       -----
   DATE       TEXT
   FROM       TEXT
```

A normal header is a PROPLIST containing the following name-value
pairs:

```
   Name       Value
   ----       -----
   DATE       TEXT
   SENDER     TEXT
   FROM       TEXT
   TO         TEXT
   CC         TEXT
   SUBJECT    TEXT
```

Document Body

The Body of the message is just a sequence of data elements which
contains the actual document. Much of the time this will be a
single TEXT element, but for some applications other data elements
may be utilized.

LIST (---)

   3.6. Message Structure

      An internet message is composed of three parts. The first is the tid
      which identifies the transaction; the second is the Command List; and
      the third part is the Document List, which is itself comprised of a
      Document-Header and a Document-Body.

      When shipped between two MPMs, a message will take the form of a LIST:

         Message is:

           LIST (tid, Command-List, Document-List)

      It is convenient to batch several messages together shipping them as
      a unit from one MPM to another. Such a group of messages is called
      a message-bag.

      A message-bag will be a LIST of Messages, each Message is of the
      form described above.

         Thus, a message-bag is:

           LIST (Message1, Message2, ...)

    Message Sharing

      When messages are batched for delivery, it may often be the case
      that the same Document will be sent to more than one recipient.
      Since the Document portion can usually be expected to be the major
      parts of the message, much repeated data would be sent if a copy of
      the Mail for each recipient were to be shipped in the message-bag.

      To avoid this redundancy, messages are assembled in the message-bag
      so that actual data appears first and references to it appear later
      in the message-bag. Since each message has a unique tid, the
      references will indicate the tid of the actual data. In this sense,
      all references to copied data may be thought of as pointing earlier
      in the message-bag. The data to be retrieved can be thought of as
      indexed by tid. Note that the semantics require such references to
      point to data already seen.

      When a portion is Shared, that portion is determined by its position
      within a message, i.e., if the Command list was to be Shared, then
      its position within a Message would contain the tid of the message
      already seen whose Command list was identical to it. The same is
      true of the Document Header and the Document Body. Only a complete
      Command, Header, or Body may be Shared, never a partial one.

If an encryption scheme is used, that portion of the message which
is encrypted can not be shared. This is due to the fact that
encrypting keys will be specific between two individuals.

  Internal Message Organization

    The tid

      This is the transaction identifier. It is assigned by the
      originating MPM.

    The Command List

      The command-list is a LIST which contains two elements, content
      and command.

      Content is one item of element type INDEX. If content=0, the item
      is not shared and the next element of the LIST is the command. If
      content=1 the item is shared. In this case, the second element
      will contain the tid of the command to share from. The tid must
      be of a prior message in the current message-bag. Other values of
      content may be defined later for different data structures.

      Thus, command-list is:

        LIST (content, tid)   if content=1

      Or,

        LIST (content, command)   if content=0

      content is:

        INDEX   which is 0 if there is no sharing
                   and is 1 if sharing occurs

      tid is:

        the tid of the message to be shared from

      command is:

        LIST (mailbox, stamp, type, operation, arguments, error-list)

    The document-list

      The document portion of an internet message is optional and when
      present is comprised of a LIST containing two elements:

   document-list is:

      LIST (header-list, body-list)

  While either the header-list or the body-list may be shared, both
  elements must appear in the m.

 The document-header

  The header-list will be a List which will always contain two
  elements. The first element will be content to indicate whether
  or not the header is to be shared. The second element will either
  be the tid of the header to be copied (if content=1) or it will be
  the document-header (which is a PROPLIST) containing the actual
  header information (if content=0). The tid must point to a
  document-header already seen in the message-bag.

  The header-list is either:

    LIST (content, tid)   if content=1

  Or,

    LIST (content, document-header)   if content=0

  document-header is:

    PROPLIST which contains header information

 The document-body

  The body-list will be a LIST of two elements. The first element
  will again be content, indicating whether or not the body is to be
  shared. If it is shared, the second element will be tid
  indicating which body to copy. This tid must be of a message
  already seen in the message-bag. If content indicates no sharing,
  then the second item is a document-body.

  body-list is:

    LIST (content, tid)   if content=1

  Or,

    LIST (content, document-body) if content=0

        document-body is:

          LIST (items comprising the body ...)

  Message Fields

    message := (tid, command-list, document-list)

    tid := (tn, ihn)

    command-list := (content, command)

    command := (mailbox, stamp, type, operation,
                 arguments, error-list)

    document-list := (header-list, body-list)

    header-list := (content, document-header)

    body-list := (content, document-body)

3.7. MPM Organization

  Introduction

    The heart of the internet message system is the MPM which is
    responsible for routing and delivering message between the networks.
    Each network must have at least one MPM. These MPMs are connected
    together, and internet mail is always transferred along channels
    between them. The system interfaces with the already existent local
    message system.

    Since the local network message system may be very different from
    the internet system, special programs may be necessary to convert
    incoming internet messages to the local format. Likewise, messages
    outgoing to other networks may be converted to the internet format.

  The MPM

    Messages in the internet mail system are shipped in "bags," each bag
    containing one or more messages. Each bag is addressed to a
    specific MPM and contains messages for the hosts on that MPM's
    network.

    Each MPM is expected to implement functions which will allow it to
    deliver local messages it receives and to forward non-local ones to
    other MPMs presumably closer to the message's destination.

    Loosely, each MPM can be separated into five components:

       1--Acceptor

         Receives incoming Message-Bags, from other MPMs, from UIPs, or
         from conversion programs.

       2--Message-Bag Processor

         Splits a Bag into these three portions:

            a.   Local Host Messages
            b.   Local Net Messages
            c.   Foreign Net Messages

       3--Local Net Delivery

         Delivers local net and local host messages, may call on
         conversion program.

       4--Foreign Net Router

         Creation of new Message-Bags for forwarding to other MPMs,
         determines route.

       5--Foreign Net Shipper

         Activates foreign shipping channels and ships Message-Bag to
         foreign MPMs. Performs data compression while shipping bags.

    All of these components can be thought of as independent. Of the
    five, the Acceptor, the Local-Net Delivery, and the Message-Bag
    Processor are fully self-contained and communicate with each other
    only through a queue, the Bag-Input Queue. The function of the
    Acceptor is to await incoming Message-Bags and to insert them into
    the Bag-Input Queue.

    That queue is the input to the Message-Bag Processor component which
    will separate and deliver suitable portions of the Message-Bags it
    retrieves from the queue to one of three queues:

       a.   Local-Host Queue
       b.   Local-Net Queue
       c.   Foreign Net Queue

    When a MPM decides to forward a message to another MPM, it must add
    its own identification (i.e., its ihn) to the stamp field of the
    command.   The stamp then becomes a record of the route the message

has taken. An MPM should examine the stamp field to see if the
message is in a routing loop. Some commands require the return of
the stamp as a trail in the matching reply command.

All of these queues have as elements complete Message-Bags (some of
which may have been portions of the original Bag).

The Local-Host and Local-Net queues serve as input to the Local-Net
Delivery process. This component is responsible for delivering
messages to its local host and other hosts on its local net to which
it is connected. It must be capable of handling whatever error
conditions the local net might return, including the ability to
retransmit. It may call on conversion program to reformat the
messages into a form the local protocol will accept. This will
probably involve such things as copying shared information.

The other two processes are more closely coupled. The Foreign Net
Router takes its input Bags from the Foreign Net Queue. From the
internal information it contains, it determines which one of the
MPMs to which it is connected should receive the Bag.

It then places the Bag along with the routing information into the
Shippable Mail Queue. The Foreign Net Shipper retrieves it from
that queue and transmits it across a channel to the intended foreign
MPM.

The Foreign Net Router should be capable of receiving external input
to its routing information table. This may come from the Foreign
Net Shipper in the case of a channel going down, requiring a
decision to either postpone delivery or to determine a new route.

The Router is responsible for maintaining sufficient topological
information to determine where to forward any incoming Message-Bag.
Decisions concerning the return of undeliverable Bags are made by
the Router.

It should be stressed here that message delivery should be reliable.
In the event that delivery is impossible, the message should be
returned to the sender along with information regarding the reason
for not delivering it.

  Implementation Recommendations

  Transaction numbers can be assigned sequentially with wrap around
  when the highest value is reached. This should ensure that no
  message with a particular transaction number from this source is in
  the network when another instance of this transaction number is
  chosen.

   3.8. Interfaces

     User Interface

        It is assumed that the interface between the MPM and the UIP
        provides for passing data structures which represent the document
        portion of the message. In addition this interface must pass the
        delivery address information (which becomes the information in the
        mailbox field of the command). It is weakly assumed that the
        information is passed between the UIP and the MPM via shared files,
        but this is not the only possible mechanism. These two processes
        may be more strongly coupled (e.g., by sharing memory), or less
        strongly coupled (e.g., by communicating via logical channels).

     Communication Interface

        It is assumed here that the MPM use an underlying communication
        system, and TCP [20] has been taken as the model. Again, this is
        not intended to limit the implementation choices, other forms of
        interprocess communication are allowed and other types of physical
        interconnection are permitted. One might even use dial telephone
        calls to interconnect MPMs (using suitable protocols to provide
        reliable communication).

Internet Message Protocol

                    4.   EXAMPLES & SCENARIOS

Example 1: Message Format

   Suppose we want to send the following message:

      Date: 1979-03-29-11:46-08:00
      From: Jon Postel <Postel@ISIB>
      Subject: Meeting Thursday
      To: Dave Crocker <DCrocker@Rand-Unix>
      CC: Mamie

      Dave:

      Please mark your calendar for our meeting Thursday at 3 pm.

      --jon.

   It will be encoded in the structured format. The following will
   present successive steps in the top down generation of this message.

      1.   message

      2.   (tid, command-list, document-list)

      3.   ((tn, ihn),
           (content, command),
           (header-list, body-list))

      4.   ((tn, ihn),
             (content,
               (mailbox, stamp, type, operation,
                 arguments, error-list)),
             ((content, document-header),
               (content, document-body)))

      5.   ((37, 167772404),
           (0, (
                 (IA: 167772359, NET: arpa, HOST: rand-unix,
                   USER: DCrocker),
                 (167772404),
                 1
                 DELIVER
                 ((REGULAR)),
                 ( ))),
           ((0, (
                 Date: 1979-03-29-11:46-08:00
                 From: Jon Postel <Postel@ISIB>
                 Subject: Meeting Thursday

                    To: Dave Crocker <DCrocker@Rand-Unix>
                    CC: Mamie)),
             (0, (Dave:

                    Please mark your calendar for our meeting
                    Thursday at 3 pm.

                    --jon.))))

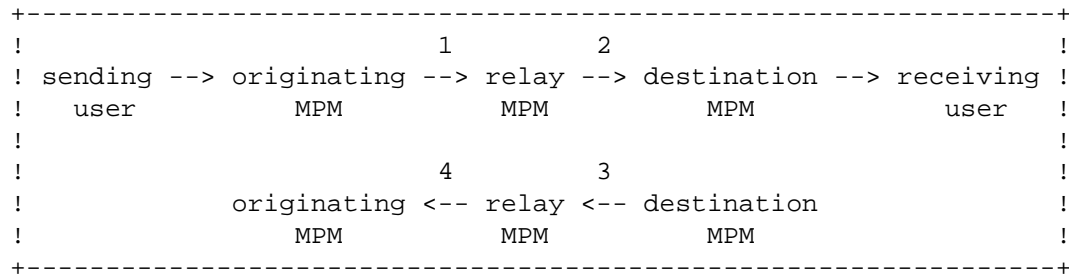   6. LIST(LIST(INDEX=37, INTEGER=167772404),
             LIST(INDEX=0,
     command   LIST(PROPLIST(IA: 167772359,
                                        NET: arpa,
     mailbox   HOST: rand-unix,
                                        USER: DCrocker),
     stamp   LIST(INTEGER=167772404),
     type   INDEX=1
     operation   TEXT="DELIVER"
     arguments   LIST(LIST(TEXT="REGULAR")),
     error-list   LIST( ))),
             LIST(LIST(INDEX=0,
     document-header   PROPLIST(
                             DATE: 1979-03-29-11:46-08:00
                             FROM: Jon Postel <Postel@ISIB>
                             SUBJECT: Meeting Thursday
                             TO: Dave Crocker <DCrocker@Rand-Unix>
                             CC: Mamie)),
                    LIST(INDEX=0,
     document-body   LIST(TEXT=
                             "Dave:

                             Please mark your calendar for
                             our meeting Thursday at 3 pm.

                             --jon."))))

Example 2: Delivery and Acknowledgment

   The following is four views of the message of example 1 during the
   successive transmission from the origination MPM, through a relay MPM,
   to the destination MPM, and the return of the acknowledgment, through
   a relay MPM, to the originating MPM.

```
   +---------------------------------------------------------------+
   !                        1         2                            !
   ! sending --> originating --> relay --> destination --> receiving !
   !   user          MPM          MPM          MPM          user   !
   !                                                               !
   !                        4         3                            !
   !          originating <-- relay <-- destination               !
   !                 MPM          MPM          MPM                 !
   +---------------------------------------------------------------+
```

                         Transmission Path

                           Figure 6.

   1. Between the originating MPM and the relay MPM.

```
        LIST( LIST( INDEX=37, INTEGER=167772404 ),
             LIST( INDEX=0,
  command         LIST( PROPLIST( IA: 167772359,
                                  NET: arpa,
  mailbox                         HOST: rand-unix,
                                  USER: DCrocker ),
  stamp                  LIST( INTEGER=167772404 ),
  type                   INDEX=1
  operation              TEXT="DELIVER"
  arguments              LIST( LIST( TEXT="REGULAR" )),
  error-list             LIST( ) ) ),
             LIST( LIST( INDEX=0,
  document-header         PROPLIST(
                            DATE: 1979-03-29-11:46-08:00
                            FROM: Jon Postel <Postel@ISIB>
                            SUBJECT: Meeting Thursday
                            TO: Dave Crocker <DCrocker@Rand-Unix>
                            CC: Mamie ) ),
                  LIST( INDEX=0,
  document-body          LIST( TEXT=
                            "Dave:

                            Please mark your calendar for
                            our meeting Thursday at 3 pm.

                            --jon." ) ) ) )
```

   The originating MPM sends the message of example 1 to a relay MPM.

   2. Between the relay MPM and the destination MPM.

          LIST(LIST(INDEX=37, INTEGER=167772404),
              LIST(INDEX=0,
     command   LIST(PROPLIST(IA: 167772359,
                                    NET: arpa,
     mailbox   HOST: rand-unix,
                                    USER: DCrocker),
     stamp   LIST(INTEGER=167772404,
                             INTEGER=167772246),
     type   INDEX=1
     operation   TEXT="DELIVER"
     arguments   LIST(LIST(TEXT="REGULAR")),
     error-list   LIST( ))),
              LIST(LIST(INDEX=0,
     document-header   PROPLIST(
                          DATE: 1979-03-29-11:46-08:00
                          FROM: Jon Postel <Postel@ISIB>
                          SUBJECT: Meeting Thursday
                          TO: Dave Crocker <DCrocker@Rand-Unix>
                          CC: Mamie)),
                 LIST(INDEX=0,
     document-body   LIST(TEXT=
                          "Dave:

                          Please mark your calendar for
                          our meeting Thursday at 3 pm.

                          --jon."))))

     The relay MPM adds its ihn to the stamp, but otherwise the message
     is unchanged.

   3. Between the destination MPM and the relay MPM.

```
            LIST( LIST( INDEX=1993, INTEGER=167772359 ),
                 LIST( INDEX=0,
  command           LIST( PROPLIST( IA: 167772404,
  mailbox                          USER: *MPM* ),
  stamp                 LIST( INTEGER=167772359 ),
  type                  INDEX=2
  operation             TEXT="ACKNOWLEDGE"
  arguments             LIST( LIST( INDEX=37,
   tid                             INTEGER=167772404 ),
                             LIST( INTEGER=167772404,
   trail                           INTEGER=167772246,
                                   INTEGER=167772359 ),
   answer                    BOOLEAN=TRUE,
   reason                    LIST( TEXT="OK" ),
   how-delivered             LIST( TEXT="ACCEPT" ) ),
  error-list          LIST( INDEX=0,
                            TEXT="No Errors") ),
  document  LIST( ) )
```

      The destination MPM delivers the message to the user's UIP, and
      composes an acknowledgment. The acknowledgment is addressed to
      the originating MPM. Note that the trail is the stamp of the
      incoming message plus the ihn of the destination MPM.

4. Between the relay MPM and the originating MPM.

```
      LIST( LIST( INDEX=1993, INTEGER=167772359 ),
            LIST( INDEX=0,
  command          LIST( PROPLIST( IA: 167772404,
  mailbox                          USER: *MPM* ),
  stamp                   LIST( INTEGER=167772359
                                INTEGER=167772246),
  type                    INDEX=2
  operation               TEXT="ACKNOWLEDGE"
  arguments               LIST( LIST( INDEX=37,
   tid                              INTEGER=167772404 ),
                                LIST( INTEGER=167772404,
   trail                            INTEGER=167772246,
                                    INTEGER=167772359 ),
   answer                    BOOLEAN=TRUE,
   reason                    LIST( TEXT="OK" ),
   how-delivered             LIST( TEXT="ACCEPT" ) ),
  error-list        LIST( INDEX=0,
                          TEXT="No Errors") ),
  document  LIST( ) )
```

The relay MPM adds its ihn to the stamp and forwards the
acknowledgment.

                              GLOSSARY

     1822
               BBN Report 1822, "The Specification of the Interconnection of
               a Host and an IMP". The specification of interface between a
               host and the ARPANET.

     Command List
               The part of a message used by the MPMs to determine the
               processing action to be taken.

     datagram
               A logical unit of data, in particular an internet datagram is
               the unit of data transfered between the internet module and a
               higher level module.

     Destination
               The destination address, an internet header datagram protocol
               field.

     Document List
               The part of the message created by or delivered to a user.

     header
               Control information at the beginning of a message, segment,
               datagram, packet or block of data.

     IMP
               The Interface Message Processor, the packet switch of the
               ARPANET.

     Internet Address
               A four octet (32 bit) source or destination address consisting
               of a Network field and a Local Address field.

     internet datagram
               The unit of data exchanged between a pair of internet modules
               (includes the internet header).

     Local Address
               The address of a host within a network. The actual mapping of
               an internet local address on to the host addresses in a
               network is quite general, allowing for many to one mappings.

message

        The unit of information transmitted between users of message
        systems. As transmitted between MPMs a message consists of a
        Transaction Identifier, a Command List, and a Document List.

module

        An implementation, usually in software, of a protocol or other
        procedure.

MPM

        A Message Processing Module, the process which implements this
        internet message protocol.

octet

        An eight bit byte.

Rest

        The 3 octet (24 bit) local address portion of an Internet
        Address.

RTP

        Real Time Protocol: A host-to-host protocol for communication
        of time critical information.

Source

        The source address, an internet header field.

TCP

        Transmission Control Protocol: A host-to-host protocol for
        reliable communication in internetwork environments.

Transaction Identifier

        The unique identifier of a message.

Type of Service

        An internet datagram protocol header field which indicates the
        type (or quality) of service for this internet packet.

UIP

        A User Interface Program, a program which presents message
        data to a user and accepts message data from a user. A
        program which interacts with the user in the composition and
        examination of messages.

XNET

        A cross-net debugging protocol.

REFERENCES

[1]   Barber, D., and J. Laws, "A Basic Mail Scheme for EIN," INWG 192,
        February 1979.

[2]   Bhushan, A., K. Progran, R. Tomlinson, and J. White,
        "Standardizing Network Mail Headers," RFC 561, NIC 18516, 5
        September 1973.

[3]   Bolt Beranek and Newman, "Specification for the Interconnection of
        a Host and an IMP," BBN Technical Report 1822, May 1978 (Revised).

[4]   Braaten, O., "Introduction to a Mail Protocol," Norwegian
        Computing Center, INWG 180, August 1978.

[5]   Cerf, V., "The Catenet Model for Internetworking," Information
        Processing Techniques Office, Defense Advanced Research Projects
        Agency, IEN 48, July 1978.

[6]   Crocker, D., J. Vittal, K. Progran, and D. Henderson, "Standard
        for the Format of ARPA Network Text Messages," RFC 733, NIC 41952,
        21 November 1977.

[7]   Crocker, D., E. Szurkowski, and D. Farber, "Components of a
        Channel-independent Memo Transmission System," Department of
        Electrical Engineering, University of Delaware,, February 1979.

[8]   Feinler, E. and J. Postel, eds., "ARPANET Protocol Handbook,"
        NIC 7104, for the Defense Communications Agency by the Network
        Information Center of SRI International, Menlo Park, California,
        Revised January 1978.

[9]   Harrenstien, K., "Field Addressing," ARPANET Message, SRI
        International, October 1977.

[10]   Haverty, J., "MSDTP -- Message Services Data Transmission
        Protocol," RFC 713, NIC 34739, April 1976.

[11]   Haverty, J., "Thoughts on Interactions in Distributed Services,"
        RFC 722, NIC 36806, 16 September 1976.

[12]   Haverty, J., D. Henderson, and D. Oestreicher, "Proposed
        Specification of an Inter-site Message Protocol," 8 July 1975.

[13]   ISO-2014, "Writing of calendar dates in all-numeric form,"
        Recommendation 2014, International Organization for
        Standardization, 1975.

[14]   ISO-3307, "Information Interchange -- Representations of time of
       the day," Recommendation 3307, International Organization for
       Standardization, 1975.

[15]   ISO-4031, "Information Interchange -- Representation of local time
       differentials," Recommendation 4031, International Organization
       for Standardization, 1978.

[16]   Myer, T., and D. Henderson, "Message Transmission Protocol,"
       RFC 680, NIC 32116, 30 April 1975.

[17]   Postel, J. "Internetwork Datagram Protocol, Version 4," USC
       Information Sciences Institute, IEN 80, February 1979.

[18]   Postel, J. "NSW Data Representation (NSWB8)," IEN 39, May 1978.

[19]   Postel, J. "NSW Transaction Protocol (NSWTP)," IEN 38, May 1978.

[20]   Postel, J. "Transmission Control Protocol, TCP, Version 4," USC
       Information Sciences Institute, IEN 81, February 1979.

[21]   Postel, J., "Assigned Numbers," RFC 750, NIC 45500,
       26 September 1978.

[22]   Postel, J., "Message System Transition Plan," JBP 64,
       USC-Information Sciences Institute, February 1979.

[23]   Rivest, R. L. "A Method for Obtaining Digital Signatures and
       Public-Key Cryptosystems" Communications of the ACM, Vol. 21,
       Number 2, February 1978.

[24]   Shoch, J., "A Note On Inter-Network Naming, Addressing, and
       Routing," Xerox Palo Alto Research Center, IEN 19, January 1978.

[25]   Thomas, R., "Providing Mail Services for NSW Users," BBN NSW
       Working Note 24, Bolt Beranek and Newman, October 1978.

[26]   White, J., "A Proposed Mail Protocol," RFC 524, NIC 17140, 13 June
       1973.

[27]   White, J., "Description of a Multi-Host Journal," NIC 23144,
       30 May 1974.

[28]   White, J., "Journal Subscription Service," NIC 23143, 28 May 1974.

March 1979

APPENDICES

A. Encryption

   It would be straightforward to add the capability to have the document
   portion of messages either wholly or partially encrypted. The
   approach is to define an additional basic data element to carry
   encrypted data. The data within this element could be composed of
   other elements, but that could only be perceived after the data was
   decrypted.


```
                    +------+------+------+------+-------
     9   Encrypt      ! 9   !     octet count    ! Data ...
                    +------+------+------+------+--------
```

   Element code 9 (ENCRYPT) is Encrypt. The format is the one octet type
   code, the three octet type count, and count octets of data. Use of
   this element indicates that the data it contains is encrypted. The
   encryption scheme is yet to be decided but will probably be the Public
   Key Encryption technique [23] due to the capacity for coded
   signatures.

   To process this, the user is asked for the appropriate key the first
   time an encryption block is seen for a particular message. The
   encrypted data is then decrypted. The data thus revealed will be in
   the form of complete data type fields. Encryption cannot occur over a
   partial field. The revealed data is then processed normally.

   Note that there is no reason why all fields of a document could not be
   encrypted including all document header information such as From,
   Date, etc.

B. Data Compression

   When message-bags are shipped between MPMs the data should be
   compressed according to the following scheme:

      shipping-unit := compression-type message-bag

      compression-type := A one octet compression type indicator.

         compression-type value   description
         ---------------------   -----------
                    0   no compression used
                    1   basic compression

      basic compression

         This basic compression procedure is the same as that defined for
         use with the ARPANET FTP [8]. Three types of compression-units
         may be formed, sequence-units, replication-units, and
         filler-units. The data is formed into a series of
         compression-units independent of the structure or object and
         element boundaries.

         sequence-unit

            A sequence-unit is a one octet flag and count followed by that
            many data octets.

               +-+-------+--------+--------+----
               !0!   n   !   n data octets ...
               +-+-------+--------+--------+----

            The flag and count octet has its high order bit zero and the
            remaining bits indicate the count (in the range 0 to 127) of
            following data octets.

         replication-unit

            A replication-unit is a one octet flag and count followed by one
            data octet, which is to be replicated count times.

               +--+------+--------+
               !10! n   ! data !
               +--+------+--------+

            The flag and count octet has its high order two bits equal
            one-zero and the remaining six bits indicate the count (in the
            range 0 to 63) of number of time to replicate the data octet.

   filler-unit

      A filler-unit is a one octet flag and count, indicating that a
      filler octet is to be inserted count times.

        +--+------+
        !11!   n !
        +--+------+

      The flag and count octet has its high order two bits equal
      one-one and the remaining six bits indicate the count (in the
      range 0 to 63) of number of time to insert the filler octet.

      The filler octet is zero, the octet with all bits zero.

Internet Message Protocol