Network Working Group
Request for Comments: 1064
M. Crispin
SUMEX-AIM
July 1988

INTERACTIVE MAIL ACCESS PROTOCOL - VERSION 2

Status of this Memo

This RFC suggests a method for workstations to dynamically access mail from a mailbox server ("repository"). This RFC specifies a standard for the SUMEX-AIM community and a proposed experimental protocol for the Internet community. Discussion and suggestions for improvement are requested. Distribution of this memo is unlimited.

Introduction

The intent of the Interactive Mail Access Protocol, Version 2 (IMAP2) is to allow a workstation or similar small machine to access electronic mail from a mailbox server. IMAP2 is the protocol used by the SUMEX-AIM MM-D (MM Distributed) mail system.

Although different in many ways from POP2 (RFC 937), IMAP2 may be thought of as a functional superset of POP2, and the POP2 RFC was used as a model for this RFC. There was a cognizant reason for this; RFC 937 deals with an identical problem and it was desirable to offer a basis for comparison.

Like POP2, IMAP2 specifies a means of accessing stored mail and not of posting mail; this function is handled by a mail transfer protocol such as SMTP (RFC 821). A comparison with the DMSP protocol of PCMAIL can be found at the end of "System Model and Philosophy" section.

This protocol assumes a reliable data stream such as provided by TCP or any similar protocol. When TCP is used, the IMAP2 server listens on port 143.

System Model and Philosophy

Electronic mail is a primary means of communication for the widely spread SUMEX-AIM community. The advent of distributed workstations is forcing a significant rethinking of the mechanisms employed to manage such mail. With mainframes, each user tends to receive and process mail at the computer he used most of the time, his "primary host". The first inclination of many users when an independent workstation is placed in front of them is to begin receiving mail at the workstation, and, in fact, many vendors have implemented facilities to do this. However, this approach has several disadvantages:

(1) Workstations (especially Lisp workstations) have a software design that gives full control of all aspects of the system to the user at the console. As a result, background tasks, like receiving mail, could well be kept from running for long periods of time either because the user is asking to use all of the machine's resources, or because, in the course of working, the user has (perhaps accidentally) manipulated the environment in such a way as to prevent mail reception. This could lead to repeated failed delivery attempts by outside agents.

(2) The hardware failure of a single workstation could keep its user "off the air" for a considerable time, since repair of individual workstation units might be delayed. Given the growing number of workstations spread throughout office environments, quick repair would not be assured, whereas a centralized mainframe is generally repaired very soon after failure.

(3) It is more difficult to keep track of mailing addresses when each person is associated with a distinct machine. Consider the difficulty in keeping track of a large number of postal addresses or phone numbers, particularly if there was no single address or phone number for an organization through which you could reach any person in that organization. Traditionally, electronic mail on the ARPANET involved remembering a name and one of several "hosts" (machines) whose name reflected the organization in which the individual worked. This was suitable at a time when most organizations had only one central host. It is less satisfactory today unless the concept of a host is changed to refer to an organizational entity and not a particular machine.

(4) It is very difficult to keep a multitude of heterogeneous workstations working properly with complex mailing protocols, making it difficult to move forward as progress is made in electronic communication and as new standards emerge. Each system has to worry about receiving incoming mail, routing and delivering outgoing mail, formatting, storing, and providing for the stability of mailboxes over a variety of possible filing and mailing protocols.

Consequently, while the workstation may be viewed as an Internet host in the sense that it implements IP, it should not be viewed as the entity which contains the user's mailbox. Rather, a mail server machine (sometimes called a "repository") should hold the mailbox, and the workstation (hereafter referred to as a "client") should access the mailbox via mail transactions. Because the mail server machine would be isolated from direct user manipulation, it could achieve high software reliability easily, and, as a shared resource, it could achieve high hardware reliability, perhaps through redundancy. The mail server could be used from arbitrary locations, allowing users to read mail across campus, town, or country using more and more commonly available clients. Furthermore, the same user may access his mailbox from different clients at different times, and multiple users may access the same mailbox simultaneously.

The mail server acts an an interface among users, data storage, and other mailers. The mail access protocol is used to retrieve messages, access and change properties of messages, and manage mailboxes. This differs from some approaches (e.g., Unix mail via NFS) in that the mail access protocol is used for all message manipulations, isolating the user and the client from all knowledge of how the data storage is used. This means that the mail server can utilize the data storage in whatever way is most efficient to organize the mail in that particular environment, without having to worry about storage representation compatibility across different machines.

In defining a mail access protocol, it is important to keep in mind that the client and server form a macrosystem, in which it should be possible to exploit the strong points of both while compensating for each other's weaknesses. Furthermore, it's desirable to allow for a growth path beyond the hoary text-only RFC 822 protocol. Unlike POP2, IMAP2 has extensive features for remote searching and parsing of messages on the server. For example, a free text search (optionally in conjunction with other searching) can be made throughout the entire mailbox by the server and the results made available to the client without the client having to transfer the entire mailbox and searching itself. Since remote parsing of a message into a structured (and standard format) "envelope" is available, a client can display envelope information and implement commands such as REPLY without having any understanding of how to parse RFC 822, etc. headers.

Additionally, IMAP2 offers several facilities for managing a mailbox beyond the simple "delete message" functionality of POP2.

In spite of this, IMAP2 is a relatively simple protocol. Although servers should implement the full set of IMAP2 functions, a simple client can be written which uses IMAP2 in much the way as a POP2 client.

IMAP2 differs from the DMSP protocol of PCMAIL (RFC 1056) in a more fundamental manner, reflecting the differing architectures of MM-D and PCMAIL. PCMAIL is either an online ("interactive mode"), or offline ("batch mode") system. MM-D is primarily an online system in which real-time and simultaneous mail access were considered important.

In PCMAIL, there is a long-term client/server relationship in which some mailbox state is preserved on the client. There is a registration of clients used by a particular user, and the client keeps a set of "descriptors" for each message which summarize the message. The server and client synchronize their states when the DMSP connection starts up, and, if a client has not accessed the server for a while, the client does a complete reset (reload) of its state from the server.

In MM-D, the client/server relationship lasts only for the duration of the IMAP2 connection. All mailbox state is maintained on the server. There is no registration of clients. The function of a descriptor is handled by a structured representation of the message "envelope". This structure makes it unnecessary for a client to know anything about RFC 822 parsing. There is no synchronization since the client does not remember state between IMAP2 connections. This is not a problem since in general the client never needs the entire state of the mailbox in a single session, therefore there isn't much overhead in fetching the state information that is needed as it is needed.

There are also some functional differences between IMAP2 and DMSP. DMSP has explicit support for bulletin boards which are only handled implicitly in IMAP2. DMSP has functions for sending messages, printing messages, listing mailboxes, and changing passwords, all of which are done outside of IMAP2. DMSP has 16 binary flags of which 8 are defined by the system. IMAP has flag names; there are currently 5 defined system flag names and a facility for some number (30 in the current implementations) of user flag names. IMAP2 has a sophisticated message search facility in the server to identify interesting messages based on dates, addresses, flag status, or textual contents without compelling the client to fetch this data for every message.

It was felt that maintaining state on the client is advantageous only in those cases where the client is only used by a single user, or if there is some means on the client to restrict access to another user's data. It can be a serious disadvantage in an environment in which multiple users routinely use the same client, the same user routinely uses different clients, and where there are no access restrictions on the client. It was also observed that most user mail access is to a relatively small set of "interesting" messages, which were either "new" mail or mail based upon some user-selected criteria. Consequently, IMAP2 was designed to easily identify those "interesting" messages so that the client could fetch the state of those messages and not those that were not "interesting".

The Protocol

The IMAP2 protocol consists of a sequence of client commands and server responses, with server data interspersed between the responses. Unlike most Internet protocols, commands and responses are tagged. That is, a command begins with a unique identifier (typically a short alphanumeric sequence such as a Lisp "gensym" function would generate e.g., A0001, A0002, etc.), called a tag. The response to this command is given the same tag from the server. Additionally, the server may send an arbitrary amount of "unsolicited data", which is identified by the special reserved tag of "*". There is another special reserved tag, "+", discussed below.

The server must be listening for a connection. When a connection is opened the server sends an unsolicited OK response as a greeting message and then waits for commands. When commands are received the server acts on them and responds with responses, often interspersed with data.

The client opens a connection, waits for the greeting, then sends a LOGIN command with user name and password arguments to establish authorization. Following an OK response from the server, the client then sends a SELECT command to access the desired mailbox. The user's default mailbox has a special reserved name of "INBOX" which is independent of the operating system that the server is implemented on. The server will generally send a list of valid flags, number of messages, and number of messages arrived since last access for this mailbox as unsolicited data, followed by an OK response. The client may terminate access to this mailbox and access a different one with another SELECT command.

The client reads mailbox information by means of FETCH commands. The actual data is transmitted via the unsolicited data mechanism (that is, FETCH should be viewed as poking the server to include the desired data along with any other data it wishes to transmit to the client). There are three major categories of data which may be fetched.

The first category is that data which is associated with a message as an entity in the mailbox. There are presently three such items of data: the "internal date", the "RFC 822 size", and the "flags". The internal date is the date and time that the message was placed in the mailbox. The RFC 822 size is subject to deletion in the future; it is the size in bytes of the message, expressed as an RFC 822 text string. Current clients only use it as part of a status display line. The flags are a list of status flags associated with the message (see below). All of the first category data can be fetched by using the macro-fetch word "FAST"; that is, "FAST" expands to "(FLAGS INTERNALDATE RFC822.SIZE)".

The second category is that data which describes the composition and delivery information of a message; that is, information such as the message sender, recipient lists, message-ID, subject, etc. This is the information which is stored in the message header in RFC 822 format message and is traditionally called the "envelope". [Note: this should not be confused with the SMTP (RFC 821) envelope, which is strictly limited to delivery information.] IMAP2 defines a structured and unambiguous representation for the envelope which is particularly nice for Lisp-based parsers. A client can use the envelope for operations such as replying and not worry about RFC 822 at all. Envelopes are discussed in more detail below. The first and second category data can be fetched together by using the macro-fetch word "ALL"; that is, "ALL" expands to "(FLAGS INTERNALDATE RFC822.SIZE ENVELOPE)".

The third category is that data which is intended for direct human viewing. The present RFC 822 based IMAP2 defines three such items: RFC822.HEADER, RFC822.TEXT, and RFC822 (the latter being the two former appended together in a single text string). Fetching "RFC822" is equivalent to typing the RFC 822 representation of the message as stored on the mailbox without any filtering or processing.

Typically, a client will "FETCH ALL" for some or all of the messages in the mailbox for use as a presentation menu, and when the user wishes to read a particular message will "FETCH RFC822.TEXT" to get the message body. A more primitive client could, of course, simply "FETCH RFC822" a la POP2-type functionality.

The client can alter certain data (presently only the flags) by means of a STORE command. As an example, a message is deleted from a mailbox by a STORE command which includes the \DELETED flag as one of the flags being set.

Other client operations include copying a message to another mailbox (COPY command), permanently removing deleted messages (EXPUNGE command), checking for new messages (CHECK command), and searching for messages which match certain criteria (SEARCH command).

The client terminates the session with the LOGOUT command. The server returns a "BYE" followed by an "OK".

A Typical Scenario

```
     Client                          Server
     ------                          ------
                                 {Wait for Connection}
    {Open Connection}        -->
                                <-- * OK IMAP2 Server Ready
                                 {Wait for command}
    A001 LOGIN Fred Secret   -->
                                <-- A001 OK User Fred logged in
                                 {Wait for command}
    A002 SELECT INBOX        -->
                                <-- * FLAGS (Meeting Notice \Answered
                                             \Flagged \Deleted \Seen)
                                <-- * 19 EXISTS
                                <-- * 2 RECENT
                                <-- A0002 OK Select complete
                                 {Wait for command}
    A003 FETCH 1:19 ALL      -->
                                <-- * 1 Fetch (......)
                                         ...
                                <-- * 18 Fetch (......)
                                <-- * 19 Fetch (......)
                                <-- A003 OK Fetch complete
                                 {Wait for command}
    A004 FETCH 8 RFC822.TEXT -->
                                <-- * 8 Fetch (RFC822.TEXT {893}
                                         ...893 characters of text...
                                <-- )
                                <-- A004 OK Fetch complete
                                 {Wait for command}

    A005 STORE 8 +Flags \Deleted -->
                                <-- * 8 Store (Flags (\Deleted
                                               \Seen))
                                <-- A005 OK Store complete
                                 {Wait for command}
    A006 EXPUNGE             -->
                                <-- * 19 EXISTS
                                <-- * 8 EXPUNGE
                                <-- * 18 EXISTS
                                <-- A006 Expunge complete
                                 {Wait for command}
    A007 LOGOUT              -->
                                <-- * BYE IMAP2 server quitting
                                <-- A007 OK Logout complete
    {Close Connection}       --><-- {Close connection}
                                 {Go back to start}
```

Conventions

The following terms are used in a meta-sense in the syntax specification below:

An ASCII-STRING is a sequence of arbitrary ASCII characters.

An ATOM is a sequence of ASCII characters delimited by SP or CRLF.

A CHARACTER is any ASCII character except """", "{", CR, LF, "%", or "\".

A CRLF is an ASCII carriage-return character followed immediately by an ASCII linefeed character.

A NUMBER is a sequence of the ASCII characters which represent decimal numerals ("0" through "9"), delimited by SP, CRLF, ",", or ":".

A SP is the ASCII space character.

A TEXT_LINE is a human-readable sequence of ASCII characters up to but not including a terminating CRLF.

One of the most common fields in the IMAP2 protocol is a STRING, which may be an ATOM, QUOTED-STRING (a sequence of CHARACTERs inside double-quotes), or a LITERAL. A literal consists of an open brace ("{"), a number, a close brace ("}"), a CRLF, and then an ASCII-STRING of n characters, where n is the value of the number inside the brace. In general, a string should be represented as an ATOM or QUOTED-STRING if at all possible. The semantics for QUOTED-STRING or LITERAL are checked before those for ATOM; therefore an ATOM used in a STRING may only contain CHARACTERs. Literals are most often sent from the server to the client; in the rare case of a client to server literal there is a special consideration (see the "+ text" response below).

Another important field is the SEQUENCE, which identifies a set of messages by consecutive numbers from 1 to n where n is the number of messages in the mailbox. A sequence may consist of a single number, a pair of numbers delimited by colon indicating all numbers between those two numbers, or a list of single numbers and/or number pairs. For example, the sequence 2,4:7,9,12:15 is equivalent to 2,4,5,6,7,9,12,13,14,15 and identifies all of those messages.

Definitions of Commands and Responses

Summary of Commands and Responses

Commands	Responses
tag NOOP	tag OK text
tag LOGIN user password	tag NO text
tag LOGOUT	tag BAD text
tag SELECT mailbox	* message_number data
tag CHECK	* FLAGS flag_list
tag EXPUNGE	* SEARCH sequence
tag COPY sequence mailbox	* BYE text
tag FETCH sequence data	* OK text
tag STORE sequence data value	* NO text
tag SEARCH search_program	* BAD text
	+ text

Commands

tag NOOP

The NOOP command returns an OK to the client. By itself, it does nothing, but certain things may happen as side effects. For example, server implementations which implicitly check the mailbox for new mail may do so as a result of this command. The primary use of this command is to for the client to see if the server is still alive (and notify the server that the client is still alive, for those servers which have inactivity autologout timers).

tag LOGIN user password

The LOGIN command identifies the user to the server and carries the password authenticating this user. This information is used by the server to control access to the mailboxes.

EXAMPLE: A001 LOGIN SMITH SESAME logs in as user SMITH with password SESAME.

tag LOGOUT

The LOGOUT command indicates the client is done with the session. The server sends an unsolicited BYE response before the (tagged) OK response, and then closes the connection.

tag SELECT mailbox

The SELECT command selects a particular mailbox. The server must check that the user is permitted read access to this mailbox. Prior to returning an OK to the client, the server must send an unsolicited FLAGS and <n> EXISTS response to the client giving the flags list for this mailbox (simply the system flags if this mailbox doesn't have any special flags) and the number of messages in the mailbox. It is also recommended that the server send a <n> RECENT unsolicited response to the client for the benefit of clients which make use of the number of new messages in a mailbox.

Multiple SELECT commands are permitted in a session, in which case the prior mailbox is deselected first.

The default mailbox for the SELECT command is INBOX, which is a special name reserved to mean "the primary mailbox for this user on this server". The format of other mailbox names is operating system dependent (as of this writing, it reflects the filename path of the mailbox file on the current servers).

EXAMPLE: A002 SELECT INBOX selects the default mailbox.

tag CHECK

The CHECK command forces a check for new messages and a rescan of the mailbox for internal change for those implementations which allow multiple simultaneous read/write access to the same mailbox (e.g., TOPS-20). It is recommend that periodic implicit checks for new mail be done by servers as well. The server should send an unsolicited <n> EXISTS response prior to returning an OK to the client.

tag EXPUNGE

The EXPUNGE command permanently removes all messages with the \DELETED flag set in its flags from the mailbox. Prior to returning an OK to the client, for each message which is removed, an unsolicited <n> EXPUNGE response is sent indicating which message was removed. The message number of each subsequent message in the mailbox is immediately decremented by 1; this means that if the last 5 messages in a 9-message mail file are expunged you will receive 5 "* 5 EXPUNGE" responses. To ensure mailbox integrity and server/client synchronization, it is recommended that the server do an implicit check prior to commencing the expunge and again when the expunge is completed. Furthermore, if the server allows multiple simultaneous access to the same mail file the server must lock the mail file for exclusive access while an expunge is taking place.

EXPUNGE is not allowed if the user does not have write access to this mailbox.

tag COPY sequence mailbox

The COPY command copies the specified message(s) to the specified destination mailbox. If the destination mailbox does not exist, the server should create it. Prior to returning an OK to the client, the server should return an unsolicited <n> COPY response for each message copied. A copy should set the \SEEN flag for all messages which were successfully copied (provided, of course, that the user has write access to this mailbox).

EXAMPLE: A003 COPY 2:4 MEETING copies messages 2, 3, and 4 to mailbox "MEETING".

COPY is not allowed if the user does not have write access to the destination mailbox.

tag FETCH sequence data

The FETCH command retrieves data associated with a message in the mailbox. The data items to be fetched may be either a single atom or an S-expression list. The currently defined data items that can be fetched are:

ALL
: Macro equivalent to: (FLAGS INTERNALDATE RFC822.SIZE ENVELOPE)

ENVELOPE
: The envelope of the message. The envelope is computed by the server by parsing the RFC 822 header into the component parts, defaulting various fields as necessary.

FAST
: Macro equivalent to: (FLAGS INTERNALDATE RFC822.SIZE)

```
     FLAGS           The flags which are set for this message.
                     This may include the following system flags:

                             \RECENT    Message arrived since
                                         last read of this mail
                                         file
                             \SEEN      Message has been read
                             \ANSWERED  Message has been answered
                             \FLAGGED   Message is "flagged" for
                                         urgent/special attention
                             \DELETED   Message is "deleted" for
                                         removal by later EXPUNGE

     INTERNALDATE    The date and time the message was written to
                     the mailbox.

     RFC822          The message in RFC 822 format.

     RFC822.HEADER   The RFC 822 format header of the message.

     RFC822.SIZE     The number of characters in the message as
                     expressed in RFC 822 format.

     RFC822.TEXT     The text body of the message, omitting the
                     RFC 822 header.

     EXAMPLES:

     A003 FETCH 2:4 ALL
        fetches the flags, internal date, RFC 822 size, and envelope
        for messages 2, 3, and 4.

     A004 FETCH 3 RFC822
        fetches the RFC 822 representation for message 3.

     A005 FETCH 4 (FLAGS RFC822.HEADER)
        fetches the flags and RFC 822 format header for message 4.

  tag STORE sequence data value

     The STORE command alters data associated with a message in the
     mailbox.  The currently defined data items that can be stored are:

     FLAGS           Replace the flags for the message with the
                     argument (in flag list format).

     +FLAGS          Add the flags in the argument to the
                     message's flag list.
```

-FLAGS Remove the flags in the argument from the message's flag list.

STORE is not allowed if the user does not have write access to this mailbox.

EXAMPLE: A003 STORE 2:4 +FLAGS (\DELETED)
marks messages 2, 3, and 4 for deletion.

tag SEARCH search_criteria

The SEARCH command searches the mailbox for messages which match the given set of criteria. The unsolicited SEARCH <1#number> response from the server is a list of messages which express the intersection (AND function) of all the messages. The currently defined criteria are:

ALL All messages in the mailbox; the default initial criterion for ANDing.

ANSWERED Messages with the \ANSWERED flag set.

BCC string Messages which contain the specified string in the envelope's BCC field.

BEFORE date Messages whose internal date is earlier than the specified date.

BODY string Messages which contain the specified string in the body of the message.

CC string Messages which contain the specified string in the envelope's CC field.

DELETED Messages with the \DELETED flag set.

FLAGGED Messages with the \FLAGGED flag set.

KEYWORD flag Messages with the specified flag set.

NEW Messages which have the \RECENT flag set but not the \SEEN flag. This is functionally equivalent to "RECENT UNSEEN".

OLD Messages which do not have the \RECENT flag set.

```
ON date          Messages whose internal date is the same as
                 the specified date.

RECENT           Messages which have the \RECENT flag set.

SEEN             Messages which have the \SEEN flag set.

SINCE date       Messages whose internal date is later than
                 the specified date.

SUBJECT string   Messages which contain the specified string
                 in the envelope's SUBJECT field.

TEXT string      Messages which contain the specified string.

TO string        Messages which contain the specified string in
                 the envelope's TO field.

UNANSWERED       Messages which do not have the \ANSWERED flag
                 set.

UNDELETED        Messages which do not have the \DELETED flag
                 set.

UNFLAGGED        Messages which do not have the \FLAGGED flag
                 set.

UNKEYWORD flag   Messages which do not have the specified flag
                 set.

UNSEEN           Messages which do not have the \SEEN flag set.
```

EXAMPLE: A003 SEARCH DELETED FROM "SMITH" SINCE 1-OCT-87
returns the message numbers for all deleted messages from Smith
that were placed in the mail file since October 1, 1987.

Responses

tag OK text

This response identifies successful completion of the command with the indicated tag. The text is a line of human-readable text which may be useful in a protocol telemetry log for debugging purposes.

tag NO text

This response identifies unsuccessful completion of the command with the indicated tag. The text is a line of human-readable text which probably should be displayed to the user in an error report by the client.

tag BAD text

This response indicates faulty protocol received from the client and indicates a bug in the client. The text is a line of human-readable text which should be recorded in any telemetry as part of a bug report to the maintainer of the client.

* number message_data

This response occurs as a result of several different commands. The message_data is one of the following:

EXISTS The specified number of messages exists in the mailbox.

RECENT The specified number of messages have arrived since the last time this mailbox was read.

EXPUNGE The specified message number has been permanently removed from the mailbox, and the next message in the mailbox (if any) becomes that message number.

STORE data
Functionally equivalent to FETCH, only it happens as a result of a STORE command.

FETCH data
This is the principle means by which data about a message is returned to the client. The data is in a Lisp-like S-expression property list form. The current properties are:

ENVELOPE An S-expression format list which describes the

```
                  envelope of a message.  The envelope is computed
                  by the server by parsing the RFC 822 header into
                  the component parts, defaulting various fields
                  as necessary.

                  The fields of the envelope are in the following
                  order: date, subject, from, sender, reply-to, to,
                  cc, bcc, in-reply-to, and message-id.  The date,
                  subject, in-reply-to, and message-id fields are
                  strings.  The from, sender, reply-to, to, cc,
                  and bcc fields are lists of addresses.

                  An address is an S-expression format list which
                  describes an electronic mail address.  The fields
                  of an address are in the following order:
                  personal name, source-route (a.k.a. the
                  at-domain-list in SMTP), mailbox name, and
                  host name.

                  Any field of an envelope or address which is
                  not applicable is presented as the atom NIL.
                  Note that the server must default the reply-to
                  and sender fields from the from field; a client is
                  not expected to know to do this.

         FLAGS    An S-expression format list of flags which are set
                  for this message.  This may include the following
                  system flags:

                  \RECENT       Message arrived since last
                                 read of this mail file
                  \SEEN         Message has been read
                  \ANSWERED     Message has been answered
                  \FLAGGED      Message is "flagged" for
                                 urgent/special attention
                  \DELETED      Message is "deleted" for
                                 removal by later EXPUNGE

         INTERNALDATE A string containing the date and time the
                      message was written to the mailbox.

         RFC822       A string expressing the message in RFC 822
                      format.

         RFC822.HEADER A string expressing the RFC 822 format
                       header of the message

         RFC822.SIZE  A number indicating the number of
```

characters in the message as expressed in RFC 822 format.

RFC822.TEXT A string expressing the text body of the message, omitting the RFC 822 header.

* FLAGS flag_list

This response occurs as a result of a SELECT command. The flag list are the list of flags (at a minimum, the system-defined flags) which are applicable for this mailbox. Flags other than the system flags are a function of the server implementation.

* SEARCH number(s)

This response occurs as a result of a SEARCH command. The number(s) refer those messages which match the search criteria. Each number is delimited by a space, e.g., "SEARCH 2 3 6".

* BYE text

This response indicates that the server is about to close the connection. The text is a line of human-readable text which should be displayed to the user in a status report by the client. This may be sent as part of a normal logout sequence, or as a panic shutdown announcement by the server. It is also used by some servers as an announcement of an inactivity autologout.

* OK text

This response indicates that the server is alive. No special action on the part of the client is called for. This is presently only used by servers at startup as a greeting message indicating that they are ready to accept the first command. The text is a line of human-readable text which may be logged in protocol telemetry.

* NO text

This response indicates some operational error at the server which cannot be traced to any protocol command. The text is a line of human-readable text which should be logged in protocol telemetry for the maintainer of the server and/or the client. No known server currently outputs such a response.

* BAD text

This response indicates some protocol error at the server which cannot be traced to any protocol command. The text is a line of human-readable text which should be logged in protocol telemetry for the maintainer of the server and/or the client. This generally indicates a protocol synchronization problem on the part of the client, and examination of the protocol telemetry is advised to determine the cause of the problem.

+ text

This response indicates that the server is ready to accept the text of a literal from the client. Normally, a command from the client is a single text line. If the server detects an error in the command, it can simply discard the remainder of the line. It cannot do this in the case of commands which contain literals, since a literal can be an arbitrarily long amount of text, and the server may not even be expecting a literal. This mechanism is provided so the client knows not to send a literal until the server definitely expects it, preserving client/server synchronization.

In actual practice, this situation is rarely encountered. In the current protocol, the only client command likely to contain a literal is the LOGIN command. Consider a situation in which a server validates the user before checking the password. If the password contains "funny" characters and hence is sent as a literal, then if the user is invalid an error would occur before the password is parsed.

No such synchronization protection is provided for literals sent from the server to the client, for performance reasons. Any synchronization problems in this direction would be due to a bug in the client or server and not for some operational problem.

Sample IMAP2 session

The following is a transcript of an actual IMAP2 session. Server output is identified by "S:" and client output by "U:". In cases where lines were too long to fit within the boundaries of this document, the line was continued on the next line preceded by a tab.

```
S:      * OK SUMEX-AIM.Stanford.EDU Interim Mail Access Protocol II
                Service 6.1(349) at Thu, 9 Jun 88 14:58:30 PDT
U:      a001 login crispin secret
S:      a002 OK User CRISPIN logged in at Thu, 9 Jun 88 14:58:42 PDT,
                job 76
U:      a002 select inbox
S:      * FLAGS (Bugs SF Party Skating Meeting Flames Request AI
                Question Note \XXXX \YYYY \Answered \Flagged \Deleted
                \Seen)
S:      * 16 EXISTS
S:      * 0 RECENT
S:      a002 OK Select complete
U:      a003 fetch 16 all
S:      * 16 Fetch (Flags (\Seen) InternalDate " 9-Jun-88 12:55:
            RFC822.Size 637 Envelope ("Sat, 4 Jun 88 13:27:11 PDT"
            "INFO-MAC Mail Message" (("Larry Fagan" NIL "FAGAN"
            "SUMEX-AIM.Stanford.EDU")) (("Larry Fagan" NIL "FAGAN"
            "SUMEX-AIM.Stanford.EDU")) (("Larry Fagan" NIL "FAGAN"
            "SUMEX-AIM.Stanford.EDU")) ((NIL NIL "rindflEISCH"
            "SUMEX-AIM.Stanford.EDU")) NIL NIL NIL
            "<12403828905.13.FAGAN@SUMEX-AIM.Stanford.EDU>"))
S:   a003 OK Fetch completed
U:   a004 fetch 16 rfc822
S:   * 16 Fetch (RFC822 {637}
S:   Mail-From: RINDFLEISCH created at  9-Jun-88 12:55:43
S:   Mail-From: FAGAN created at  4-Jun-88 13:27:12
S:   Date: Sat, 4 Jun 88 13:27:11 PDT
S:   From: Larry Fagan  <FAGAN@SUMEX-AIM.Stanford.EDU>
S:   To: rindflEISCH@SUMEX-AIM.Stanford.EDU
S:   Subject: INFO-MAC Mail Message
S:   Message-ID: <12403828905.13.FAGAN@SUMEX-AIM.Stanford.EDU>
S:   ReSent-Date: Thu, 9 Jun 88 12:55:43 PDT
S:   ReSent-From: TC Rindfleisch <Rindfleisch@SUMEX-AIM.Stanford.EDU>
S:   ReSent-To: Yeager@SUMEX-AIM.Stanford.EDU,
                Crispin@SUMEX-AIM.Stanford.EDU
S:   ReSent-Message-ID:
        <12405133897.80.RINDFLEISCH@SUMEX-AIM.Stanford.EDU>
S:
S:   The file is <info-mac>usenetv4-55.arc  ...
S:   Larry
S:   -------
```

```
   S:   )
   S:   pa004 OK Fetch completed
   U:   a005 logout
   S:   * BYE DEC-20 IMAP II server terminating connection
   S:   a005 OK SUMEX-AIM.Stanford.EDU Interim Mail Access Protocol
                   Service logout
```

Implementation Discussion

As of this writing, SUMEX has completed an IMAP2 client for Xerox Lisp machines written in hybrid Interlisp/CommonLisp and is beta-testing a client for TI Explorers written entirely in CommonLisp. SUMEX has also completed a portable IMAP2 client protocol library module written in C. This library, with the addition of a small main program (primarily user interface) and a TCP/IP driver, became a rudimentary remote system mail-reading program under Unix. The first production use of this library will be as a part of a MacII client which is under development.

As of this writing, SUMEX has completed IMAP2 servers for TOPS-20 written in DEC-20 assembly language and 4.2/3 BSD Unix written in C. The TOPS-20 server is fully compatible with MM-20, the standard TOPS-20 mailsystem, and requires no special action or setup on the part of the user. The INBOX under TOPS-20 is the user's MAIL.TXT. The TOPS-20 server also supports multiple simultaneous access to the same mailbox, including simultaneous access between the IMAP2 server and MM-20. The 4.2/3 BSD Unix server requires that the user use mail.txt format which is compatible only with SRI MM-32 or Columbia MM-C. The 4.2/3 BSD Unix server only allows simultaneous read access; write access must be exclusive.

The Xerox Lisp client and DEC-20 server have been in production use for over a year; the Unix server was put into production use a few months ago. IMAP2 has been used to access mailboxes at remote sites from a local workstation via the Internet. For example, from the Stanford local network the author has read his mailbox at a Milnet site.

This specification does not make any formal definition of size restrictions, but the DEC-20 server has the following limitations:

- length of a mailbox: 7,077,888 characters
- maximum number of messages: 18,432 messages
- length of a command line: 10,000 characters
- length of the local host name: 64 characters
- length of a "short" argument: 39 characters
- length of a "long" argument: 491,520 characters
- maximum amount of data output in a single fetch: 655,360 characters

To date, nobody has run up against any of these limitations, many of which are substantially larger than most current user mail reading programs.

There are several advantages to the scheme of tags and unsolicited responses. First, the infamous synchronization problems of SMTP and similar protocols do not happen with tagged commands; a command is not considered satisfied until a response with the same tag is seen. Tagging allows an arbitrary amount of other responses ("unsolicited" data) to be sent by the server with no possibility of the client losing synchronization. Compare this with the problems that FTP or SMTP clients have with continuation, partial completion, and commentary reply codes.

Another advantage is that a non-lockstep client implementation is possible. The client could send a command, and entrust the handling of the server responses to a different process which would signal the client when the tagged response comes in. Under certain circumstances, the client could even have more than one command outstanding.

It was observed that synchronization problems can occur with literals if the literal is not recognized as such. Fortunately, the cases in which this can happen are relatively rare; a mechanism (the special "+" tag response) was introduced to handle those few cases which could happen. The proper way to address this problem in all cases is probably to move towards a record-oriented architecture instead of the text stream model provided by TCP.

Unsolicited data needs some discussion. Unlike most protocols, in which the server merely does the client's bidding, an IMAP2 server has a semi-autonomous role. By means of sending "unsolicited data", the server is in effect sending a command to the client -- to update and/or extend its (incomplete) model of the mailbox with new information from the server. In this viewpoint, a "fetch" command is merely a request to the server to include the desired data in any other "unsolicited" data the server may send, and a server acknowledgement to the "fetch" is a statement that all the requested data has been sent.

In terms of implementation, the client may have a local cache of data from the mailbox. This cache is incomplete, and at startup is empty. A listener processes all unsolicited data, and updates the cache based on this data. If a tagged response arrives, the listener unblocks the process which sent the tagged request.

Perhaps as a result of opening a mailbox, unsolicited data from the server arrives. The first piece of data is the number of messages. This is used to size the cache; note that by sending a new "number of messages" unsolicited data message the cache would be re-sized (this is how newly arrived mail is handled). If the client attempts to access information from the cache, it will encounter empty spots which will trigger "fetch" requests. The request would be sent, some unsolicited data including the answer to the fetch will flow back, and then the "fetch" response will unblock the client.

People familiar with demand-paged virtual memory operating system design will recognize this model as being very similar to page-fault handling on a demand-paged system.

Formal Syntax

The following syntax specification uses the augmented Backus-Naur Form (BNF) notation as specified in RFC 822 with one exception; the delimiter used with the "#" construct is a single space (SP) and not a comma.

```
address          ::= "(" addr_name SP addr_adl SP addr_mailbox SP
                     addr_host ")"

addr_adl         ::= nil / string

addr_host        ::= nil / string

addr_mailbox     ::= nil / string

addr_name        ::= nil / string

check            ::= "CHECK"

copy             ::= "COPY" SP sequence SP mailbox

data             ::= ("FLAGS" SP flag_list / "SEARCH" SP 1#number /
                     "BYE" SP text_line / "OK" SP text_line /
                     "NO" SP text_line / "BAD" SP text_line)

date             ::= string in form "dd-mmm-yy hh:mm:ss-zzz"

envelope         ::= "(" env_date SP env_subject SP env_from SP
                     env_sender SP env_reply-to SP env_to SP
                     env_cc SP env_bcc SP env_in-reply-to SP
                     env_message-id ")"

env_bcc          ::= nil / "(" 1*address ")"

env_cc           ::= nil / "(" 1*address ")"

env_date         ::= string

env_from         ::= nil / "(" 1*address ")"

env_in-reply-to ::= nil / string

env_message-id   ::= nil / string

env_reply-to     ::= nil / "(" 1*address ")"

env_sender       ::= nil / "(" 1*address ")"
```

```
   env_subject     ::= nil / string

   env_to          ::= nil / "(" 1*address ")"

   expunge         ::= "EXPUNGE"

   fetch           ::= "FETCH" SP sequence SP ("ALL" / "FAST" /
                       fetch_att / "(" 1#fetch_att ")")

   fetch_att       ::= "ENVELOPE" / "FLAGS" / "INTERNALDATE" /
                       "RFC822" / "RFC822.HEADER" / "RFC822.SIZE" /
                       "RFC822.TEXT"

   flag_list       ::= ATOM / "(" 1#ATOM ")"

   literal         ::= "{" NUMBER "}" CRLF ASCII-STRING

   login           ::= "LOGIN" SP userid SP password

   logout          ::= "LOGOUT"

   mailbox         ::= "INBOX" / string

   msg_copy        ::= "COPY"

   msg_data        ::= (msg_exists / msg_recent / msg_expunge /
                       msg_fetch / msg_copy)

   msg_exists      ::= "EXISTS"

   msg_expunge     ::= "EXPUNGE"

   msg_fetch       ::= ("FETCH" / "STORE") SP "(" 1#("ENVELOPE" SP
                        envelope / "FLAGS" SP "(" 1#(recent_flag
                        flag_list) ")" / "INTERNALDATE" SP date /
                        "RFC822" SP string / "RFC822.HEADER" SP string /
                        "RFC822.SIZE" SP NUMBER / "RFC822.TEXT" SP
                        string) ")"

   msg_recent      ::= "RECENT"

   msg_num         ::= NUMBER

   nil             ::= "NIL"

   noop            ::= "NOOP"

   password        ::= string
```

```
   recent_flag     ::= "\RECENT"

   ready           ::= "+" SP text_line

   request         ::= tag SP (noop / login / logout / select / check /
                       expunge / copy / fetch / store / search) CRLF

   response        ::= tag SP ("OK" / "NO" / "BAD") SP text_line CRLF

   search          ::= "SEARCH" SP 1#("ALL" / "ANSWERED" /
                       "BCC" SP string / "BEFORE" SP string /
                       "BODY" SP string / "CC" SP string / "DELETED" /
                       "FLAGGED" / "KEYWORD" SP atom / "NEW" / "OLD" /
                       "ON" SP string / "RECENT" / "SEEN" /
                       "SINCE" SP string / "TEXT" SP string /
                       "TO" SP string / "UNANSWERED" / "UNDELETED" /
                       "UNFLAGGED" / "UNKEYWORD" / "UNSEEN")

   select          ::= "SELECT" SP mailbox

   sequence        ::= NUMBER / (NUMBER "," sequence) / (NUMBER ":"
                       sequence)

   store           ::= "STORE" SP sequence SP store_att

   store_att       ::= ("+FLAGS" SP flag_list / "-FLAGS" SP flag_list /
                       "FLAGS" SP flag_list)

   string          ::= atom / """" 1*character """" / literal

   system_flags    ::= "\ANSWERED" SP "\FLAGGED" SP "\DELETED" SP
                       "\SEEN"

   tag             ::= atom

   unsolicited     ::= "*" SP (msg_num SP msg_data / data) CRLF

   userid          ::= string
```

Acknowledgements

Bill Yeager and Rich Acuff both contributed invaluable suggestions in the evolution of IMAP2 from the original IMAP. The SUMEX IMAP2 software was written by Mark Crispin (DEC-20 server, Xerox Lisp client, C client), Frank Gilmurray (Common Lisp client), Christopher Lane (Xerox Lisp client), and Bill Yeager (Unix server). Any mistakes or flaws in this IMAP2 protocol specification are, however, strictly my own.