Network Working Group
Request for Comments: 1176
Obsoletes: RFC 1064

M. Crispin
Washington

August 1990

INTERACTIVE MAIL ACCESS PROTOCOL - VERSION 2

Status of this Memo

This RFC suggests a method for personal computers and workstations to dynamically access mail from a mailbox server ("repository"). It obosoletes RFC 1064. This RFC specifies an Experimental Protocol for the Internet community. Discussion and suggestions for improvement are requested. Please refer to the current edition of the "IAB Official Protocol Standards" for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Introduction

The intent of the Interactive Mail Access Protocol, Version 2 (IMAP2) is to allow a workstation, personal computer, or similar small machine to access electronic mail from a mailbox server. Since the distinction between personal computers and workstations is blurring over time, it is desirable to have a single solution that addresses the need in a general fashion. IMAP2 is the "glue" of a distributed electronic mail system consisting of a family of client and server implementations on a wide variety of platforms, from small single-tasking personal computing engines to complex multi-user timesharing systems.

Although different in many ways from the Post Office Protocols (POP2 and POP3, hereafter referred to collectively as "POP") described in RFC 937 and RFC 1081, IMAP2 may be thought of as a functional superset of these. RFC 937 was used as a model for this RFC. There was a cognizant reason for this; POP deals with a similar problem, albeit with a less comprehensive solution, and it was desirable to offer a basis for comparison.

Like POP, 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).

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 Internet community. The advent of distributed personal computers and workstations has forced a significant rethinking of the mechanisms employed to manage electronic mail. With mainframes, each user tends to receive and process mail at the computer he uses 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 many vendors have implemented facilities to do this. However, this approach has several disadvantages:

(1) Personal computers and many 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 such as receiving mail may not run for long periods of time; either because the user is asking to use all the machine’s resources, or because the user has (perhaps accidentally) manipulated the environment in such a way that it prevents mail reception. In many personal computers, the operating system is single-tasking and this is the only mode of operation. Any of these conditions could lead to repeated failed delivery attempts by outside agents.

(2) The hardware failure of a single machine can keep its user "off the air" for a considerable time, since repair of individual units may be delayed. Given the growing number of personal computers and workstations spread throughout office environments, quick repair of such systems is not assured. On the other hand, a central mainframe is generally repaired soon after failure.

(3) Personal computers and workstations are often not backed up with as much diligence as a central mainframe, if at all.

(4) 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 many 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.

(5) It is difficult to keep a multitude of heterogeneous machines

> 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 a personal computer or workstation may be viewed as an Internet host in the sense that it implements TCP/IP, it should not be viewed as the entity that contains the user's mailbox. Instead, a mail server machine ("server", sometimes called a "repository") should hold the mailbox, and the personal computer or workstation (hereafter referred to as a "client") should access the mailbox via mail transactions.

Because the mail server machine is isolated from direct user manipulation, it should achieve high software reliability easily, and, as a shared resource, it should also achieve high hardware reliability, perhaps through redundancy. The mail server may be accessed from arbitrary locations, allowing users to read mail across campus, town, or country using 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. A mail access protocol retrieves messages, accesss and changes properties of messages, and otherwise manages 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 use 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.

A mail access protocol further differs in that it transmits information only on demand. A well-designed mail access protocol requires considerably less network traffic than Unix mail via NFS, particularly when the mail file is large. The result is that a mail access protocol can scale well to situations of large mailboxes or networks with high latency or low speed.

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 is desirable to allow for a

growth path beyond the hoary text-only RFC 822 protocol, specifically in the area of attachments and multi-media mail, to ease the eventual transition to ISO solutions.

Unlike POP, IMAP2 has extensive features for remote searching and parsing of messages on the server. A free text search (optionally with other searching) can be made in 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. The effect of this is twofold: it further improves the ability to scale well in instances where network traffic must be reduced, and it reduces the complexity of the client program.

Additionally, IMAP2 offers several facilities for managing individual message state and the mailbox as a whole beyond the simple "delete message" functionality of POP. Another benefit of IMAP2 is the use of tagged responses to reduce the possibility of synchronization errors and the concept of state on the client (a "local cache") that the server may update without explicit request by the client. These concepts and how they are used are explained under "Implementation Discussion" below.

In spite of this functional richness, IMAP2 is a small protocol. Although servers should implement the full set of IMAP2 functions, a simple client can be written that uses IMAP2 in much the way as a POP client.

A related protocol to POP and IMAP2 is the DMSP protocol of PCMAIL (RFC 1056). IMAP2 differs from DMSP more fundamentally, reflecting a differing architecture from PCMAIL. PCMAIL is either an online ("interactive mode"), or offline ("batch mode") system with long-term shared state. Some POP based systems are also offline; in such systems, since there is no long-term shared state POP is little more than a download mechanism of the "mail file" to the client. IMAP2-based software 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 that 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 IMAP2-based software, the client/server relationship lasts only for the duration of the TCP 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" as noted above. There is no client/server 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 functions for sending messages, printing messages, listing mailboxes, and changing passwords; these are done outside IMAP2. DMSP has 16 binary flags of which 8 are defined by the system. IMAP2 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 small set of "interesting" messages, which were either new mail or mail based on 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.

The client opens a connection and waits for the greeting. The client must not send any commands until it has received the greeting from the server.

Once the greeting has been received, the client may begin sending commands and is not under any obligation to wait for a server response to this command before sending another command, within the constraints of TCP flow control. When commands are received the server acts on them and responds with command responses, often interspersed with data. The effect of a command can not be considered complete until a command response with a tag matching the command is received from the server.

Although all known IMAP2 servers at the time of this writing process commands to completion before processing the next command, it is not required that a server do so. However, many commands can affect the results of other commands, creating processing-order dependencies (or, for SEARCH and FIND, ambiguities about which data is associated with which command). All implementations that operate in a non-lockstep fashion must recognize such dependencies and defer or synchronize execution as necessary. In general, such multi-processing is limited to consecutive FETCH commands.

Generally, the first command from the client is a LOGIN command with user name and password arguments to establish identity and access authorization, unless this has already been accomplished through other means, e.g. Kerberos. Until identity and access authorization have been established, no operations other than LOGIN or LOGOUT are permitted.

Once identity and authorization have been established, the client must send a SELECT command to access the desired mailbox; no mailbox is selected by default. SELECT’s argument is implementation-dependent; however the word "INBOX" must be implemented to mean the primary or default mailbox for this user, independent of any other server semantics. On a successful SELECT, the server will 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 with FETCH commands. The actual

data is transmitted via the unsolicited data mechanism (that is, FETCH should be viewed as instructing 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 that may be fetched.

The first category is data that is associated with a message as an entity in the mailbox. There are now 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 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 that 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 that 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 that is particularly suited 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 two categories of 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 that 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). RFC822.HEADER is the "raw", unprocessed RFC 822 format header of the message. Fetching "RFC822" is equivalent to fetching the RFC 822 representation of the message as stored on the mailbox without any filtering or processing.

An intelligent 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 POP-type functionality.

The client can alter certain data (currently only the flags) by a STORE command. As an example, a message is deleted from a mailbox by a STORE command that includes the \DELETED flag as a flag 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 that 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 that 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.

A common field 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 (equivalent to all numbers between those two numbers), or a list of single numbers 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 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	* number message_data
tag BBOARD bulletin_board	* FLAGS flag_list
tag FIND MAILBOXES pattern	* SEARCH sequence
tag FIND BBOARDS pattern	* BBOARD string
tag CHECK	* MAILBOX string
tag EXPUNGE	* BYE text
tag COPY sequence mailbox	* OK text
tag FETCH sequence data	* NO text
tag STORE sequence data value	* BAD text
tag SEARCH search_program	+ 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 that 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 that 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 informs the server that the client is done with the session. The server should send an unsolicited BYE response before the (tagged) OK response, and then close the network 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. Before returning an OK to the client, the server must send the following unsolicited data to the client:

```
FLAGS        mailbox's defined flags
<n> EXISTS   the number of messages in the mailbox
<n> RECENT   the number of new messages in the mailbox
```

in order to define the initial state of the mailbox at the client.

Multiple SELECT commands are permitted in a session, in which case the previous mailbox is automatically deselected when a new SELECT is made.

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).

It is customary, although not required, for the text of an OK response to the SELECT command to begin with either "[READ-ONLY]" or "[READ-WRITE]" to show the mailbox's access status.

EXAMPLE: A002 SELECT INBOX
selects the default mailbox.

tag BBOARD bulletin_board

The BBOARD command is equivalent to SELECT, and returns the same output. However, it differs from SELECT in that its argument is a shared mailbox (bulletin board) name instead of an ordinary mailbox. The format of a bulletin name is implementation specific, although it is strongly encouraged to use something that resembles a name in a generic sense and not a file or mailbox name

on the particular system. There is no requirement that a bulletin board name be a mailbox name or a file name (in particular, Unix netnews has a completely different namespace from mailbox or file names).

Support for BBOARD is optional.

tag FIND MAILBOXES pattern

The FIND MAILBOXES command accepts as an argument a pattern (including wildcards) that specifies some set of mailbox names that are usable by the SELECT command. The format of mailboxes is implementation dependent. The special mailbox name INBOX is not included in the output.

Two wildcard characters are defined; "*" specifies any number (including zero) characters may match at this position and "%" specifies a single character may match at this position. For example, FOO*BAR will match FOOBAR, FOOD.ON.THE.BAR and FOO.BAR, whereas FOO%BAR will match only FOO.BAR. "*" will match all mailboxes.

The FIND MAILBOXES command will return some set of unsolicited MAILBOX replies that have as their value a single mailbox name.

```
EXAMPLE:  A002 FIND MAILBOXES *
          * MAILBOX FOOBAR
          * MAILBOX GENERAL
          A002 FIND completed
```

Although the use of explicit file or path names for mailboxes is discouraged by this standard, it may be unavoidable. It is important that the value returned in the MAILBOX unsolicited reply be usable in the SELECT command without remembering any path specification that may have been used in the FIND MAILBOXES pattern.

Support for FIND MAILBOXES is optional. If a client’s attempt returns BAD as a response then the client can make no assumptions about what mailboxes exist on the server other than INBOX.

tag FIND BBOARDS pattern

The FIND BBOARDS command accepts as an argument a pattern that specifies some set of bulletin board names that are usable by the BBOARD command. Wildcards are permitted as in FIND MAILBOXES.

The FIND BBOARDS command will return some set of unsolicited

BBOARD replies that have as their value a single bulletin board name.

```
EXAMPLE:  A002 FIND BBOARDS *
          * BBOARD FOOBAR
          * BBOARD GENERAL
          A002 FIND completed
```

Support for FIND BBOARDS is optional. If a client's attempt returns BAD as a response then the client can make no assumptions about what bulletin boards exist on the server, or that they exist at all.

tag CHECK

The CHECK command forces a check for new messages and a rescan of the mailbox for internal change for those implementations that allow multiple simultaneous read/write access to the same mailbox. It is recommend that periodic implicit checks for new mail be done by servers as well. The server should send unsolicited EXISTS and RECENT responses with the current status before 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. Before returning an OK to the client, for each message that is removed, an unsolicited EXPUNGE response is sent. The message number for each successive 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 unsolicited EXPUNGE responses for message 5. To ensure mailbox integrity and server/client synchronization, it is recommended that the server do an implicit check before 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. Before 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 that 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 that are set for this message. This may include the following system flags:

\RECENT	Message arrived since the previous time this mailbox was read
\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. The \SEEN flag is implicitly set; if this causes the flags to change they should be included as part of the fetch results. This is the concatenation of RFC822.HEADER and RFC822.TEXT.

RFC822.HEADER The "raw" RFC 822 format header of the message as stored on the server.

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. The \SEEN flag is implicitly set as with RFC822 above.

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.

Note: An attempt to FETCH already-transmitted data may have no result. See the Implementation Discussion below.

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 that match the given set of criteria. The unsolicited SEARCH <1#number> response from the server is a list of messages that express the intersection (AND function) of all the messages which match that criteria. For 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.

In all search criteria which use strings, a message matches the criteria if the string is a case-independent substring of that field. 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.
FROM string	Messages which contain the specified string in the envelope’s FROM field.
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.

Responses

tag OK text

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

tag NO text

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

tag BAD text

This response identifies faulty protocol received from the client; The text is a line of human-readable text that 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 previous 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
Obsolete and functionally equivalent to FETCH.

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 that 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 that 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 that 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 that are set for this message. This may include the following system flags:

\RECENT	Message arrived since the previous time this mailbox was read
\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) that 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 to those messages that match the search criteria. Each number is delimited by a space, e.g., "SEARCH 2 3 6".

* BBOARD string

This response occurs as a result of a FIND BBOARDS command. The string is a bulletin board name that matches the pattern in the command.

* MAILBOX string

This response occurs as a result of a FIND MAILBOXES command. The string is a mailbox name that matches the pattern in the command.

* BYE text

This response identifies that the server is about to close the connection. The text is a line of human-readable text that 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 identifies normal operation on the server. No special action by the client is called for, however, the text should be displayed to the user in some fashion. This is currently only used by servers at startup as a greeting message to show they are ready to accept the first command.

* NO text

  This response identifies a warning from the server that does not affect the overall results of any particular request. The text is a line of human-readable text that should be presented to the user as a warning of improper operation.

* BAD text

  This response identifies a serious error at the server; it may also indicate faulty protocol from the client in which a tag could not be parsed. The text is a line of human-readable text that should be presented to the user as a serious or possibly fatal error. It should also be recorded in any telemetry as part of a bug report to the maintainer of the client and server.

+ text

  This response identifies 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 for commands that 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 expects it, preserving client/server synchronization.

  In practice, this condition is rarely encountered. In the current protocol, the only client command likely to contain a literal is the LOGIN command. Consider a server that 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 caused by a bug in the client or server.

Sample IMAP2 session

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

```
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 12:55:44 PDT"
      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:   a004 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

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 that would signal the client when the tagged response comes in. Under certain circumstances, the client may 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 rare; a mechanism (the special "+" tag response) was introduced to handle those few cases. The proper way to address this problem is probably to move towards a record-oriented architecture instead of the text stream model provided by TCP.

An IMAP2 client must maintain a local cache of data from the mailbox. This cache is an incomplete model of the mailbox, 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 that sent the tagged request.

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 sending "unsolicited data", the server is in effect sending a command to the client -- to update or extend the client’s cache with new information from the server. In other words, a "fetch" command is merely a request to the server to ensure that the client’s cache has the most up-to-date version of the requested information. A server acknowledgement to the "fetch" is a statement that all the requested data has been sent.

Although no current server does this, a server is not obliged by the protocol to send data that it has already sent and is unchanged. An exception to this is the actual message text fetching operations (RFC822, RFC822.HEADER, and RFC822.TEXT), owing to the possibly excessive resource consumption of maintaining this data in a cache. It can not be assumed that a FETCH will transmit any data; only that an OK to the FETCH means that the client’s cache has the most up-to-

date information.

When a mailbox is selected, the initial unsolicited data from the server arrives. The first piece of data is the number of messages. By sending a new EXISTS unsolicited data message the server causes the client to resize its cache (this is how newly arrived mail is handled). If the client attempts to access information from the cache, it will encounter empty spots that 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 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

bboard          ::= "BBOARD" SP 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"

   find            ::= "FIND" SP find_option SP string

   find_option     ::= "MAILBOXES" / "BBOARDS"

   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 / find /
                       bboard) 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
```

Implementation Status

This information is current as of this writing.

The University of Washington has developed an electronic mail client library called the "C-Client". It provides complete IMAP2, SMTP, and local mailbox (both /usr/spool/mail and mail.txt formats) services in a well-defined way to a user interface main program. Using the C-Client, the University of Washington has created an operational client for BSD Unix and two operational clients (one basic, one advanced) for the NeXT.

Stanford University/SUMEX has developed operational IMAP2 clients for Xerox Lisp machines, Texas Instruments Explorers, and the Apple Macintosh. The core of the Macintosh client is an early version of the C-Client. SUMEX has also developed IMAP2 servers for TOPS-20 and BSD Unix.

All of the above software is in production use, with enthusiastic local user communities. Active development continues on the Macintosh and C-Client based clients and the BSD Unix server. This software is freely available from the University of Washington and SUMEX.

IMAP2 software exists for other platforms; for example Nippon Telephone and Telegraph (NTT) has developed an operational IMAP2 client for the NTT ELIS. Several organizations are working on a PC client.

IMAP2 can be used to access mailboxes at very remote sites, where echo delays and frequent outages make TELNET and running a local mail reader intolerable. For example, from a desktop workstation on the University of Washington local network the author routinely uses IMAP2 to read and manage mailboxes on various University of Washington local servers, at two systems at Stanford University, at a Milnet site, and at a site in Tokyo, Japan.

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.

Acknowledgements

Bill Yeager and Rich Acuff both contributed invaluable suggestions in the evolution of IMAP2 from the original IMAP. James Rice pointed out several ambiguities in the previous IMAP2 specification and otherwise would not allow me to leave bad enough along. Laurence Lundblade reviewed a draft of this version and made several helpful suggestions.

Many dedicated individuals have worked on IMAP2 software, including: Mark Crispin, Frank Gilmurray, Christopher Lane, Hiroshi Okuno, Christopher Schmidt, and Bill Yeager.

Any mistakes, flaws, or sins of omission in this IMAP2 protocol specification are, however, strictly my own; and the mention of any name above does not imply an endorsement.

Security Considerations

Security issues are not discussed in this memo.

Author’s Address

Mark R. Crispin
Panda Programming
6158 Lariat Loop NE
Bainbridge Island, WA 98110-2020

Phone: (206) 842-2385

EMail: mrc@Tomobiki-Cho.CAC.Washington.EDU