Network Working Group
Request for Comments: 1203
Obsoletes: RFC 1064

J. Rice
Stanford

February 1991

INTERACTIVE MAIL ACCESS PROTOCOL - VERSION 3

Status of this Memo

This RFC suggests a method for workstations to access mail dynamically from a mailbox server ("repository"). This RFC specifies a standard for the SUMEX-AIM community and 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.

Scope

The following document is a modified version of RFC 1064, the definition of the IMAP2 protocol. This RFC has been written specifically as a counter proposal to RFC 1176, which itself proposes modifications to IMAP2. Sadly, RFC 1176 was made without internal consultation with the IMAP community, so we are in a position of feeling we have to present a counter proposal to what, if we do not act, will become a de facto standard. The reasons for this counter proposal are numerous but fall mostly into the following categories:

- IMAP2 is insufficiently powerful for a number of server/client interactions which we believe to be important. RFC 1176 negligibly enhances the functionality of IMAP2.

- IMAP2 makes what we believe to be an erroneous definition for unsolicited vs. solicited data. IMAP3 as specified herein attempts to correct this. RFC 1176 makes no effort to remedy these problems.

- RFC 1176 has explicitly modified the intent of RFC 1064 by allowing the server to make assumptions about the client’s caching architecture. We believe this to be a grave error and do not support it in this proposal.

- RFC 1176 specifies a number of "optional" features in the protocol without specifying a suitable metaprotocol by which servers and clients can adequately negotiate over the set of implemented features. This proposal specifies a mechanism by which servers and clients can come to an unambiguous understanding about which features are usable by each party.

- RFC 1176 pays only lip-service to being network protocol independent and, in fact assumes the use of TCP/IP. Neither RFC 1064 nor this proposal make any such assumption.

Although there are numerous other detailed objections to RFC 1176, we believe that the above will serve to show that we believe strongly in the importance of mailbox abstraction level mail protocols and, after a couple of years of use of IMAP2 under RFC 1064 we believe that we have a good enough understanding of the issues involved to be able to take the next step.

It is important to take this next step because of the rapid pace of both mail system and user interface development. We believe that, for IMAP not to die in its infancy, IMAP must be ready to respond to emerging ISO and RFC standards in mail, such as for multi-media mail. We believe that RFC 1176 not only provides a very small increment in functionality over RFC 1064 but also adds a number of bugs, which would be detrimental to the IMAP cause. Thus we propose the following definition for IMAP3.

Compatibility notes:

In revising the IMAP2 protocol it has been our intent, wherever possible to make upwards compatible changes to produce IMAP3. There were, however, some places that had to be changed incompatibly in order to compensate for either ambiguities in the IMAP2 protocol as defined by RFC 1064 or behavior that proved undesirable in the light of experience.

It is our goal, however, that existing IMAP2 clients should still be supported and that, at least for the foreseeable future, all IMAP3 servers will support IMAP2 behavior as their default mode.

The following are the major differences between this proposal, RFC 1176 and RFC 1064:

- In this proposal we specify a difference between "solicited" and "unsolicited" data sent from the server. It is generally the case that data sent by the server can be sent either in response to an explicit request by the client or by the server of its own volition. Any data that the server is required to sent to the client as the result of a request is said to be solicited and carries the same tag as the request that provoked it. Any data sent by the server to the client that is not required by the protocol is said to be unsolicited and carries the special "*" tag. RFC 1176 preserves the original RFC 1064 terminology that calls all such data sent by the server "unsolicited" even when

it is, in fact, solicited.

- This proposal introduces the experimental concept of distinguishing between Generic, Canonical and Concrete keys, allowing the mailbox to be viewed as a relational database indexed by these keys. This should allow the IMAP protocol to evolve away from its current reliance on RFC 822. RFC 1176 does not have such a unifying model.

- The SEARCH command has been changed so as to allow multiple simultaneous searches to be made and to allow unsolicited search messages to be sent by the server. Such a change is essential to allow more sophisticated servers that can process commands asynchronously, possibly substantially delaying searches over slow backing storage media, for example. It is also important to allow servers to be able to send unsolicited search messages that might inform the client of interesting patterns of messages, such as new and unseen mail.

- This proposal introduces a specific protocol for the negotiation of protocol versions and server features. This is important because it allows client/server pairs to come to an agreement on what behavior is really available to it. RFC 1176 introduces a number of "optional" commands, which are in some way analogous to "feature-introduced" commands in this proposal. The principle distinction between these is that in RFC 1176 there is no way for a client to discover the set of optional commands, nor is there a way for it to determine whether a specific command really is supported, since RFC 1176 requires the use of the "BAD" response if a feature is not supported. There is, therefore, no way for the client to determine why the attempted command did not work. This also means that, for example, a client cannot disable certain user commands or make them invisible on menus if they are not supported, since there is no way for the client to discover whether the commands are indeed supported without trying to execute such a command.

- This proposal introduces a mechanism for clients to create and delete user flags (keywords). This is nor supported in either RFC 1176 or RFC 1064, requiring the user to add keys manually on the server, generally by editing some form of "init" file.

- RFC 1064 has no mechanism for determining whether a mailbox is readonly or not. RFC 1176 introduces a non-enforced convention of encoding data about the readonly status of a mailbox in the SELECT message’s OK respose comment field. This is not regular with respect to the rest of the protocol, in which the comment field is used for no purpose other than documentation. This

proposal introduces specific protocol additions for the dynamic determination and modification of the readonly/readwrite status of mailboxes.

Introduction

The intent of the Interactive Mail Access Protocol, Version 3 (IMAP3) is to allow a (possibly unreliable) workstation or similar machine to access electronic mail from a reliable mailbox server in an efficient manner.

Although different in many ways from POP2 (RFC 937), IMAP3 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, IMAP3 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 IMAP server listens on port 220. When CHAOS is used the IMAP server listens for the logical contact name "IMAP3".

Communication in IMAP is defined to be using the ASCII character interpretation of data. Communication using other conventions may be possible by the selection of features on some servers.

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, IMAP3 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, IMAP3 offers several facilities for managing a mailbox beyond the simple "delete message" functionality of POP2.

In spite of this, IMAP3 is a relatively simple protocol. Although servers should implement the full set of IMAP3 functions, a simple client can be written which uses IMAP3 in much the way as a POP2 client.

IMAP3 differs from the DMSP protocol of PCMAIL (RFC 1056) in a more fundamental manner, reflecting the differing architectures of IMAP and PCMAIL. PCMAIL is either an online ("interactive mode"), or offline ("batch mode") system. IMAP 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 IMAP, the client/server relationship lasts only for the duration of the IMAP3 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 IMAP3 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 IMAP3 and DMSP. DMSP has functions for sending messages, printing messages, and changing passwords, all of which are done outside of IMAP3. 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 (29 in the current implementations) of user flag names. IMAP3 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, IMAP3 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".

One crucial philosophical difference between IMAP and other common

mail protocols is that IMAP is a mailbox access protocol, not a protocol for manipulating mail files. In the IMAP model, unlike other mail system models in which mail is stored in a linear mail file, no specification is made for the implementation architecture for mail storage. Servers may choose to implement mailboxes as files but this is a detail of which the client can be totally unaware.

What is more, in the IMAP model, mailboxes are viewed as mappings from keys into values. There are broadly three types of keys, generic, canonical and concrete. Generic keys are generic, mail protocol independent keys defined by IMAP which are meaningful across multiple mail encoding formats. An example of such a generic key might be "TO", which would be associated with the "To:" field of an RFC 822 format message.

Canonical keys represent the way in which the server can associate values that are generally "about" a certain key concept, possibly integrating several mail format specific fields, without having to worry the client with the particular details of any particular message format. Thus, the canonical TO key (called $TO) could denote anything that could reasonably be construed as being directed towards someone. Hence, in an RFC 822 message the server could find the union of the "To:", "Resent-To", "Apparently-To:" and "CC:" fields to be the appropriate value associated with the canonical $TO key.

Concrete keys allow the client to gain access to certain mail format specific concepts, that are not pre-specified by the IMAP protocol, in a well defined manner. For example, If the client asks for the value associated with the "APPARENTLY-TO" key then, if the message were to be in RFC 822 format, the server would look for a header field called "Apparently-To:". If no such field is found or the field is not implemented or meaningful for the particular message format then the server will respond with the null value, called NIL, indicating the non-existence of the field.

Thus, IMAP servers are at liberty to implement mailboxes as a relational databases if it seems convenient. Indeed, we anticipate that future mail systems will tend to use database technology for the storage and indexing of mailboxes as a result of the pressure caused by the increasing size of mailboxes.

Although for historical reasons IMAP is currently somewhat closely associated with RFC 822, we anticipate that future developments in IMAP will remove these mail format specific components and will move towards the generic model mentioned above. This will allow IMAP more easily to incorporate such things as multi-media mail.

The Protocol

The IMAP3 protocol consists of a sequence of client commands and server responses to those commands, with extra information from the server data being sent asynchronously to and independent to the responses to client commands. 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.

We distinguish between data sent by the server as the result of a client request, which we term "SOLICITED" and data sent by the server not as the result of a client request, which we term "UNSOLICITED". The server may send unsolicited data at any time that would not fragment another piece of data on the same stream rendering it unintelligible. The server is contractually required, however, to return all data that is solicited by the client before the return of the completion signal for that command, i.e., all solicited data must be returned within the temporal extent of the request/completion acknowledgement wrapper. This does not, however, preclude the simultaneous processing of multiple requests by the client, it simply requires that the client be confident that it has all the requested data when a request finishes. This allows the implementation of both synchronous and asynchronous clients.

Solicited data is identified by the tag of the initial request by the client. Unsolicited data is identified by the special reserved tag of "*". There is another special reserved tag, "+", discussed below.

Note: the tagging of SOLICITED data is only permitted for a selected server version other than 2.0.

No assumptions concerning serial or monolithic processing by the server can be made by a correct client. The server is at liberty to process multiple requests by the same client in any order. This allows servers to process costly searches over mailboxes on slow backing storage media in the background, while still preserving interactive performance. Clients can, however, assume the serialization of the request/data/completion behavior mentioned above.

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 solicited data, followed by an OK response. The client may terminate access to this mailbox and access a different one with another SELECT command.

Because the SELECT command affects the state of the server in a fundamental way, the server is required to process all outstanding commands for any given mailbox before sending the OK tag for the SELECT command. Thus, the client will always know that all responses before an OK SELECT response will refer to the old mailbox and all responses following it will apply to the new mailbox.

Because, in the real world, local needs or experimental work will dictate that servers will support both supersets of the defined behavior and incompatible changes, servers will support a SELECT.VERSION command and a SELECT.FEATURES command, the purpose of which is to allow clients to select the overall behavior and specific features that they want from a server. The default behavior of any server is to process commands and to have interaction syntax the same as is specified by IMAP2 in RFC 1064. A server may not behave in any other manner unless the SELECT.VERSION or SELECT.FEATURES commands are used to select different behavior.

Over time, when groups of generally useful changes to the current, default behavior of the server are found, these will be collected together and incorporated in such a way that all of the features can be selected simply by selecting a particular major version number of the protocol. It should be noted that the version numbers (both major and minor) selected by the SELECT.VERSION command denote versions of the IMAP protocol, not versions of the server per se. Thus, although in general changes to the protocol specification will be made in such a way that they are upwards compatible, this cannot be guaranteed. No client should rely on tests of the form "if major_version > 2 then..." being valid for all protocol versions, since incompatible changes might be made in the future.

The client reads mailbox information by means of FETCH commands. The actual data is transmitted via the solicited 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.] IMAP3 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 IMAP3 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 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 IMAP3 Server Ready
                                  {Wait for command}
  A001 SUPPORTED.VERSIONS     -->
                                <-- * SUPPORTED.VERSIONS ((2 0 )
                                       (3 0 EIGHT.BIT.TRANSPARENT
                                            AUTO.SET.SEEN
                                            TAGGED.SOLICITED))
                                  A001 OK Supported Versions returned.
                                  {Wait for command}
  A002 SELECT.VERSION (3 0) -->
                                <-- A002 OK Version 3.0 Selected.
                                  {Wait for command}
  A002 SELECT.FEATURES TAGGED.SOLICITED -->
                                <-- A002 OK Features selected.
                                  {Wait for command}
  A003 LOGIN Fred Secret    -->
                                <-- A003 OK User Fred logged in
                                  {Wait for command}
  A004 SELECT INBOX         -->
                                <-- A004 FLAGS (Meeting Notice \Answered
                                            \Flagged \Deleted \Seen)
                                <-- A004 19 EXISTS
                                <-- A004 2 RECENT
                                <-- A004 OK Select complete
                                  {Wait for command}
  A005 FETCH 1:19 ALL       -->
                                <-- A005 1 Fetch (......)
                                       ...
                                <-- A005 18 Fetch (......)
                                <-- A005 19 Fetch (......)
                                <-- A005 OK Fetch complete
                                  {Wait for command}
  A006 FETCH 8 RFC822.TEXT -->
                                <-- A006 8 Fetch (RFC822.TEXT {893}
                                     ...893 characters of text...
                                <-- )
                                <-- A006 OK Fetch complete
                                  {Wait for command}
```

```
    A007 STORE 8 +Flags \Deleted -->
                                <-- A007 8 Store (Flags (\Deleted
                                                         \Seen))
                                <-- A007 OK Store complete
                                    {Wait for command}
    A008 EXPUNGE              -->
                                <-- A008 19 EXISTS
                                <-- A008 8 EXPUNGE
                                <-- A008 18 EXISTS
                                <-- A008 Expunge complete
                                    {Wait for command}
    A009 LOGOUT               -->
                                <-- A009 BYE IMAP3 server quitting
                                <-- A009 OK Logout complete
    {Close Connection}        --><-- {Close connection}
                                    {Go back to start}
```

A more complex scenario produced by a pipelining multiprocess client.

```
        Client                          Server
        ------                          ------
                                    {Wait for Connection}
    {Open session as above}
                                <-- A004 19 EXISTS
                                <-- A004 2 RECENT
                                <-- A004 OK Select complete
                                    {Wait for command}
    A005 SEARCH RECENT        -->
                                <-- A005 SEARCH (18 19) (RECENT)
                                <---A005 OK Search complete
    A006 FETCH 18:19 ALL RFC822.TEXT
    A007 STORE 18:19 +FLAGS (\SEEN)
    A008 FETCH 1:17 ALL       -->
                                <-- A006 18 Fetch (... RFC822.TEXT ...)
    A009 STORE 18 +FLAGS (\DELETED)
                                <-- A006 19 Fetch (... RFC822.TEXT ...)
                                <-- A006 OK Fetch complete
                                <-- A007 18 STORE (Flags (\Seen))
    A010 STORE 19 +FLAGS (\DELETED)
                                <-- A007 19 STORE (Flags (\Seen))
                                <-- A007 OK Store complete
                                <-- A008 1 Fetch (......)
                                       ...
                                <-- A008 16 Fetch (......)
                                <-- A008 17 Fetch (......)
                                <-- A008 OK Fetch complete
                                <-- A009 18 STORE (Flags (\Seen
                                                          \Deleted))
```

```
                                <-- A009 OK Store complete
                                <-- A010 19 STORE (Flags (\Seen
                                                          \Deleted))
                                <-- A010 OK Store complete
                                    {Wait for command}
                                <-- * EXISTS 23
                                <-- * RECENT 4
                                <-- * SEARCH (20 21 22 23) (RECENT)
A011 FETCH 20:23 ALL RFC822.TEXT
```

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 IMAP3 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:

```
tag NOOP
tag LOGIN user password
tag LOGOUT
tag SELECT mailbox
tag CHECK
tag EXPUNGE
tag COPY sequence mailbox
tag FETCH sequence data
tag STORE sequence data value
tag SEARCH criteria
tag BBOARD bboard
tag FIND (BBOARDS / MAILBOXES) pattern
tag READONLY
tag READWRITE
tag SELECT.VERSION (major_version minor_version)
tag SELECT.FEATURES features
tag SUPPORTED.VERSIONS
tag FLAGS
tag SET.FLAGS
```

Responses (can be either solicited or unsolicited):

```
*/tag FLAGS flag_list
*/tag SEARCH (numbers) (criteria)
*/tag EXISTS
*/tag RECENT
*/tag EXPUNGE
*/tag STORE data
*/tag FETCH data
*/tag BBOARD bboard_name
*/tag MAILBOX non_inbox_mailbox_name
*/tag SUPPORTED.VERSIONS version_data
*/tag READONLY
*/tag READWRITE
*/tag OK text
*/tag NO text
*/tag BAD text
```

```
       */tag BYE text

Responses (can only be solicited):
       tag COPY message_number

Responses (can only be unsolicited):
       + 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 a solicited 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 solicited 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. It is further recommended that servers should send an unsolicited READONLY message if the mailbox that has been selected is not

writable by the user.

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 path of the mailbox on the current servers), though it could reflect any server-specific naming convention for the namespace of mailboxes. Such a namespace need not and should not be viewed as being equivalent or linked to the server machine's file system.

```
EXAMPLES: A002 SELECT INBOX  ;; selects the default mailbox.
          A002 197 EXISTS    ;; server says 197 messages in INBOX
          A002 5 RECENT      ;; server says 5 are recent.
          A002 OK Select complete.
or
          A003 SELECT /usr/fred/my-mail.txt
           ;; select a different user specified 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 must send a solicited <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, a solicited <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 mailbox are expunged you will receive 5 "5 EXPUNGE" responses for message 5. 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 mailbox the server must guarantee both the integrity of the mailbox and

the views of it held by the clients.

EXPUNGE is not allowed if the user does not have write access to this mailbox. If a user does not have write access to the mailbox then the server is required to signal this fact by replying with a NO response with a suitable text string that can be presented to the user explaining that the mailbox is read-only. It is further recommended that servers send an unsolicited READONLY message to clients that attempt an expunge operation on a read only 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 must return a solicited <n> COPY response for each message copied.

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. If a user does not have write access to the destination mailbox then the server is required to signal this fact by replying with a NO response with a suitable text string that can be presented to the user explaining that the mailbox is read-only. It is further recommended that servers send an unsolicited READONLY message to clients that attempt to copy to a read only mailbox. IMAP3 does not specify "where" the message will be put in the mailbox to which it has been copied.

tag FETCH sequence fetch_att

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 attributes that can be fetched are any of those mentioned specifically below along with any generic, canonical or concrete key. The set of predefined generic keys is: {BCC, BODY, CC, FROM, HEADER, SIZE, SUBJECT, TEXT, TO}. The set of predefined canonical keys is {$CC, $FROM, $SUBJECT, $TO}. The value returned by the server for a non-existent or non-meaningful key is defined to be the null value, NIL.

ALL	Equivalent to: (FLAGS INTERNALDATE RFC822.SIZE ENVELOPE)
ENVELOPE	The envelope of the message. The envelope is computed by the server by parsing the header,

i.e., the RFC 822 header for an RFC822 format message, 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 mailbox
\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.

```
A006 FETCH 42 $SUBJECT
A006 FETCH $SUBJECT "Some subject text..."
A006 OK FETCH completed ok.
```
fetches the canonical subject field.

```
   A007 FETCH 42 APPARENTLY-TO
   A007 FETCH APPARENTLY-TO NIL
   A007 OK FETCH found no value.
      fetches the concrete apparently-to field.
```

tag STORE sequence data value

The STORE command alters the values associated with particular keys for a message in the mailbox. As is the case for the FETCH command, any generic, canonical or concrete key may be used to index the value provided. In addition to these, the following pre-defined keys are provided.

FLAGS — Replace the flags for the message with the argument (in flag list format). The server must respond with a solicited STORE FLAGS message, showing the new state of the flags after the store.

+FLAGS — Add the flags in the argument to the message's flag list. The server must respond with a solicited STORE FLAGS message, showing the new state of the flags after the store.

-FLAGS — Remove the flags in the argument from the message's flag list. The server must respond with a solicited STORE FLAGS message, showing the new state of the flags after the store.

RFC822.HEADER — Replace the header of the message(s) with that specified. This allows users to use their mailboxes as databases with header fields as keys. The server must respond with solicited STORE RFC822.HEADER, STORE RFC822.SIZE and STORE ENVELOPE messages, showing the new state of the reparsed header after the store.

RFC822.TEXT — Replace the body of the messages with that specified. The server must respond with solicited STORE RFC822.TEXT and STORE RFC822.SIZE messages, showing the new state of the message after the store.

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

The server is required to send a solicited STORE response for

each store operation that results in a format transformation by the server. For example, the server is required to send a STORE FLAGS response when the client performs a STORE +FLAGS or a STORE -FLAGS, since the client may not easily be able to know what the result of this command will be. Similarly, if the client emits a STORE FROM command then the server should respond with a suitable STORE FROM response because the client would be sending a string value to be stored and the server should transform this into a set of addresses. In general, however, although it is legal for the server to send a solicited STORE response for each STORE operation, this is discouraged, since it might result in the retransmission of very large and unnecessary amounts of data that have been stored.

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 server response SEARCH (criteria) (numbers) gives the set of messages which match the conjunction of the criteria specified. In addition to each of the search criteria there is its logical inverse. The logical inverse criterion is denoted by the ~ (tilda) sign.

Thus, no message that matches the criterion:
FROM crispin

will match the criterion:
~FROM crispin

The criteria for the search can be any generic, canonical or concrete key. In addition to these, the following pre-defined keys are also provided:

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.
HEADER string	Messages which contain the specified string in the message header.
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.

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 mailbox since October 1, 1987.

Implementation note: The UNANSWERED, UNDELETED, UNFLAGGED,

UNKEYWORD and UNSEEN criteria, described below, are preserved in IMAP3 for IMAP2 compatibility. They are, however, considered obsolete and new Client programs are encouraged to use the ~ notation for the logical inverses of search criteria with a view to the dropping of this outmoded syntax in later versions.

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.

tag READONLY

The READONLY command indicates that the client wishes to make the mailbox read-only. The server is required to reply with a solicited READONLY or READWRITE response.

tag READWRITE

The READWRITE command indicates that the client wishes to make the mailbox read-write. The server is required to reply with a solicited READONLY or READWRITE response.

tag SUPPORTED.VERSIONS

The SUPPORTED.VERSIONS solicits from the server a SUPPORTED.VERSIONS message, which encapsulates information about which versions and features the server supports.

tag SELECT.VERSION (major_version minor_version)

The SELECT.VERSION command indicates that the client wishes to select certain behavior on the part of the server. The major and minor versions indicate the specific version of the protocol being selected.

EXAMPLE: A002 SELECT.VERSION (3 0)

A client may not request a server version that is not supported by

the server, i.e., which is specifically mentioned in the response to a SUPPORTED.VERSIONS command. An attempt to do so by a client will result in a NO response from the server. It is an error for the SELECT.VERSION command to be used after a mailbox has been selected. The rationale for this is that for some server implementations it might be necessary to spawn separate programs to implement widely divergent protocol versions. Thus, the client cannot be allowed to expect any server state to be preserved after the use of the SELECT.VERSION command. The default version of all servers is 2.0, i.e., IMAP2 as defined by RFC 1064.

tag SELECT.FEATURES 1#features

The SELECT.FEATURES command indicates that the client wishes to select certain specific features on the part of the server. A client may not request a feature that is not supported by the server, i.e., one that is explicitly mentioned in the set of features for the selected version returned by the SUPPORTED.VERSIONS command. An attempt to do so by a client will result in a NO response from the server.

EXAMPLE: A002 SELECT.FEATURES AUTO.SET.SEEN ~TAGGED.SOLICITED
EIGHT.BIT.TRANSPARENT

i.e., select the set of features called AUTO.SET.SEEN and EIGHT.BIT.TRANSPARENT and deselect the feature called TAGGED.SOLICITED. The use of the SELECT.FEATURES command completely resets the set of selected features. Note: These are only example feature names and are not necessarily supported by any server. See the appendix on features for more information on features. Note: Some features, when present in the server, will cause the upwards compatible extension of the grammar, i.e., by adding extra commands. The server is at liberty not to remove these upwards compatible extensions to the command tables when a feature is disabled. Thus, it is an error for a client to rely on getting a NO or BAD response in any way, for instance to determine the selectedness or presence of a feature.

tag BBOARD bboard

The BBOARD command is equivalent to SELECT, except that its argument is a bulletin board (BBoard) name. The format of a BBoard 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 BBoard name be a mailbox name or a file name (in particular, Unix netnews has a completely different namespace from mailbox or file names).

The result from the BBOARD command is identical from that of the SELECT command. For example, in the TOPS-20 server implementation, the command

```
A0002 BBOARD FOO
```

is exactly equivalent to the command

```
A0002 SELECT POBOX:<BBOARD>FOO.TXT
```

Note: the equivalence in this example is *not* required by the protocol, and merely reflects the fuzzy distinction between mailboxes and BBoards on TOPS-20.

tag FIND (BBOARDS / MAILBOXES) pattern

The FIND command accepts as arguments the keywords BBOARDS or MAILBOXES and a pattern which specifies some set of BBoard/mailbox names which are usable by the BBOARD/SELECT command. Two wildcard characters are defined; "*" specifies that any number (including zero) characters may match at this position and "%" specifies that 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; furthermore, "*" will match all BBoards/mailboxes. The following quoting convention applies to wildcards: "*" is the literal "*" character, "\%" is the literal "%" character and "\\" is the literal "\" character. Notes: The format of mailboxes is server implementation dependent. The special mailbox name INBOX is not included in the output to the FIND MAILBOXES command.

The FIND command solicits any number of BBOARD or MAILBOX responses from the server as appropriate.

Examples:

```
    A0002 FIND BBOARDS *
    A0002 BBOARD FOOBAR
    A0002 BBOARD GENERAL
    A0002 OK FIND completed
```

or

```
    A0002 FIND MAILBOXES FOO%BA*
    A0002 MAILBOX FOO.BAR
    A0002 MAILBOX FOO.BAZZAR
    A0002 OK FIND completed
```

Note: 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 solicited reply be usable in the SELECT command without remembering any path specification which may have been used in the FIND MAILBOXES pattern.

tag FLAGS

The FLAGS command solicits a FLAGS response from the server.

tag SET.FLAGS flag_list

The SET.FLAGS command defines the user specifiable flags for this mailbox, i.e., the keywords. If this set does not include flags formerly sent to the client by the server in a FLAGS message then this constitutes a request to delete the flag. Any new flags should be created. This command does not affect the system defined flags and any system flags that are included in the flag_list will be ignored. The server must respond to this command with a solicited FLAGS message. If the deletion of a flag results in the invalidation of the flag sets of any messages then the server is required to send solicited STORE FLAGS messages to the client for each modified message.

Responses:

*/tag OK text

In its solicited form 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.

In its unsolicited form, this response indicates simply 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. This usage, although legal, is by no means required. The text is a line of human-readable text which may be logged in protocol telemetry.

*/tag NO text

In its solicited form 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.

In its unsolicited form 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.

*/tag BAD text

In its solicited form response indicates faulty protocol received from the client and indicates a bug. 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.

In its unsolicited form 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, and examination of the protocol telemetry is advised to determine the cause of the problem.

*/tag BYE text

This 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. IMAP2 requires that the server emit a solicited BYE response as part of a normal logout sequence. This solicited form is not required under IMAP3, though is still legal for compatibility. In its unsolicited form the BYE response is used as a panic shutdown announcement by the server. It is required to be used by any server which performs autologouts due to inactivity.

*/tag number message_data

The solicited (tag number message_data) response is generated as the result of a number of client requests. The server may also emit any the following at any time as unsolicited data (i.e., * number message_data). 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 selected with the SELECT command or equivalent.

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. The server must send a solicited EXPUNGE response for each message that it expunges as the result of an EXPUNGE command. Note: future versions of the protocol may allow the use of a message sequence as a value returned by the EXPUNGE response to allow the

more efficient compaction of client representations of mailboxes.

STORE data
: Functionally equivalent to FETCH, only it is sent by the server when the state of a mailbox changes. The server must send solicited STORE responses as the result of any change caused by a STORE command.

FETCH data
: This is the principle means by which data about a message is sent to the client. The data is in a Lisp-like S-expression property list form. Just as the FETCH request from the client can fetch any generic, canonical or concrete key, so also the FETCH response can return values for any of these keys as well as for the pre-defined attributes mentioned below. Note that the server is permitted to send any unsolicited FETCH or STORE messages that it should choose, be they the values associated with generic, canonical or concrete keys. Clients are required to ignore any such FETCH responses that it cannot interpret. For example, clients are not required to be able to understand, i.e., use fruitfully, the canonical $TO key, but they are required to be able to ignore an unsolicited $TO message correctly.

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 (i.e., the at-domain-list in SMTP), mailbox name, host name and comments. Implementation note: The addition of the comment field is an incompatible extension from IMAP2. The server is required not to provide

this field when running in IMAP2 mode.

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 mailbox
\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.
Note: Some implementations of IMAP2 servers had the (undocumented) behavior of setting the \SEEN flag as a side effect of fetching the body of a message. This resulted in erroneous behavior for clients that prefetch messages that the user might not get around to reading. Thus, this behavior is explicitly disallowed in IMAP3.
Note: this is not a significant performance restriction because it is always possible for IMAP3 clients to use an interaction with the server of the following type:

```
A001 FETCH 42 RFC822
A002 STORE 42 +FLAGS (\SEEN)
A001 42 FETCH RFC822 {637} ......
A001 OK Fetch completed
A002 42 STORE FLAGS (\SEEN \FLAGGED...)
A002 OK Store Completed.
```

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. See also note for RFC822.

*/tag FLAGS flag_list

A solicited FLAGS response must occur as a result of a SELECT command. The flag list is the list of flags (at a minimum, the IMAP defined flags) which are applicable for this mailbox. Flags other than the system flags are a function of the server implementation.

*/tag SEARCH (numbers) (search_criteria)

This response occurs as a result of a SEARCH command. The number(s) refer to those messages which match the search criteria. In its solicited form this message allows clients to find interesting groups of messages, e.g., unseen messages from Crispin. In its unsolicited form it allows the server to inform the client of interesting patterns, e.g., when new mail arrives, recent and from Crispin. Compatibility note: The search_criteria are sent by the server along with the matching numbers so unsolicited SEARCH messages may be interpreted. This syntax is not upwards compatible with IMAP2 and so the new syntax is intended to make it simple for clients that are not able to take advantage of unsolicited SEARCH messages still to interpret solicited SEARCH messages simply by ignoring everything that follows the list of numbers with minimal parsing. Such clients may not, however, simply discard the rest of the line because there might be LITERALs in the search pattern.

Examples:
A00042 SEARCH (2 3 6) (FROM Crispin ~SEEN)
and
* SEARCH (42) (FROM Crispin RECENT)

*/tag READONLY

This indicates that the mailbox is read-only. The server is required to respond to a READONLY or READWRITE command with either a solicited READONLY or a solicited READWRITE response. Note: If the client attempts a mutation operation, such as STORE, on a mailbox to which it does not have write access then the server is required to reply with a solicited READONLY response on the first

such attempted mutation. The server may also choose to send solicited READONLY responses for each subsequent attempted mutation.

*/tag READWRITE

This indicates that the mailbox is read-write. The server is required to respond to a READONLY or READWRITE command with either a solicited READONLY or a solicited READWRITE response.

*/tag BBOARD bboard_name

This message is produced in its solicited form as a response to a FIND BBOARDS command. In its unsolicited form it represents a notification by the server that a new BBoard has been added. Bboard_name must be a name that can be supplied to the BBOARD command so as to select the appropriate bboard.

*/tag MAILBOX non_inbox_mailbox_name

This message is produced in its solicited form as a response to a FIND MAILBOXES command. In its unsolicited form it represents a notification by the server that a new mailbox has been added, perhaps as the result of a COPY command creating a new mailbox. Non_inbox_mailbox_name must be a name that can be supplied to the SELECT command so as to select the appropriate mailbox. Note: non_inbox_mailbox_name is never the string "INBOX".

*/tag SUPPORTED.VERSIONS (version_specs)

This message is used either as a response to the SUPPORTED.VERSIONS or, in its unsolicited form, to indicate the dynamic addition or removal of support for features or protocol versions. Each version_spec is of the form (4 2 EIGHT.BIT.TRANSPARENT AUTO.SET.SEEN ...), i.e., a major version number and a minor version number for the protocol and the set of features supported under the server's implementation of that protocol version. A server may not dynamically remove support for any version or feature that has been selected by any currently logged in client by the use of the VERSION command.

Example:

```
A00005 SUPPORTED.VERSIONS ((2 0 )
       (2 2 TAGGED.SOLICITED)
       (3 0 EIGHT.BIT.TRANSPARENT TAGGED.SOLICITED))
```

Indicates that two major versions are supported and one minor version is supported and that tagged solicited messages are

supported in versions 2.2 and 3.0 with eight bit characters being supported under version 3. For each feature mentioned in the list of features there is also always the negation of that feature. For example, if the server supports the TAGGED.SOLICITED feature then it also supports the ~TAGGED.SOLICITED feature, which disables this feature. Note: These are only example feature names and are not necessarily supported by any server. See the appendix on features for more information on features.

+ 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 commands likely to contain literals are the LOGIN command and the STORE RFC822.HEADER or STORE RFC822.TEXT commands. 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 IMAP3 session

The following is a transcript of an actual IMAP3 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 Interactive Mail Access Protocol
               III Service 6.1(349) at Mon, 14 May 90 14:58:30 PDT
U:     a001 SUPPORTED.VERSIONS
S:     * SUPPORTED.VERSIONS ((2 0 ) (3 0 EIGHT.BIT.TRANSPARENT
                 AUTO.SET.SEEN TAGGED.SOLICITED))
```

```
   S:     A001 Supported Versions returned.
   U:     a002 SELECT.VERSION (3 0)
   S:     a002 OK Version 3.0 Selected.
   U:     a003 SELECT.FEATURES TAGGED.SOLICITED
   S:     a003 OK Features selected.
   U:     a004 login crispin secret
   S:     a004 OK User CRISPIN logged in at Thu, 9 Jun 90 14:58:42 PDT,
                  job 76
   U:     a005 select inbox
   S:     a005 FLAGS (Bugs SF Party Skating Meeting Flames Request AI
                  Question Note \XXXX \YYYY \Answered \Flagged \Deleted
                  \Seen)
   S:     a005 16 EXISTS
   S:     a005 0 RECENT
   S:     a006 OK Select complete
   U:     a006 fetch 16 all
   S:     a006 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" NIL)) (("Larry Fagan" NIL "FAGAN"
              "SUMEX-AIM.Stanford.EDU" NIL)) (("Larry Fagan" NIL "FAGAN"
              "SUMEX-AIM.Stanford.EDU" NIL)) ((NIL NIL "rindflEISCH"
              "SUMEX-AIM.Stanford.EDU" NIL)) NIL NIL NIL
              "<12403828905.13.FAGAN@SUMEX-AIM.Stanford.EDU>"))
   S:  a006 OK Fetch completed
   U:  a007 fetch 16 rfc822
   S:  a007 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:  a007 OK Fetch completed
   U:  a008 logout
   S:  a008 BYE UNIX IMAP III server terminating connection
```

```
S:  a008 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 beginning distribution of a client for TI Explorer Lisp machines. 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 is as a part of a MacII client which has now been under daily use (by real users) at Stanford for quite some time.

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 IMAP3 server and MM-20. The 4.2/3 BSD Unix server requires that the user use either Unix Mail format or mail.txt format which is compatible with SRI MM-32 or Columbia MM-C. The 4.2/3 BSD Unix server allows simultaneous read access; write access must be exclusive. There is also an experimental IMAP3 server running on the TI Explorer class of machine, which uses MM mailbox format and which can communicate over both TCP and Chaos.

The Xerox Lisp client and DEC-20 server have been in production use for over two years; the Unix server was been in production use for over a year. IMAP3 has been used to access mailboxes at remote sites from a local workstation via the Internet. For example, from the Stanford local network one of the authors has read his mailbox at a Milnet site.

A number of IMAP clients have now been developed or are being developed. Amongst these are versions that run on the following machines:

- Xerox Lisp machines
- Apple Macintosh
- NeXT
- IBM PC
- TI Explorer Lisp machines
- "Glass teletype" version that runs under Unix

. GNU Emacs
. X Windows
. NTT ELIS

Each of these client programs is carefully tuned to optimize the performance and user interface in a manner that is consistent with the the user interface model of the native machine. For example, the Macintosh client features a "messy-desk" interface that allows the cutting and pasting of text with the use of the clipboard with a menu driven interface with keyboard accelerators.

This specification does not make any formal definition of size restrictions, but some of the existing servers have the following limitations:

DEC-20
. 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

TI-Explorer
. length of a mailbox: limited by the Minimum of the size of the virtual address space and the size of the file system
. maximum number of messages: limited by the the size of the virtual address space
. length of a command line: limited by the the size of the virtual address space
. length of the local host name: limited by the the size of the virtual address space
. length of a "short" argument: limited by the the size of the virtual address space
. length of a "long" argument: limited by the the size of the virtual address space
. maximum amount of data output in a single fetch: not limited

Typical values for these limits are 30Mb for file systems and 128Mb for virtual address space.

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 solicited

responses and unsolicited data. First, the infamous synchronization problems of SMTP and similar protocols do not happen with tagged commands; a command is not considered satisfied until a completion acknowledgement with the same tag is seen. Tagging allows an arbitrary amount of other responses ("solicited" 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. Some clients might be implemented in a thoroughly asynchronous manner, having, perhaps, multiple outstanding commands at any given time. Note: this does not require that the server process these commands in anything other than a lock-step manner. It simply allows clients to take advantage of servers that are able to do such asynchronous operations.

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 IMAP3 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, although a "fetch" command is a request for specific information from the client, the server is always at liberty to include more than the desired data as "unsolicited". A server acknowledgement to the "fetch" is a statement that at least all the requested data has been sent.

In terms of implementation, a simple lock-step client may have a local cache of data from the mailbox. This cache is incomplete in general, and at select time is empty. A listener on the IMAP connection in the client processes all solicited and unsolicited data symmetrically, and updates the cache based on this data, i.e., the client faults on a cache miss and asks the server to fill that cache slot synchronously. If a tagged completion response arrives, the listener unblocks the process which sent the tagged request.

Clearly, given this model it is not strictly necessary to distinguish

most solicited from unsolicited data. Doing so, however, apart from being clearer, also allows such simplistic, lock-step client implementations that extract the specific value of the response to command by trapping the tagged response. This allows the client not to have to block on some complex predicate that involves watching to see an update in a cache cell.

For example, perhaps as a result of opening a mailbox, solicited data from the server arrives. The first piece of data is the number of messages. This is used to size the cache; note that, if new mail arrives, by sending a new "number of messages" unsolicited data message server will cause the cache to be re-sized. 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 solicited 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 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_comment ")"

addr_adl        ::= nil / string

addr_comment    ::= nil / string

addr_host       ::= nil / string

addr_mailbox    ::= nil / string

addr_name       ::= nil / string

bboard          ::= "BBOARD" SP bboard_name

bboard_name     ::= string

bboard_notify   ::= "BBOARD" sp bboard_name

canonical_key   ::= "$CC" /  "$FROM" / "$SUBJECT" / "$TO"
```

```
check           ::= "CHECK"

concrete_key    ::= string

copy            ::= "COPY" SP sequence SP mailbox

criterion       ::= "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" / key SP string

criteria        ::= 1#criterion

data            ::= ("FLAGS" SP flag_list /
                     search_notify / bboard_notify / mailbox_notify /
                     supported_versions_notify / "READONLY" / "READWRITE" /
                       "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_length      ::= NUMBER

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"

feature         ::= ATOM

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

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

find            ::= "FIND" ("BBOARDS" / "MAILBOXES") pattern

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

flags           ::= "FLAGS"

generic_key     ::= "BCC" / "BODY" / "CC" / "FROM" / "HEADER" / "SIZE" /
                    "SUBJECT" / "TEXT" / "TO"

key             ::= generic_key / canonical_key / concrete_key

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

login           ::= "LOGIN" SP userid SP password

logout          ::= "LOGOUT"

mailbox         ::= "INBOX" / string

mailbox_notify ::= MAILBOX non_inbox_mailbox_name

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
                     env_length 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 / key SP string_list) ")"

msg_recent      ::= "RECENT"

msg_num         ::= NUMBER

nil             ::= "NIL"

non_inbox_mailbox_name ::= string

noop            ::= "NOOP"

numbers         ::= 1#NUMBER

password        ::= string

pattern         ::= string

recent_flag     ::= "\RECENT"

read_only       ::= "READONLY"

read_write      ::= "READWRITE"

ready           ::= "+" SP text_line

request         ::= tag SP (noop / login / logout / select / check /
                    expunge / copy / fetch / store / search /
                    select_version / select_features /
                    supported_versions / bboard / find /
                    read_only / read_write / flags / set_flags ) CRLF

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

search          ::= "SEARCH" SP criteria

search_notify   ::= "SEARCH" SP (numbers) SP (criteria)

select          ::= "SELECT" SP mailbox

select_features ::= "SELECT.FEATURES" 1#feature

select_version  ::= "SELECT.VERSION" SP "(" NUMBER SP NUMBER ")"

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

```
set_flags       ::= "SET.FLAGS" SP flag_list

solicited       ::= tag SP (msg_num SP msg_data / data /
                            solicited_only) CRLF

solicited_only  ::=                {None currently defined}

store           ::= "STORE" SP sequence SP store_att

store_att       ::= ("+FLAGS" SP flag_list / "-FLAGS" SP flag_list /
                    "FLAGS" SP flag_list / RFC822.TEXT SP string
                    / RFC822.HEADER SP string / key SP string)

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

string_list     ::= string / ("(" 1#string ")")

supported_versions ::= "SUPPORTED.VERSIONS"

supported_versions_notify ::= "SUPPORTED.VERSIONS" "(" 1#version_spec
                              ")"

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

tag             ::= atom

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

userid          ::= string

version_spec    ::= "(" NUMBER SP NUMBER SP 1#feature ")"
```

Appendix: Features.

In this section we outline the standard features that are supported by all IMAP3 servers and identify those features which are recommended or experimental. For each of these features the default setting is specified. This means that it is required of any server that supports a given feature to make the default enabledness of that feature as is specified below. It is required that for each feature supported by a server the inverse feature should also be supported. The inverse feature name shall always be defined as the feature name preceded by the "~" character. Thus, the AUTO.SET.SEEN feature is disabled by the ~AUTO.SET.SEEN feature.

Required Features:

AUTO.SET.SEEN - When this features is enabled (default is disabled), the \\SEEN flag is set for all appropriate messages as a side effect of any of the following:

FETCH of RFC822
FETCH of RFC822.TEXT
COPY

Justification: This feature is provided for the use of clients that are unable to pipeline their commands effectively and communicate over high latency connections. When disabled, the server will not perform any such side effects. This feature is also provided so as to smooth the transition from IMAP2 to IMAP3.

TAGGED.SOLICITED - When this feature is enabled (default is enabled for IMAP3, disabled for IMAP2 mode), solicited responses from the server will have the tag specified by the client. When this feature is disabled, solicited responses from the server will have the IMAP2 compatible tag "*", not the tag specified by the client.
Justification: This feature is provided so as to smooth the transition from IMAP2 to IMAP3.

Recommended Features.

EIGHT.BIT.TRANSPARENT - When this feature is enabled (default is disabled), the server allows the transparent transmission of eight bit characters. When this feature is disabled, the value of any bit other than the least significant 7 bits transmitted by the server is unspecified. If this feature is enabled, the characters that compose all command keywords specified in the IMAP3 grammar and all feature names use only their 7 least significant bits.
Justification: This feature is provided for the purpose of supporting national character sets within messages, encoded languages such as Japanese Kanji characters and also of binary data, such as programs, graphics and sound.

NEW.MAIL.NOTIFY - When this feature is enabled (default is disabled for compatibility with the majority of existing IMAP2 servers), the server will notify the client of the arrival of new mail in the currently selected mailbox using the appropriate RECENT and EXISTS unsolicited messages without the client needing to send periodic CHECK commands.
Justification: This feature is provided to allow clients to

switch off any periodic polling strategy that they may use to look for new mail. Such polling unnecessarily uses bandwidth and can cause the interactive performance to degrade because the user can be kept waiting while some background process is doing a CHECK.

SEND - When this feature is enabled (default is disabled) a new "SEND" command becomes available to the client. The SEND command instructs the server to send a message, rather than requiring the client to use its own, local message sending capability, for example. An example of of the send command might be as follows:

```
tag42 SEND RFC822 {2083}
From: James Rice <Rice@Sumex-Aim.Stanford.Edu>
To:.....
```

If the server is unable to parse the message being sent then it is required to issue a suitable NO notification to the client. If the message cannot be delivered for some reason then the server should send a suitable message to the FROM: address of the message detailing the delivery failure.
When the SEND feature is enabled, the "send" production in the grammar is added and as defined below. The "send" request is added to the list of requests in the request production also as shown below:

```
message_format  ::= RFC822

request         ::= tag SP (noop / login / logout / select / check /
                    expunge / copy / fetch / store / search /
                    select_version / select_features /
                    supported_versions / bboard / find /
                    read_only / read_write / flags /
                    set_flags / send) CRLF

send            ::= SEND SP message_format SP string
```

Justification: This feature is provided so that mail can be sent by the same reliable server that is used for the storage of mail. This has, amongst others, the following benefits:

- Single process clients need not be delayed by mail transmission.
- Mail sent by the client will have the server named as the message's sender. This can be important because there are a lot of mailers that erroneously cause reply mail to be sent to the Sender, not the From or Reply-To address. Since the client in general is not listening for mail being sent to it directly this can cause mail to be lost.

- Clients can be written that do not have any native message sending capability.

ADD.MESSAGE - When this feature is enabled (default is disabled) a new "ADD.MESSAGE" command becomes available to the client. The ADD.MESSAGE command instructs the server to add the specified message to the designated mailbox. This command can be thought of as being like a COPY command except in this case the message that is put in the designated mailbox is specified as a string, rather than as a message number to be copied from the currently selected mailbox. An example use of this command might be as follows:

```
tag42 ADD.MESSAGE OUTGOING-MAIL RFC822 {2083}
From: James Rice <Rice@Sumex-Aim.Stanford.Edu>
To:.....
```

This will have the effect of adding the message to the mailbox called OUTGOING-MAIL.
If the server is unable to parse the message being added then it is required to issue a suitable NO notification to the client.
When the ADD.MESSAGE feature is enabled, the "add_message" production in the grammar is added and as defined below.
The "add_message" request is added to the list of requests in the request production also as shown below:

```
add_message          ::= ADD.MESSAGE SP mailbox SP format SP string

message_format  ::= RFC822

request         ::= tag SP (noop / login / logout / select / check /
                    expunge / copy / fetch / store / search /
                    select_version / select_features /
                    supported_versions / bboard / find /
                    read_only / read_write / flags / set_flags /
                    add_message) CRLF
```

Justification: This feature is provided so that clients can easily add mail to specific mailboxes. This allows clients to implement such behavior as outgoing mail storage (BCC) without the need to resort to mailing to special BCC mailboxes.

RENUMBER - When this feature is enabled (default is disabled) the RENUMBER command becomes available to the client.
The RENUMBER command will reorder the assignment of message numbers to the messages in the mailbox. If this results in a change to the association of any message number with any message then the server is required to send solicited RESET

responses to the client. The intent of this command is to allow users to view mailboxes in user-meaningful order efficiently. While the client could do the ordering, it would be less efficient in general. Note that the server may or may not change the actual storage of the messages and the ordering may or may not remain in effect after another mailbox is selected or the IMAP session is terminated. Informally, the syntax for the RENUMBER command is:

```
    tag RENUMBER field_name ordering_type
```

this has the effect of changing the IMAP grammar to be as follows:

```
ordering_type   ::= DATE / NUMERIC / ALPHA

renumber        ::= RENUMBER SP field_name SP ordering_type

request         ::= tag SP (noop / login / logout / select / check /
                    expunge / copy / fetch / store / search /
                    select_version / select_features /
                    supported_versions / bboard / find /
                    read_only / read_write / flags / set_flags /
                    renumber) CRLF
```

For example:

```
 tag42 RENUMBER FROM ALPHA
                    ;;;RENUMBER alphabetically by the from field
 tag42 RESET 10:20,49
                    ;;;Messages 10 to 20 and 49 have changed
 tag42 OK RENUMBER finished.  Sequence has changed
 tag43 FETCH ALL 10:20,49
                    ;;;Client chooses to fetch the changed msgs.
```

To support this the RESET message is defined as follows:

*/tag RESET message_sequence
This solicited of unsolicited message from the server informs the client that it should flush any information that it has retained for the specified messages.

Justification: This feature is provided so that clients can view mailboxes in an order that is convenient to the user. This is particularly important in the context of mailboxes that the user copies messages to from other mailboxes. This user-controlled filing process often does not happen in any well-defined order. Because messages in a mailbox are

implicitly ordered (usually by arrival date, though this is not a required ordering predicate), the user can be confused by the apparent order of messages in the mailbox. The addition of the RENUMBER command makes it unnecessary for the user to leave IMAP and use some other mail system to sort mailboxes.

ENCODING - When this feature is enabled (default is disabled) a new generic key named ENCODING is defined. The value associated with the generic ENCODING key is a list of (tag encoding-type options...) lists that represent the ordered, possibly encoded body of the message. Each such list represents a segment of the body of the message and the way in which it is encoded. Any options that follow the encoding_type are further qualifiers that describe the format of the segment. Each tag is created by the server and is unique with respect to the other tags allocated for the other elements in the ENCODING list. The client may use the tags returned by the server as concrete keys to access a field which is encoded using the encoding type and options mentioned in the appropriate list. Thus:

```
tag41 FETCH 196 ENCODING ; Client asks for encoding field of msg 196.
tag41 FETCH ENCODING NIL ; Server replies. This message is not encoded.
tag41 OK Fetch completed.
tag42 FETCH 197 ENCODING ; Client asks for encoding field of msg 197.
tag42 FETCH ENCODING ((G001 UUENCODE) (G002 HEX)) ; Server replies.
tag42 OK Fetch completed.
tag43 FETCH 197 G002     ; Client asks for field named G002
tag43 FETCH G002 "A0 00 FF 13 42......." ; Server sends value of field.
tag43 OK Fetch completed.
```

or

```
tag44 STORE 197 G002 "0A 00 FF 31 24......."
   ; Store back the segment with nibbles swapped
```

Note: As a side-effect of enabling this feature, the generic key TEXT will be redefined so as to return only those body parts of a message that are of type TEXT. The concrete key RFC822.TEXT, on the other hand, would still return everything in the body of the message, even if it was full of strange, binary character sequences.

When the client STOREs to a field denoted by one of the above tags the server will interpret the value being passed as being in the same format as is currently specified in the ENCODING field. The

server is not required to be able to reformat the data associated with the ENCODING tags if the client STOREs a new value for the ENCODING field. The interpretability of a message in the context of its ENCODING field is undefined if the client side-effects that ENCODING field, unless the client also STOREs new, reformatted values for the fields that have had their encoding changed.

If the client stores a new value for the ENCODING field then the tags in the new value will be used to index the parts of the body. All tags in a client-STOREd ENCODING that are the same as those originally generated by the server in response to a FETCH ENCODING command are said still to denote the fields that they originally denoted, though possibly reordered. Any tags not originally defined by the server will denote new message parts, in the appropriate format, in the relative position specified. The exclusion of any tags that the server originally defined in a FETCH of the ENCODING field will indicate the deletion of that part of the message. Newly created message parts are undefined by default, so if the client fails to follow the STOREing of the ENCODING field with suitable STORE commands for the values associated with any newly created tags, these fields will contain the null value NIL.

Justification: This feature is supplied so as to allow support for emergent multi-part and multi-media mail standards.

INDEXABLE.FIELDS - When this feature is enabled (default is disabled) the grammar of fetch commands is changed to allow the client to select a specific subsequence from the field in question. For example:

```
tag42 FETCH 197 BODY 2000:3999
```

would fetch the second two thousand bytes of the body of message 197. This feature allows resource limited clients to access small parts of large messages. The formal syntax for this is:

```
fetch_att       ::= "ENVELOPE" / "FLAGS" / "INTERNALDATE" /
                    fetch_key / (fetch_key SP NUMBER ":" NUMBER)

fetch_key       ::= "RFC822" / "RFC822.HEADER" / "RFC822.SIZE" /
                    "RFC822.TEXT" / key
```

If the lower bound number (the number to the left of the colon) exceeds the maximum size of the field then the empty string is returned. If the upper bound exceeds the maximum size of the field but the lower bound does not then the server will return the remaining substring of the field after the lower bound. The

bounds specified are zero indexed into the fields and the bounds index fields by 8-bit bytes.

Justification: This feature is provided so as to allow resource-limited clients to read very large messages and also to allow clients to improve interactive response for the reading of large messages by fetching the first "screen full" of data to display immediately and fetching the rest of the message in the background.

SET.EOL - When enabled (default is disabled), this feature allows the new command SET.EOL to be available, changing the grammar as follows:

```
character       ::= "CR" / "LF" / number

request         ::= tag SP (noop / login / logout / select / check /
                    expunge / copy / fetch / store / search /
                    select_version / select_features /
                    supported_versions / bboard / find /
                    read_only / read_write / flags / set_flags /
                    set_eol) CRLF

set_eol         ::= "SET.EOL" 1#character
```

This has the effect of changing the end of line character sequence generated by the server for newlines within strings to the sequence of characters specified. The characters in the sequence can be either the specified symbolically named characters or a numerical value, specifying the decimal value of the character to use. Thus, if the client would like newlines in strings to be indicated by a carriage return followed by a control-d, the client would issue the following command:

```
tag42 SET.EOL CR 4
```

If the server is unable to support the combination of characters requested by the client as its end-of-line pattern it will reply with a NO response. This might be the case, for example, if a server is only able to generate its own native line feed pattern and the CRLF required by IMAP by default.

The server is required to change any length denoting values, such as envelope byte counts for all future transactions to reflect the new eol setting. This change in reported sizes should apply to all generic size fetching keys, but not to concrete ones such as RFC822.SIZE, which by their very nature require a size measurement in RFC822 format, i.e., with CRLF as the end-of-line convention.

Justification: This feature is provided because frequently clients and servers might have end-of-line conventions other than the CRLF specified by RFC822. It is undesirable that the IMAP be linked too closely to RFC822 and selecting a different convention might allow substantial performance improvements in both clients and servers by saving either client, server or both from having to shuffle text around so as to add or remove non-local end-of-line sequences.

Acknowledgements:

This text is based on RFC 1064 by Mark Crispin.

The following have made major contributions to this proposed update to the IMAP2 protocol:

James Rice <Rice@sumex-aim.stanford.edu>
Richard Acuff <acuff@sumex-aim.stanford.edu>
Bill Yeager <yeager@sumex-aim.stanford.edu>
Christopher Lane <lane@sumex-aim.stanford.edu>
Bjorn Victor <Bjorn.Victor@docs.uu.se>

Additional input was also received from:

Andrew Sweer <sweer@sumex-aim.stanford.edu>
Tom Gruber <Gruber@sumex-aim.stanford.edu>
Kevin Brock <Brock@Sumex-Aim.Stanford.Edu>
Mark Crispin <MRC@cac.washington.edu>

Security Considerations

Security issues are not discussed in this memo.

Author’s Address

James Rice
Stanford University
Knowledge Systems Laboratory
701 Welch Road
Building C
Palo Alto, CA 94304

Phone: (415) 723-8405
EMail: RICE@SUMEX-AIM.STANFORD.EDU