Network Working Group   Mark Crispin
Internet Draft: IMAP2bis   University of Washington
Obsoletes: RFC 1176, 1064   October 1993
Document: internet-drafts/draft-ietf-imap-imap2bis-02.txt

INTERACTIVE MAIL ACCESS PROTOCOL - VERSION 2bis

Status of this Memo

This document is an Internet Draft. Internet Drafts are working documents of the Internet Engineering Task Force (IETF), its Areas, and its Working Groups. Note that other groups may also distribute working documents as Internet Drafts.

Internet Drafts are draft documents valid for a maximum of six months. Internet Drafts may be updated, replaced, or obsoleted by other documents at any time. It is not appropriate to use Internet Drafts as reference material or to cite them other than as a "working draft" or "work in progress". Please check the I-D abstract listing contained in each Internet Draft directory to learn the current status of any this or any other Internet Draft.

This is a draft document of the IETF IMAP Working Group. It is a draft specification of the IMAP2bis protocol, based upon the following earlier specifications: unpublished IMAP2bis.TXT document, RFC 1176, and RFC 1064. This document is not a complete or final specification of the IMAP2bis protocol.

Only matters that are believed to be uncontroversial, or issues that are believed to be resolved, appear in this document. The entirety of this document is subject to change and extension. A list of open issues may be found in the file mail/imap.unresolved on Internet site ftp.CAC.Washington.EDU.

A version of this draft document will be submitted to the RFC editor as a Proposed Standard for the Internet Community. Discussion and suggestions for improvement are requested. This document will expire before 31 March 1994. Distribution of this draft is unlimited. Comments are solicited and should be sent to imap@CAC.Washington.EDU.

Introduction

The Interactive Mail Access Protocol, Version 2bis (IMAP2bis) allows a client to access and manipulate electronic mail on a server. IMAP2bis is designed to permit manipulations of remote mailboxes as

if they were local. IMAP2bis includes operations for creating, deleting, and renaming mailbox folders; checking for new mail; permanently removing messages; setting and clearing flags; RFC 822 and MIME parsing; searching; and selective fetching of message attributes, texts, and portions thereof.

IMAP2bis does not specify a means of posting mail; this function is handled by a mail transfer protocol such as SMTP (RFC 821).

IMAP2bis assumes a reliable data stream such as provided by TCP. When TCP is used, an IMAP2bis server listens on port 143.

System Model and Philosophy

There are three fundamental models of client/server email: offline, online, and disconnected use. IMAP2bis can be used in any one of these three models.

The offline model is the most familiar form of client/server email today, and is used by protocols such as POP-3 (RFC 1225) and UUCP. In this model, a client application periodically connects to a server. It downloads all the pending messages to the client machine and deletes these from the server. Thereafter, all mail processing is local to the client. This model is store-and-forward; it moves mail on demand from an intermediate server (maildrop) to a single destination machine.

The online model is most commonly used with remote filesystem protocols such as NFS. In this model, a client application manipulates mailbox data on a server machine. A connection to the server is maintained throughout the session. No mailbox data are kept on the client; the client retrieves data from the server as is needed. IMAP2bis introduces a form of the online model that requires considerably less network bandwidth than a remote filesystem protocol, and provides the opportunity for using the server for CPU or I/O intensive functions such as parsing and searching.

The disconnected use model is a hybrid of the offline and online models, and is used by protocols such as PCMAIL (RFC 1056). In this model, a client user downloads some set of messages from the server, manipulates them offline, then at some later time uploads the changes. The server remains the authoritative repository of the messages. The problems of synchronization (particularly when multiple clients are involved) are handled through the means of unique identifiers for each message.

Each of these models have their own strengths and weaknesses:

Feature	Offline	Online	Disc
Can use multiple clients	NO	YES	YES
Minimum use of server connect time	YES	NO	YES
Minimum use of server resources	YES	NO	NO
Minimum use of client disk resources	NO	YES	NO
Multiple remote mailboxes	NO	YES	YES
Fast startup	NO	YES	NO
Mail processing when not online	YES	NO	YES

Although IMAP2bis was originally designed to accommodate the online model, it can support the other two models as well. This makes

possible the creation of clients that can be used in any of the three models. For example, a user may wish to switch between the online and disconnected models on a regular basis (e.g. owing to travel).

IMAP2bis is designed to transmit message data on demand, and to provide the facilities necessary for a client to decide what data it needs at any particular time. There is generally no need to do a wholesale transfer of an entire mailbox or even of the complete text of a message. This makes a difference in situations where the mailbox is large, or when the link to the server is slow.

More specifically, IMAP2bis supports server-based RFC 822 and MIME processing. With this information, it is possible for a client to determine in advance whether it wishes to retrieve a particular message or part of a message. For example, a user connected to an IMAP2bis server via a dialup link can determine that a message has a 2000 byte text segment and a 40 megabyte video segment, and elect to fetch only the text segment.

In IMAP2bis, the client/server relationship lasts only for the duration of the TCP connection, and mailbox state is maintained on the server. There is no registration of clients. Except for any unique identifiers used in disconnected use operation, the client initially has no knowledge of mailbox state and learns it from the IMAP2bis server when a mailbox is selected. This initial transfer is minimal; the client requests additional state data as it needs.

The Protocol

The IMAP2bis protocol consists of a sequence of client commands and server responses, with server data interspersed between the responses. Unlike most Internet protocols, commands and responses are tagged. That is, a command begins with a unique identifier (typically a short alphanumeric sequence such as a Lisp "gensym" function would generate e.g., A0001, A0002, etc.) called a tag. The response to the command is given the same tag from the server.

Additionally, the server may send an arbitrary amount of "unsolicited data". Unsolicited data is identified by the special reserved tag of "*". The unsolicited data mechanism transmits most data in IMAP2bis. The term "unsolicited data" suggests that the data may have been transmitted without any explicit request by the client for that data. No distinction is made in IMAP2bis between data transmitted as a result of a client command and data that are unilaterally transmitted by the server. One form of unilaterally transmitted data that commonly occurs is an alert of a change to the mailbox made by some process other than the IMAP2bis client or server; for example, changes in the size of the mailbox (new mail) or in the status of individual messages.

There is another special reserved tag, "+", discussed below.

The server must be listening for a connection. When a connection is opened the server sends a greeting message and then waits for commands. This greeting is either a PREAUTH (meaning that the user has already been identified and authenticated by an external mechanism such as rsh) or OK (meaning that the user is not yet authenticated) unsolicited response. The server may also send a BYE unsolicited response and close the connection if it rejects the connection.

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

Once the greeting has been received, the client may begin sending commands. It is not under any obligation to wait for a server response to a command before sending another command, subject to the constraints of underlying flow control. When commands are received the server acts on them and responds with command responses, often interspersed with data.

In general, the command responses do not themselves contain the requested data. Instead, they indicate the completion status of the request. There are three fundamental responses: success (OK), error

(NO), request faulty or not understood (BAD). The effect of a command can not be considered complete until a command response with a tag matching the command is received from the server.

It is not required that a server process a command to completion before beginning processing of the next command, except when the processing of the previous command may affect the results of the next command by changing the state of the current mailbox. This has certain other effects; for example, this implies that an EXPUNGE response can not be transmitted as part of a response to a command that uses sequence numbers, because EXPUNGE results in message numbers being changed.

Client implementations should update their local cache of data with any received unsolicited data, regardless of whether or not the client expected that data. Unlike command completion responses, data are not necessarily associated with a specific command. The tagged command completion response signals that the client cache is now updated with the results of the corresponding command.

If authentication has not yet been completed, it must now be done via the LOGIN command before any access to data is permitted. The only permitted commands before successful authentication are LOGIN, NOOP, and LOGOUT. See the section below on authentication.

Once authenticated, the client must send a mailbox selection command to access the desired mailbox; no mailbox is selected by default. Mailbox names are implementation dependent. However, the word "INBOX" must be implemented to mean the primary or default mailbox for this user, independent of any other server semantics. It is permitted for a server not to have an INBOX if there is no concept of a primary or default mailbox for this user. The name "INBOX" MUST NOT be used for any other purpose.

On a successful selection, the server will send a list of valid flags, number of messages, and number of messages arrived since last access for this mailbox as unsolicited data, followed by an OK response. The client may close access to this mailbox and access a different one with another selection command.

Several flags are predefined in IMAP2bis. All IMAP2bis flags begin with a backslash ("\") character. Servers MUST, at a minimum, support all the predefined flags in this specification. In addition, a server may also have some implementation-defined per-mailbox flags (called, for historical reasons, "keywords") that do not begin with backslash. Clients should use the information from the server's FLAGS response at message selection to determine what flags the server supports.

The client requests mailbox data with FETCH commands, and receives it via the unsolicited data mechanism. Three major categories of mailbox data exist.

The first category is data that are associated with a message as an entity in the mailbox. There are now four such items of data: the "internal date", the "RFC 822 size", the "flags", and the "unique id". The internal date is the date and time that the message was placed in the mailbox. The RFC 822 size is the size in octets of the message, expressed as an RFC 822 text string. The flags are a list of status flags associated with the message. The unique id is an identifier that is guaranteed to refer to this message and to none other in the mailbox and that, unlike IMAP2bis sequence numbers, persists across sessions.

The second category is data that describe the composition and delivery information of a message; that is, information such as the message sender, recipient lists, message-ID, subject, MIME structure, etc. This is the information that is stored in the RFC 822 and MIME headers. In IMAP2bis, the RFC 822 header information that may be fetched is called the "envelope structure" (not to be confused with SMTP envelopes). Similarly, the MIME header information that may be fetched is called the "body structure". A client can use the parsed envelope and body structures and not worry about having to do its own RFC 822 or MIME parsing.

The third category is textual data, some of which are intended for direct human viewing. IMAP2bis defines six such items: RFC822.HEADER, RFC822.HEADER.LINES, RFC822.HEADER.LINES.NOT, RFC822.TEXT, RFC822, and MIME body parts. It is possible to fetch an individual MIME body part of a message without fetching any other data associated with the message.

A simple client can "FETCH RFC822" to get the entire message without any processing. A more advanced client might fetch some combination of the first and second categories of data for use as a presentation menu. Then, when the user wishes to read a particular message, it will fetch the appropriate texts.

Data structures in IMAP2bis are represented as an S-expression list similar to that used in the Lisp programming language. An S-expression consists of a sequence of data items delimited by space and bounded at each end by parentheses. An S-expression may itself contain other S-expressions, using parentheses to indicate nesting. S-expression syntax was chosen because it provides a concise and precise means of expressing nested data (e.g. MIME structures).

The client can alter certain data with a STORE command. As an

example, a message is deleted from a mailbox by setting the \Deleted flag with a STORE command.

Other client operations that can be done to a mailbox include copying messages to other mailboxes, permanently removing deleted messages, checking for updated mailbox state, and searching for messages that match certain criteria. It is also possible to select a different mailbox, create a new mailbox, rename an existing mailbox, or delete an existing mailbox.

The client should end the session with the LOGOUT command. The server returns a "BYE" followed by an "OK", at which point both the client and the server close the connection. If the client closes the network connection without a LOGOUT command, the server should do its normal logout procedures without attempting any further interaction with the client.

Authentication

Pre-authentication is only possible when the connection to the IMAP2bis service is made through some link protocol that provides its own authentication mechanism. It is not used with a TCP connection to port 143.

An example of pre-authentication is the BSD "RSH" protocol, that provides authentication through a "trusted host" facility. Another example would be a manual invocation of an IMAP2bis server from a logged-in timesharing job.

A pre-authenticated IMAP2bis server should recognize that authentication has already happened, and enter the post-login state. In its greeting message, it should use the unsolicited response "PREAUTH" instead of "OK" to indicate that external authentication has taken place.

This is an example of a pre-authentication scenario. In this and all other examples in this document, S: indicates server dialog and C: indicates client dialog.

```
S: * PREAUTH IMAP2bis Server pre-authenticated as user "Smith"
C: A001 SELECT INBOX
S: * FLAGS (\Answered \Flagged \Deleted \Seen)
S: * 19 EXISTS
S: * 2 RECENT
S: A001 OK SELECT complete
```

A connection that is not pre-authenticated is constrained to using the LOGIN command for establishing authentication. Authentication via the LOGIN command is with either a user name and password pair, or with an user identifier and Kerberos authenticator. See the description of the LOGIN command for more details.

Servers may allow non-authenticated access to certain mailboxes or bulletin boards. The convention is to use a LOGIN command with the userid "anonymous". A password is still required. It is implementation-dependent what requirements, if any, are placed on the password and what access restrictions are placed on anonymous users.

Implementations are NOT required to support pre-authentication, Kerberos authentication, or the anonymous convention.

Definitions of Commands and Responses

Summary of Defined Commands and Responses

Commands	Responses
tag NOOP	tag OK resp_text
tag LOGIN user password	tag NO resp_text
tag LOGOUT	tag BAD resp_text
tag CREATE mailbox	* PREAUTH resp_text
tag DELETE mailbox	* OK resp_text
tag RENAME old_mailbox new_mailbox	* NO resp_text
tag FIND MAILBOXES pattern	* BAD resp_text
tag FIND ALL.MAILBOXES pattern	* BYE resp_text
tag FIND BBOARDS pattern	* MAILBOX mstring
tag FIND ALL.BBOARDS pattern	* BBOARD mstring
tag SUBSCRIBE MAILBOX mailbox	* SEARCH 1#number
tag UNSUBSCRIBE MAILBOX mailbox	* FLAGS flag_list
tag SUBSCRIBE BBOARD mailbox	* number EXISTS
tag UNSUBSCRIBE BBOARD mailbox	* number RECENT
tag SELECT mailbox	* number EXPUNGE
tag EXAMINE mailbox	* number FETCH data
tag BBOARD mailbox	* number COPY
tag CHECK	* number STORE data
tag EXPUNGE	+ text
tag COPY sequence mailbox	
tag APPEND mailbox 0#flag literal	
tag FETCH sequence data	
tag PARTIAL msgno data start count	
tag STORE sequence data value	
tag UID AFTER uniqueid	
tag UID COPY sequence mailbox	
tag UID FETCH sequence data	
tag UID STORE sequence data value	
tag SEARCH search_program	
tag x_command arguments	

Note: there is no pairing between commands and responses on the same line. Any command may result in any number (including none at all) of any of responses beginning with "*" (referred to as "unsolicited data"), followed by one tagged response.

Commands

If, during the execution of any command, the server observes that the mailbox size has changed, the server should output an unsolicited EXISTS and RECENT response reflecting the changed size to alert the client. Similarly, any observed change in message status should cause an unsolicited FETCH response with the new flag data.

tag NOOP

The NOOP command returns an OK to the client. By itself, it does nothing else. However, since any command can return a status update as unsolicited data, this command can be used to poll for new mail or for message status updates.

Another possible use of this command is for the client to "ping" the server so that the client and server know that each other are still alive. This is useful with servers that have an inactivity autologout timer.

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.

If a server supports authentication via Kerberos, it may accept the string "@KERBEROS:" followed by the hexadecimal representation of a Kerberos authenticator.

EXAMPLE: The following is a Kerberos login scenario (note that the line breaks in the sample authenticator are for editorial clarity and are not in a real authenticator):

```
S: * OK Kerberos IMAP2bis Server
C: a001 LOGIN smith @KERBEROS:040700414e445245572e434d552e4544550
   038202c868f3890b377fc8266acc1bedb96b80d3fa76489898e74cd1c952dc
   4003ea3428f29f1c470016cf5adc22f939e6deff2747254c1815d5b0b90d4c
   5a2cba21eb0abe32f9acbf568d751bf4cc13f5ba4e6d82c638a8b5421
S: a001 OK [df84a4cb8323454f] Login OK via Kerberos
```

The token in the brackets in the OK response is the Kerberos authentication response, encrypted with the session key in network

byte order and an incremented checksum as in the usual Kerberos procedure.

tag LOGOUT

The LOGOUT command informs the server that the client is done with the session. The server should send an unsolicited BYE response before the (tagged) OK response, and then close the network connection.

Mailbox manipulation commands: CREATE, DELETE, RENAME

These commands permit the manipulation of entire mailboxes.

tag CREATE mailbox

The CREATE command creates a mailbox with the given name. This command returns an OK response only if a new mailbox with that name has been created. It is an error to attempt to create a mailbox with a name that refers to an extant mailbox. Any error in creation will return a NO response.

Creating INBOX is not permitted. If there is a primary or default mailbox for this user, it MUST exist and be called INBOX.

tag DELETE mailbox

The DELETE command deletes a mailbox with the given name. This command returns an OK response only if a mailbox with that name has been deleted. It is an error to attempt to delete a mailbox name that does not exist. Any error in deletion will return a NO response.

A server SHOULD NOT attempt to test that a mailbox is empty before permitting deletion; this would prevent the deletion of a mailbox that for some reason can not be opened or expunged, leaving to possible denial of service problems. Any such checking should be left to the discretion of the client.

Deleting INBOX is not permitted.

tag RENAME old_mailbox new_mailbox

The RENAME command changes the name of a mailbox. This command returns an OK response only if a mailbox with the old name exists and has been successfully renamed to the new name. It is an error to attempt to rename with an old mailbox name that does not exist or a new mailbox name that already exists. Any error in renaming will return a NO response.

Renaming INBOX is permitted. A new, empty INBOX is created in its place.

FIND commands

The FIND commands return some set of unsolicited MAILBOX or BBOARD replies, depending on the type of FIND, that have as their value a single mailbox name.

Three wildcard characters are defined in the pattern argument. "*" specifies any number (including zero) characters may match at this position. "%" and "?" specify 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 and FOO?BAR match only FOO.BAR. "*" will match all mailboxes.

tag FIND MAILBOXES pattern

The FIND MAILBOXES command accepts as an argument a pattern (including wildcards) that specifies some set of mailbox names that the user has declared as being "active" or "subscribed". The exact meaning of this is implementation-dependent, since the concept of a set of "active" or "subscribed" mailboxes that is preserved across sessions may not be meaningful for a particular server or server implementation. If the SUBSCRIBE MAILBOX and UNSUBSCRIBE MAILBOX commands are implemented then this command returns the list manipulated by those commands.

```
EXAMPLE:  C: A002 FIND MAILBOXES *
          S: * MAILBOX FOOBAR
          S: * MAILBOX GENERAL
          S: A002 OK FIND completed
```

tag FIND ALL.MAILBOXES pattern

The FIND ALL.MAILBOXES command is similar to FIND MAILBOXES;

however, it should return a complete list of all mailboxes available to the user. Data are returned as in FIND MAILBOXES.

The special name INBOX is included in the output from FIND ALL.MAILBOXES unless INBOX is not supported by this server or for this user. The criteria for omitting INBOX is whether SELECT INBOX will return failure; it is not relevant whether the user's real INBOX resides on this or some other server. FIND MAILBOXES and SUBSCRIBE MAILBOX provide a mechanism for the user to identify that this is his or her real INBOX.

FIND ALL.MAILBOXES must, at least, return all the mailbox names that are returned by FIND MAILBOXES.

The exact meaning of this is implementation-dependent, since the concept of a bounded or deterministic set of "mailboxes available to the user" may not be meaningful for a particular server or server implementation.

tag FIND BBOARDS pattern

The FIND BBOARDS command accepts as an argument a pattern that specifies some set of bulletin board names that the user has declared as being "active" or "subscribed". Wildcards are permitted as in FIND MAILBOXES.

The FIND BBOARDS command will return some set of unsolicited BBOARD replies that have as their value a single bulletin board name.

```
EXAMPLE:  C: A002 FIND BBOARDS *
          S: * BBOARD FOOBAR
          S: * BBOARD GENERAL
          S: A002 OK FIND completed
```

The exact meaning of this is implementation-dependent, since the concept of a set of "active" or "subscribed" bboards that is preserved across sessions may not be meaningful for a particular server or server implementation. If the SUBSCRIBE BBOARD and UNSUBSCRIBE BBOARD commands are implemented then this command returns the list manipulated by those commands.

tag FIND ALL.BBOARDS pattern

The FIND ALL.BBOARDS command is similar to FIND BBOARDS; however, it should return a complete list of all bulletin

boards available to the user. Data are returned as in FIND BBOARDS.

FIND ALL.BBOARDS must, at least, return all the bboard names that are returned by FIND BBOARDS.

The exact meaning of this is implementation-dependent, since the concept of a bounded or deterministic set of "bboards available to the user" may not be meaningful for a particular server or server implementation.

Subscription commands

These commands permit the manipulation of mailbox or bulletin board subscriptions. Subscription status should be preserved between sessions.

tag SUBSCRIBE MAILBOX mailbox

The SUBSCRIBE MAILBOX command adds the specified mailbox name to the list of "active" or "subscribed" mailboxes as returned by the FIND MAILBOXES command. This command returns an OK response only if the subscription is successful.

tag UNSUBSCRIBE MAILBOX mailbox

The UNSUBSCRIBE MAILBOX command removes the specified mailbox name from the list of "active" or "subscribed" mailboxes as returned by the FIND MAILBOXES command. This command returns an OK response only if the unsubscription is successful.

tag SUBSCRIBE BBOARD bboard

The SUBSCRIBE BBOARD command adds the specified mailbox name to the list of "active" or "subscribed" bulletin boards as returned by the FIND BBOARDS command. This command returns an OK response only if the subscription is successful.

tag UNSUBSCRIBE BBOARD bboard

The UNSUBSCRIBE BBOARD command removes the specified mailbox name from the list of "active" or "subscribed" bulletin boards as returned by the FIND BBOARDS command. This command returns

an OK response only if the unsubscription is successful.

tag SELECT mailbox

This command selects a particular mailbox. The server must check that the user is permitted read access to this mailbox. Before returning an OK to the client, the server must send the following unsolicited data to the client:

FLAGS	mailbox's defined flags
<n> EXISTS	the number of messages in the mailbox
<n> RECENT	the number of messages added to the mailbox since the previous time this mailbox was read

to define the initial state of the mailbox at the client. If it can not be determined which messages were added since the previous time a mailbox was read, then all messages SHOULD be considered recent. An example of this is if no "last read" time information is available or a read-only mailbox that does not permit a change of "last read" time.

Multiple selection commands are permitted in a session. The previous mailbox is automatically deselected when a new selection is made. If concurrent access to multiple mailboxes is desired, the client should open additional sessions as needed.

The mailbox name INBOX is a special name reserved to mean "the primary mailbox for this user on this server". The format of other mailbox names is implementation dependent.

The text of an OK response to the SELECT command should begin with either "[READ-ONLY]" or "[READ-WRITE]" to show the mailbox's access status.

tag EXAMINE mailbox

The EXAMINE command is similar to SELECT, and returns the same output; however, the selected mailbox is identified as read-only and no changes are permitted to this mailbox. EXAMINE has the same mailbox namespace as SELECT.

tag BBOARD mailbox

The BBOARD command is similar to SELECT, and returns the same output. Its argument is a shared mailbox (bulletin board) name instead of an ordinary mailbox. There is no requirement that the namespace for BBOARD be the same as that for SELECT and EXAMINE.

BBOARD also differs from EXAMINE in that it may allow changes (e.g. marking a message as seen or deleted) to a mailbox; the exact handling of this is implementation dependent.

tag CHECK

The CHECK command requests a checkpoint of the mailbox. CHECK may cause an operation that may take a non-instantaneous amount of real-time to complete. The exact meaning of a checkpoint is implementation-dependent. Possible interpretations include forcing an update of the server's disk of all changes made to the selected mailbox, rescanning of the entire mailbox, etc. If an implementation has no such considerations, CHECK should be equivalent to NOOP.

CHECK should NOT be used to poll for new mail; new mail checking happens implicitly as part of every command. NOOP should be used for any new mail polling. CHECK should NOT be used to get the current size of the mailbox; there is no guarantee that CHECK will cause an EXISTS or RECENT unsolicited response.

tag EXPUNGE

The EXPUNGE command permanently removes all messages with the \Deleted flag set from the currently selected mailbox. Before returning an OK to the client, for each message that is removed, an unsolicited EXPUNGE response is sent. The message number for each successive message in the mailbox is immediately decremented by 1; this means that if the last 5 messages in a 9-message mail file are expunged the client will receive 5 unsolicited EXPUNGE responses for message 5.

tag COPY sequence mailbox

The COPY command copies the specified message(s) to the specified destination mailbox. The flags of the message(s) SHOULD be preserved in the copy.

If the destination mailbox does not exist, a server SHOULD return an error. It SHOULD NOT automatically create the mailbox. Unless it is certain that the destination mailbox can not be created, the server MUST send the special information token "[TRYCREATE]" as the prefix of the text of the tagged NO response. This gives a hint to the client that it can attempt a CREATE command and retry the COPY if the CREATE is successful.

If the COPY command is unsuccessful for any reason, IMAP2bis server implementations MUST restore the destination mailbox its prior state before the COPY attempt.

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

tag APPEND mailbox 0#flag literal

The APPEND command appends the literal argument as a new message in the specified destination mailbox. This argument is in the format of an RFC 822 message. If any flags are specified, those flags SHOULD be set in the resulting message. If the append is unsuccessful for any reason the mailbox must be restored to its prior state before the APPEND attempt; no partial appending is permitted. If the mailbox is currently selected, the normal new mail actions should occur.

Server implementations SHOULD return a NO response if the length of the literal is zero.

If the destination mailbox does not exist, a server MUST return an error, and MUST NOT automatically create the mailbox. Unless it is certain that the destination mailbox can not be created, the server MUST send the special information token "[TRYCREATE]" as the prefix of the text of the tagged NO response. This gives a hint to the client that it can attempt a CREATE command and retry the APPEND if the CREATE is successful.

Note that this functionality is unsuitable for message delivery, because it does not provide a mechanism to transfer RFC 821 (SMTP) envelope information.

tag FETCH sequence data

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

ALL	Macro equivalent to: (FLAGS INTERNALDATE RFC822.SIZE ENVELOPE)
BODY	Non-extensible form of BODYSTRUCTURE.

BODY[section] The text of a particular body section. The section specification is a set of one or more part numbers delimited by periods.

Single-part messages only have a part 1.

Multipart messages are assigned consecutive part numbers, as they occur in the message. If a particular part is of type message or multipart, its parts must be indicated by a period followed by the part number within that nested multipart part. It is not permitted to fetch a multipart part itself, only its individual members.

A part of type MESSAGE and subtype RFC822 also has nested parts. These are the parts of the MESSAGE part's body. Nested part 0 of a part of type MESSAGE and subtype RFC822 is the RFC 822 header of the message.

Every message has at least one part.

EXAMPLE: Here is a complex message with its associated section specifications.

```
1   TEXT/PLAIN
2   APPLICATION/OCTET-STREAM
3   MESSAGE/RFC822
3.0   (RFC 822 header of the message)
3.1   TEXT/PLAIN
3.2   APPLICATION/OCTET-STREAM
    MULTIPART/MIXED
4.1   IMAGE/GIF
4.2   MESSAGE/RFC822
4.2.0   (RFC 822 header of the message)
4.2.1   TEXT/PLAIN
          MULTIPART/ALTERNATIVE
4.2.2.1  TEXT/PLAIN
4.2.2.2  TEXT/RICHTEXT
```

Note that there is no section specification for the Multi-part parts (no section 4 or 4.2.2).

The \Seen flag is implicitly set; if this causes the flags to change they should be included as part of the fetch results.

BODYSTRUCTURE The MIME body structure of the message. This is computed by the server by parsing the MIME header lines.

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

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

FLAGS — The flags that are set for this message.

FULL — Macro equivalent to:
(FLAGS INTERNALDATE RFC822.SIZE ENVELOPE BODY)

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

RFC822 — The message in RFC 822 format. The \Seen flag is implicitly set; if this causes the flags to change they should be included as part of the fetch results. This is the concatenation of RFC822.HEADER and RFC822.TEXT.

RFC822.HEADER — The RFC 822 format header of the message as stored on the server including the delimiting blank line between the header and the body.

RFC822.HEADER.LINES header_line_list
All header lines (including continuation lines) of the RFC 822 format header of the message with a field-name (as defined in RFC 822) that matches any of the strings in header_line_list. The matching is case-independent but otherwise exact.

RFC822.HEADER.LINES.NOT header_line_list
All header lines (including continuation lines) of the RFC 822 format header of the message with a field-name (as defined in RFC 822) that does not match any of the strings in header_line_list. The matching is case-independent but otherwise exact.

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

RFC822.TEXT — The text body of the message, omitting the RFC 822 header. The \Seen flag is implicitly set; if this causes the flags to change they

should be included as part of the fetch results.

UID The unique identifier for the message.

EXAMPLES:

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

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

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

tag PARTIAL msgno data start_octet octet_count

The PARTIAL command is equivalent to the associated FETCH command, with the added functionality that only the specified number of octets, beginning at the specified starting octet, are returned. Note that only a single message can be fetched at a time. The first octet of a message, and hence the minimum for the starting octet, is octet 1.

The following FETCH items are valid data for PARTIAL: RFC822, RFC822.HEADER, RFC822.TEXT, and BODY[section].

Any partial fetch that attempts to read beyond the end of the text is truncated as appropriate. If the starting octet is beyond the end of the text, an empty string is returned.

The data are returned with the FETCH response. There is no indication of the range of the partial data in this response; thus it is generally not possible to implement caching with PARTIAL data. It is also not possible to stream multiple PARTIAL commands of the same data item without processing and synchronizing at each step, since each PARTIAL fetch of data replaces any prior (PARTIAL) FETCH of the data.

Note that when partial fetching it is possible to break in the middle of a line or a critical sequence such as a BASE64 quadruple or QUOTED-PRINTABLE shift. Implementations using partial fetching should keep this in mind. There is no requirement that partial fetches follow any sequence; so if it turns out that a partial fetch of octets 1 through 10000 breaks in an awkward place, it is permitted to continue with a partial fetch of 9987 through 19987,

etc.

The handling of the \Seen flag is the same as with the FETCH command.

tag STORE sequence data value

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

FLAGS	Replace the flags for the message with the argument (in flag list format).
+FLAGS	Add the flags in the argument to the message's flag list.
-FLAGS	Remove the flags in the argument from the message's flag list.

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

UID commands

These commands use unique identifiers instead of message numbers in their arguments to reference a particular message or range of messages. The unique identifier of a message is guaranteed not to refer to any other message in the mailbox. Unlike IMAP2bis sequence numbers, unique identifiers persist across sessions.

Sequence ranges are permitted; note however that there is no guarantee that unique identifiers be contiguous. A non-existent unique identifier within a sequence range is ignored without any error message generated.

Because of the potential for ambiguity, the UID command does not change responses. That is, the number after the "*" in an unsolicited FETCH response is a message number, not a unique identifier. However, servers MUST implicitly include UID as part of any FETCH response caused by a UID command, regardless of whether UID was specified.

```
EXAMPLE:  C: A003 UID FETCH 4827313:4828442 FLAGS
          S: * 23 FETCH (FLAGS (\Seen) UID 4827313)
          S: * 24 FETCH (FLAGS (\Seen) UID 4827943)
          S: * 25 FETCH (FLAGS (\Seen) UID 4828442)
          S: A003 UID FETCH completed
```

tag UID AFTER uniqueid

The UID AFTER command determines what unique identifiers exist that are greater than the specified unique identifier. It returns unsolicited FETCH responses for each such message.

For example, if the specified unique identifier refers to message 572 in a mailbox with 613 messages, the results returned are equivalent to doing "FETCH 573:613 UID".

tag UID COPY sequence mailbox

The UID COPY command is identical to the COPY command, with the exception that the numbers used in the sequence are unique identifiers instead of message numbers.

tag UID FETCH sequence data

The UID FETCH command is identical to the FETCH command, with the exception that the numbers used in the sequence are unique identifiers instead of message numbers.

tag UID STORE sequence data value

The UID STORE command is identical to the STORE command, with the exception that the numbers used in the sequence are unique identifiers instead of message numbers.

tag SEARCH search_criteria

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

```
        A003 SEARCH DELETED FROM "SMITH" SINCE 1-OCT-87
```

returns the message numbers for all deleted messages from Smith

that were placed in the mail file since October 1, 1987.

In all search criteria that use strings, a message matches the criteria if the string is a substring of the field. The matching is case-independent except as noted below.

The server may interpret an RFC 1522 format string to express text in a character set other than US-ASCII. The criteria matches if the RFC 1522 interpreted string matches an interpreted substring (MIME or RFC 1522 as appropriate) of the field.

A server implementation may omit case-independent matching on RFC 1522 strings.

The currently defined search criteria are:

ALL	All messages in the mailbox; the default initial criterion for ANDing.
ANSWERED	Messages with the \Answered flag set.
BCC istring	Messages that contain the specified string in the envelope structure's BCC field.
BEFORE date	Messages whose internal date is earlier than the specified date.
BODY istring	Messages that contain the specified string in the body of the message.
CC istring	Messages that contain the specified string in the envelope structure's CC field.
DELETED	Messages with the \Deleted flag set.
FLAGGED	Messages with the \Flagged flag set.
FROM istring	Messages that contain the specified string in the envelope structure's FROM field.
KEYWORD flag	Messages with the specified flag set.
NEW	Messages that have the \Recent flag set but not the \Seen flag. This is functionally equivalent to "RECENT UNSEEN".
OLD	Messages that do not have the \Recent flag set.

ON date — Messages whose internal date is within the specified date.

RECENT — Messages that have the \Recent flag set.

SEEN — Messages that have the \Seen flag set.

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

SUBJECT istring — Messages that contain the specified string in the envelope structure's SUBJECT field.

TEXT istring — Messages that contain the specified string.

TO istring — Messages that contain the specified string in the envelope structure's TO field.

UIDAFTER uniqueid — Messages that have a UID greater than the specified UID.

UIDBEFORE uniqueid — Messages that have a UID less than the specified UID.

UNANSWERED — Messages that do not have the \Answered flag set.

UNDELETED — Messages that do not have the \Deleted flag set.

UNFLAGGED — Messages that do not have the \Flagged flag set.

UNKEYWORD flag — Messages that do not have the specified flag set.

UNSEEN — Messages that do not have the \Seen flag set.

Responses

The first group of responses complete a request, and indicate whether the command was successful. The response text is a line of human readable text, optionally prefixed by an atom inside square brackets that conveys a special information token between cooperating servers and clients.

The currently defined special information tokens are:

PARSE	An error occurred in parsing the RFC 822 or MIME headers of a message in the mailbox.
READ-ONLY	The mailbox is open read-only, or its access while open has changed from read-write to read-only.
READ-WRITE	The mailbox is open read-write, or its access while open has changed from read-only to read-write.
TRYCREATE	An APPEND or COPY attempt failed because the target mailbox does not exist. The server sends this as a hint to the client that the operation would probably succeed if the mailbox is first created by means of the CREATE command.
UNSEEN	Followed by a decimal number, indicates the number of the first unread message. This is intended to be used with certain bboard formats to assist the user in finding the first unread message in those cases where "unread" and "recent" are separate concepts.
hex string	A hexadecimal string is returned as a special information token to represent a Kerberos return authenticator. This only occurs in response to a LOGIN command that uses Kerberos authentication.

Other special information tokens defined by particular client or server implementations should be prefixed with an "X" until they are added to a revision of this protocol.

tag OK resp_text

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

tag NO resp_text

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

tag BAD resp_text

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

The second group of responses convey human-readable information. The response text is a line of human readable text, optionally prefixed by an atom inside square brackets that conveys special information between cooperating servers and clients.

* PREAUTH resp_text

This response is one of three possible greetings at session startup. It indicates that the session has already been authenticated by external means and thus no LOGIN command is needed.

* OK resp_text

This response identifies an information message from the server. It does not indicate completion of any particular request, nor is it necessarily related to any request. The text is a line of human-readable text that should be presented to the user as an information message.

This response is also one of three possible greetings at session startup. It indicates that the session is not yet authenticated and that a LOGIN command is needed.

* NO resp_text

This response identifies a warning message from the server. It does not indicate completion of any request, nor is it necessarily related to any request. The text is a line of human-readable text

that should be presented to the user as a warning of improper operation.

* BAD resp_text

This response identifies a serious error message from the server. It does not indicate completion of any request, nor is it necessarily related to any request. It may also indicate a faulty command from the client in which a tag could not be parsed. The text is a line of human-readable text that should be presented to the user as a serious or possibly fatal error.

* BYE text

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

This response is also one of three possible greetings at session startup. It indicates that the server is not willing to accept a session from this client.

The third group of responses convey data about the mailbox or messages inside the mailbox. This is how message data are transmitted from the server to the client.

* MAILBOX mstring

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

* BBOARD mstring

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

* SEARCH number(s)

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

* FLAGS flag_list

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

* number message_data

This response occurs as a result of any command when a mailbox is selected. The message_data is one of the following:

EXISTS The number of messages in the mailbox.

RECENT The number of messages that have arrived since the previous time this mailbox was read.

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

An unsolicited EXPUNGE response MUST NOT be sent except while responding to a request other than FETCH, STORE, or SEARCH. All references to message numbers sent after an unsolicited EXPUNGE response are adjusted to reflect the effect of the expunge.

Discussion: a potential ambiguity exists with the FETCH, STORE, and SEARCH requests if the server is permitted to send unsolicited EXPUNGE responses. This is because these requests can be streamed. If two successive FETCH requests are streamed, and if during the time of the processing of the first request there is an expunge response, then the sequence of the second request is no longer valid.

FETCH data

This is the principal means that data about a message are returned to the client. The data are in an S-expression form, and consists of a sequence of pairs of data item name and their values. The current data items are:

BODY Similar to BODYSTRUCTURE, but without the extension data.

BODY[section] A string expressing the body contents of the specified section. The string should be interpreted by the client according to the content transfer encoding, body type, and subtype.

Note that non-textual data are transfer encoded; therefore, the string is likely to be 7-bit US-ASCII. This is NOT necessarily the byte size or character set of the interpreted result.

BODYSTRUCTURE An S-expression format list that describes the body structure of a message. This is computed by the server by parsing the RFC 822 header and body into the component parts, defaulting various fields as necessary.

Multiple parts are indicated by S-expression nesting. Instead of a body type as the first element of the list there is a nested body. The second element of the list is the multipart subtype (mixed, digest, parallel, alternative, etc.).

Extension data follows the multipart subtype. Extension data is never returned with the older BODY fetch, but may be returned with a BODYSTRUCTURE fetch. Extension data, if present, must be in the defined order.

No multipart extension data is currently defined.

Any subsequent data is extension data, not yet defined in this version of the protocol. Such extension data consist of zero or more NILs, strings, numbers, and/or potentially nested lists of such data. Clients which do a BODYSTRUCTURE fetch MUST be prepared to accept such extension data. Servers MUST NOT send such extension data until it has been defined by a future version of the protocol.

The basic fields of a non-multipart body part are in the following order:

- body type - a string giving the content type name as defined in MIME
- body subtype - a string giving the content subtype name as defined in MIME
- body parameter list - an S-expression list of attribute/value pairs [e.g. (foo bar baz rag) where "bar" is the value of "foo" and "rag" is the value of "baz"] as defined in MIME.
- body id - a string giving the content id as defined in MIME.
- body description - a string giving the content description as defined in MIME.
- body encoding - a string giving the content transfer encoding as defined in MIME.
- body size - a number giving the size of the body in octets. Note that this size is the size in its transfer encoding and not the resulting size after any decoding.

A body type of type MESSAGE and subtype RFC822 contains, immediately after the basic fields, the envelope structure, body structure, and size in text lines of the encapsulated message.

A body type of type TEXT contains, immediately after the basic fields, the size of the body in text lines. Note that this size is the size in its transfer encoding and not the resulting size after any decoding.

Extension data follows the basic fields and the type-specific fields listed above. Extension data is never returned with the older BODY fetch, but may be returned with a BODYSTRUCTURE fetch. Extension data, if present, must be in the defined order.

The extension data of a non-multipart body part are in the following order:

- body MD5 - a string giving the content MD5 value as defined in MIME

Any subsequent extension data are not yet defined in this version of the protocol, and would be in the form described above under multipart extension data.

ENVELOPE An S-expression format list that describes the envelope structure of a message. This 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 structure 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 address structures.

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

RFC 822 group syntax is indicated by a special form of address structure in which the host name file is NIL. If the mailbox name field is also NIL, this is an end of group marker (semi-colon in RFC 822 syntax). If the mailbox name field is non-NIL, this is a start of group marker, and the mailbox name field holds the group name phrase.

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

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

\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

as well as the following special flag:

\Recent Message arrived since the previous time this mailbox was read

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

RFC822 A string expressing the message in RFC 822 format.

RFC822.HEADER A string expressing the RFC 822 format header of the message, including the delimiting blank line between the header and the body. This is used for the FETCH data items RFC822.HEADER, RFC822.HEADER.LINES, and RFC822.HEADER.LINES.NOT (note that a blank line is always included regardless of header line restrictions).

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.

UID A number expressing the unique identifier of the message.

COPY Obsolete. New server implementations MUST NOT transmit this response. Client implementations SHOULD ignore this response (not report an error).

STORE data
Obsolete and functionally equivalent to FETCH. New server implementations MUST NOT transmit this response. Client implementations SHOULD treat this response as equivalent to the FETCH response.

The final group of responses contains a single, special purpose response.

+ resp_text

This response identifies that the server is ready to accept the text of a literal from the client. The text of this response is a line of human-readable text of the server's choosing (it is generally never seen by a client's human user).

The purpose of this command is to solve a synchronization problem that can occur if a string in a command is a literal. This may occur when logging in (if the password contains "funny" characters), and always occurs when using the APPEND command, since a message consists of multiple lines.

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

No such synchronization protection is provided for literals sent from the server to the client. Any synchronization problems in this direction would be caused by a bug in the client or server.

Sample IMAP2bis session

The following is a transcript of an IMAP2bis session. A long line in this sample is broken for editorial clarity.

```
S:   * OK IMAP2bis Service 7.2(62) at Thu, 29 Jul 1993 21:34:23 -0700 (PDT)
C:   a001 login mrc secret
S:   a001 OK LOGIN completed
C:   a002 select inbox
S:   * 18 EXISTS
S:   * FLAGS (\Answered \Flagged \Deleted \Seen)
S:   * 0 RECENT
S:   a002 OK [READ-WRITE] SELECT completed
S:   a003 fetch 12 full
S:   * 12 FETCH (FLAGS (\Seen) INTERNALDATE "14-Jul-1993 02:44:25 -0700"
      RFC822.SIZE 4282 ENVELOPE ("Wed, 14 Jul 1993 02:23:25 -0700 (PDT)"
      "IMAP2bis WG mtg summary and minutes" (("Terry Gray" NIL "gray"
      "cac.washington.edu")) ((NIL NIL "owner-imap" "cac.washington.edu"))
      (("Terry Gray" NIL "gray" "cac.washington.edu")) ((NIL NIL "imap"
      "cac.washington.edu")) ((NIL NIL "minutes" "CNRI.Reston.VA.US")
      ("John C Klensin" NIL "KLENSIN" "INFOODS.MIT.EDU")("Erik Huizer"
      NIL "Erik.Huizer" "SURFnet.nl")) NIL NIL
      "<Pine.3.84.9307140123.B27397-0100000@shiva2.cac.washington.edu>")
       BODY ("TEXT" "PLAIN" ("CHARSET" "US-ASCII") NIL NIL "7BIT" 3028 92))
S:   a003 OK FETCH completed
C:   a004 fetch 12 rfc822.header
S:   * 12 FETCH (RFC822.HEADER {485}
S:   Date: Wed, 14 Jul 1993 02:23:25 -0700 (PDT)
S:   From: Terry Gray <gray@cac.washington.edu>
S:   Reply-To: Terry Gray <gray@cac.washington.edu>
S:   Subject: IMAP2bis WG mtg summary and minutes
S:   To: imap@cac.washington.edu
S:   Cc: minutes@CNRI.Reston.VA.US,
S:           John C Klensin <KLENSIN@INFOODS.MIT.EDU>,
S:           Erik Huizer <Erik.Huizer@SURFnet.nl>
S:   Message-Id:
S:    <Pine.3.84.9307140123.B27397-0100000@shiva2.cac.washington.edu>
S:   Mime-Version: 1.0
S:   Content-Type: TEXT/PLAIN; CHARSET=US-ASCII
S:
S:   )
S:   a004 OK FETCH completed
C:   a005 store 12 +flags \deleted
S:   * 12 FETCH (FLAGS (\Seen \Deleted))
S:   a005 OK +FLAGS completed
C:   a006 logout
S:   * BYE IMAP2bis server terminating connection
S:   a006 OK LOGOUT completed
```

Design Discussion

IMAP2bis is a textual protocol. The use of MIME encoding in IMAP2bis makes it possible to support 8-bit textual and binary mail.

IMAP2bis implementations MAY transmit 8-bit or multi-octet characters in literals, but should do so only when the character set is identified. For example, 8-bit characters are specifically permitted in MIME body parts (fetching BODY[section]) of type TEXT. 8-bit characters are also permitted in the argument to APPEND.

Servers MUST NOT transmit 8-bit characters in RFC822.HEADER fetch results. Servers MUST NOT transmit 8-bit characters in RFC822.TEXT (and by extension RFC822) fetch results, unless there are MIME data in the message that identify the character sets of all 8-bit data.

Because 8-bit characters are permitted in the argument to APPEND, a server that is unable to preserve 8-bit data properly MUST be able to reversibly convert 8-bit APPEND data to 7-bit using MIME.

Although a BINARY body encoding is defined, IMAP2bis does not permit unencoded binary strings. A "binary string" is any string with NUL characters; a string with an excessive amount of CTL characters may also be considered to be binary. The mixing of unencoded binary in the same stream as textual commands would make the protocol more vulnerable to synchronization problems. Implementations MUST encode binary data into BASE64 before transmitting it with IMAP2bis.

When operating in the online model, an IMAP2bis client should maintain a local cache of data from the mailbox. This cache is an incomplete model of the mailbox, and at startup is generally empty. As the client processes all unsolicited data, it updates the cache based on this data. When a tagged response arrives, the client's cache has been updated from the associated request.

Note that a server can send data that the client did not request, such as mailbox size or flag updates. A server MUST send mailbox size updates automatically while processing a command. A server SHOULD send message flag updates automatically, without requiring the client to request such updates explicitly.

Regardless of what implementation decisions a client may take on caching, a client MUST record EXISTS and RECENT updates and MUST NOT assume that a CHECK or NOOP command will return EXISTS or RECENT information.

Although it is permitted for a server to send an unsolicited response while there is no command in progress, this practice SHOULD NOT be

followed because of flow control considerations. It can cause an incautious implementation to deadlock. A deadlock is avoided if either of the following conditions are true: (1) except for the greeting, the server never sends responses while there is no command in progress; (2) the server process is capable of reading commands while sending data. The latter condition generally requires either a multi-threading server implementation or use of a polling facility and non-blocking I/O.

If a server has an inactivity autologout timer, that timer MUST be of at least 30 minutes' duration. The receipt of a NOOP command from the client during that interval should suffice to reset the autologout timer. Periodic transmission of a NOOP from the client during periods of inactivity also has the benefit of avoiding the possible deadlock noted above.

It is frequently asked why there is no message posting function in IMAP2bis. Message posting is orthogonal to the scope of a mail access protocol and detracts from its primary focus. SMTP (RFC 821) provides the minimal functionality needed for message posting without losing valuable capabilities (such as blind carbon copies). Any message posting function in IMAP2bis would need, at a minimum, to provide equivalent functionality.

At the time of the writing of this document, an extensive set of extensions to SMTP is in the Internet standards process. Should those extensions become an Internet Standard it would be necessary to revise IMAP2bis again to provide corresponding capabilities, were a message posting facility to be included in IMAP2bis. In other words, a duplication of effort would be required each time a change is made to message transport technology.

Another undesirable aspect of message posting in IMAP2bis occurs when a remote server is used. It is unlikely that a client would support multiple means of posting a message. It adds excessive size and complexity that can not be afforded, particularly on smaller machines. It also can lead to poor performance. Consider a client connecting to an IMAP2bis server over an interactive satellite link to a foreign country. A local message posting (SMTP) server is available that uses a lower-cost batched link. Here, it would be wasteful to use the interactive link for posting.

Message posting to IMAP2bis has been suggested as a means of authenticating postings. The problem is that access authentication credentials are not necessarily the same as posting authentication credentials. At some sites, the disclosure of a portion of access authentication credentials in a mail message (as a "From" or "Sender" address) may be a serious security breach of greater significance

than forged mail.

The Internet message transport infrastructure has no concept of authentication credentials, and neither authentication syntax nor semantics are transferred within a message. As a result, any attempt at authenticating a message via posting authentication is completely ineffective once the message leaves the authenticating server; any indication of authentication in the message can easily be reproduced further down the line. Public-key based message authentication systems such as Privacy Enhanced Mail are now under development to address this problem.

IMAP2bis does not address problems with multiple IMAP2bis servers at a single site, access control lists, and mobility of client configuration and address book information. These and other issues are being considered for a companion protocol.

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 (SPACE) and not a comma.

Except as noted otherwise, all alphabetic characters in the IMAP2bis protocol are case-insensitive. For example, "LOGIN", "login" and "lOgIn" all refer to the same command, and \FLAGGED, \Flagged, and \FlAgGeD all refer to the same flag. The use of upper or lower case characters to define token strings is for editorial clarity only, although they may be construed as defining a suggested usage. Implementations MUST accept these strings in a case-insensitive fashion.

Syntax marked as obsolete may be encountered with implementations written for an older version of this specification. New implementations SHOULD accept obsolete syntax as input, but MUST NOT otherwise use it.

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

addr_adl        ::= nstring

addr_host       ::= nstring
                    ;; NIL indicates RFC 822 group syntax

addr_mailbox    ::= nstring
                    ;; NIL indicates end of RFC 822 group; if non-NIL
                    ;; and addr_host is NIL, holds RFC 822 group name

addr_name       ::= nstring

append          ::= "APPEND" SPACE mailbox [SPACE 1#flag] SPACE literal

astring         ::= atom / string

atom            ::= 1*<any CHAR except specials, SPACE, and CTLs>

bboard          ::= "BBOARD" SPACE mailbox_bboard

body            ::= "(" body_structure ")"

body2           ::= "(" body2_structure ")"
```

```
   body2_extension ::= nstring / number / "(" 1#body2_extension ")"
                       ;; Future expansion.  Clients MUST accept body2
                       ;; extension fields.  Servers MUST NOT generate
                       ;; body2 extension fields.

   body2_md5       ::= nstring
                       ;; reserved for MD5 checksum

   body2_multipart ::= 1*body2 SPACE body_subtype [SPACE 1#body2_extension]

   body2_structure ::= body2_terminal / body2_multipart

   body2_terminal  ::= body_terminal SPACE body2_md5 [SPACE 1#body2_extension]

   body_basic      ::= body_type_basic SPACE body_subtype SPACE body_fields

   body_fields     ::= body_parameter SPACE body_id SPACE body_description
                       SPACE body_encoding SPACE body_size

   body_description
                   ::= nstring

   body_encoding   ::= <"> body_enc_def <"> / body_enc_other

   body_enc_def    ::= "7BIT" / "8BIT" / "BINARY" / "BASE64"/
                       "QUOTED-PRINTABLE"

   body_enc_other  ::= string

   body_id         ::= nstring

   body_msg        ::= body_msg_822 / body_msg_other

   body_msg_822    ::= body_type_msg SPACE body_subtyp_822 SPACE body_fields
                       SPACE envelope SPACE body SPACE body_size_lines

   body_msg_other  ::= body_type_msg SPACE body_subtype SPACE body_fields
                       ;; subtype MUST NOT be "RFC822"

   body_multipart  ::= 1*body SPACE body_subtype

   body_parameter  ::= nil / "(" 1#(string string) ")"

   body_section    ::= number / number "." body_section

   body_size       ::= number
                       ;; size in octets
```

```
   body_size_lines ::= number

   body_structure  ::= body_terminal / body_multipart

   body_subtype    ::= string

   body_subtyp_822 ::= <"> "RFC822" <">

   body_terminal   ::= body_basic / body_msg / body_text

   body_text       ::= body_type_text SPACE body_subtype SPACE body_fields
                       SPACE body_size_lines

   body_type_basic ::= <"> ("APPLICATION" / "AUDIO" / "IMAGE" / "VIDEO") <"> /
                       string

   body_type_msg   ::= <"> "MESSAGE" <">

   body_type_text  ::= <"> "TEXT" <">

   CHAR            ::= <any 7-bit US-ASCII character except NUL, 0x01 - 0x7f>

   CHAR8           ::= <any 8-bit octet except NUL, 0x01 - 0xff>

   check           ::= "CHECK"

   copy            ::= "COPY" SPACE sequence SPACE mailbox

   CR              ::= <ASCII CR, carriage return, 0x0C>

   create          ::= create_real / create_check

   create_check    ::= "CREATE" SPACE "INBOX"
                       ;; returns a NO response (not BAD)

   create_real     ::= "CREATE" SPACE mailbox_other

   CRLF            ::= CR LF

   CTL             ::= <any ASCII control character and DEL, 0x00 - 0x1f,
0x7f>

   date            ::= date_text / <"> date_text <">

   date_day        ::= 1*2DIGIT
                       ;; day of month

   date_day_fixed  ::= (SPACE 1DIGIT) / 2DIGIT
                       ;; fixed-format version of date_day
```

```
   date_month      ::= "Jan" / "Feb" / "Mar" / "Apr" / "May" / "Jun" /
                       "Jul" / "Aug" / "Sep" / "Oct" / "Nov" / "Dec"

   date_text       ::= date_day "-" date_month "-" (date_year / date_year_old)

   date_year       ::= 4DIGIT

   date_year_old   ::= 2DIGIT
                       ;; Obsolete, (year - 1900)

   date_time       ::= <"> (date_time_new / date_time_old) <">

   date_time_new   ::= date_day_fixed "-" date_month "-" date_year SPACE
                       time SPACE zone

   date_time_old   ::= date_day_fixed "-" date_month "-" date_year_old SPACE
                       time "-" zone_old
                       ;; Obsolete

   delete          ::= "DELETE" SPACE mailbox_other

   DIGIT           ::= <any ASCII decimal digit, 0x30 - 0x39>

   DIGIT_HEX       :: DIGIT / "a" / "b" / "c" / "d" / "e" / "f"

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

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

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

   env_date        ::= nstring

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

   env_in-reply-to ::= nstring

   env_message-id  ::= nstring

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

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

   env_subject     ::= nstring
```

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

   examine         ::= "EXAMINE" SPACE mailbox

   expunge         ::= "EXPUNGE"

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

   fetch_att       ::= fetch_att_lines / fetch_att_other / fetch_att_text

   fetch_att_lines ::= "RFC822.HEADER.LINES" SPACE header_line_list /
                       "RFC822.HEADER.LINES.NOT" SPACE header_line_list /

   fetch_att_other ::= "BODY" / "BODYSTRUCTURE" / "ENVELOPE" / "FLAGS" /
                       "INTERNALDATE" / "RFC822.SIZE" / "UID"

   fetch_att_text  ::= "BODY[" body_section "]" / "RFC822" /
                       "RFC822.HEADER" / "RFC822.TEXT"

   find            ::= find_mailbox / find_bboard

   find_bboard     ::= find_bboards / find_boards_all

   find_bboards    ::= "FIND" SPACE "BBOARDS" SPACE find_pattern

   find_bboards_all
                   ::= "FIND" SPACE "ALL.BBOARDS" SPACE find_pattern

   find_mailbox    ::= find_mailboxes / find_mailboxes_all

   find_mailboxes  ::= "FIND" SPACE "MAILBOXES" SPACE find_pattern

   find_mailboxes_all
                   ::= "FIND" SPACE "ALL.MAILBOXES" SPACE find_pattern

   find_pattern    ::= astring
                       ;; includes find_wildcards

   find_wildcards  ::= "%" / "?" / "*"

   flag            ::= user_flag / system_flag

   flag_list       ::= "(" 1#flag ")"

   flags           ::= 1#flag / flag_list

   greeting        ::= "*" SPACE (resp_cond_auth / resp_cond_bye) CRLF
```

```
   header_line     ::= astring

   header_line_list
                   ::= "(" 1#header_line ")"

   inbox           ::= "INBOX"
                       ;; case-independent, but SHOULD be upper-case

   istring         ::= astring
                       ;; possible RFC 1522 format data

   kerberos_authenticator
                   ::= 1*DIGIT_HEX

   kerberos_response
                   ::= 1*DIGIT_HEX

   LF              ::= <ASCII LF, line feed, 0x0A>

   literal         ::= "{" number "}" CRLF 1*CHAR8
                       ;; The number represents the number of CHAR8 octets.

   login           ::= "LOGIN" SPACE userid SPACE password

   logout          ::= "LOGOUT"

   mailbox         ::= inbox / mailbox_other

   mailbox_bboard  ::= astring
                       ;; May not be INBOX (in any case).  Should not
                       ;; include find_wildcards.  May be case-dependent
                       ;; as a function of server implementation.  May
                       ;; be a different namespace from mailbox_other.



   mailbox_other   ::= astring
                       ;; May not be INBOX (in any case).  Should not
                       ;; include find_wildcards.  May be case-dependent
                       ;; as a function of server implementation

   mailbox_data    ::= "MAILBOX" SPACE mstring / "BBOARD" SPACE mstring /
                        "SEARCH" [SPACE 1#number] / "FLAGS" SPACE flag_list

   message_data    ::= number SPACE (msg_exists / msg_recent / msg_expunge /
                       msg_fetch / msg_obsolete)
```

```
   msg_copy        ::= "COPY"
                       ;; Obsolete

   msg_exists      ::= "EXISTS"

   msg_expunge     ::= "EXPUNGE"

   msg_fetch       ::= "FETCH" SPACE "(" 1#("BODY" SPACE body /
                       "BODYSTRUCTURE" SPACE body2 /
                       "BODY[" body_section "]" nstring /
                       "ENVELOPE" SPACE envelope /
                       "FLAGS" SPACE "(" 0#(recent_flag / flag) ")" /
                       "INTERNALDATE" SPACE date_time /
                       "RFC822" SPACE nstring /
                       "RFC822.HEADER" SPACE nstring /
                       "RFC822.SIZE" SPACE number /
                       "RFC822.TEXT" SPACE nstring /
                       "UID" SPACE uniqueid) ")"

   msg_obsolete    ::= msg_copy / msg_store
                       ;; Obsolete unsolicited data responses

   msg_recent      ::= "RECENT"

   msg_store       ::= "STORE" SPACE "(" 1#("FLAGS" SPACE
                       "(" 0#(recent_flag / flag) "))"
                       ;; Obsolete

   mstring         ::= text_line
                       ;; Represents a mailbox

   nil             ::= "NIL"

   noop            ::= "NOOP"

   nstring         ::= nil / string

   number          ::= 1*DIGIT

   partial         ::= "PARTIAL" SPACE number SPACE fetch_att_text SPACE
                       number SPACE number

   password        ::= astring / "@KERBEROS:" kerberos_authenticator

   QCHAR           ::= <any CHAR except qspecials, CR, and LF>

   qspecials       ::= <"> / "%" / "\"
```

```
   quoted_string   ::= <"> *QCHAR <">

   recent_flag     ::= "\Recent"

   ready           ::= "+" SPACE resp_text

   rename          ::= "RENAME" SPACE mailbox SPACE mailbox_other

   request         ::= tag SPACE (request_auth / request_authed /
request_open)
                       ;; modal based on state

   request_any     ::= noop / logout
                       ;; valid in all modes

   request_auth    ::= request_any / login
                       ;; valid only when in not authenticated mode

   request_authed  ::= request_any / create / delete / rename / find /
                       subscribe / unsubscribe / select / examine / bboard /
                       append / x_command
                       ;; valid only when in authenticated or mailbox open
mode

   request_open    ::= request_authed / check / expunge / copy / fetch /
                       partial / store / uid / search / x_command
                       ;; valid only when in mailbox open mode

   response        ::= *<response_data> response_done

   response_data   ::= "*" SPACE (resp_cond_state / resp_cond_bye /
                       mailbox_data / message_data) CRLF

   response_done   ::= response_tagged / response_fatal

   response_fatal  ::= "*" SPACE resp_cond_bye CRLF

   response_tagged ::= tag SPACE resp_cond_state CRLF

   resp_cond_auth  ::= ("OK" / "PREAUTH") SPACE resp_text
                       ;; authentication condition

   resp_cond_bye   ::= "BYE" SPACE resp_text
                       ;; server will disconnect condition

   resp_cond_state ::= ("OK" / "NO" / "BAD") SPACE resp_text
                       ;; status condition

   resp_text       ::= [resp_token SPACE] text_line
```

```
   resp_token      ::= "[" resp_token_type "]" [SPACE res_token_arg]

   resp_token_arg  ::= 1#number
                       ;; arguments depend upon token type

   resp_token_type ::= "PARSE" / "READ-ONLY" / "READ-WRITE" / "TRYCREATE" /
                       "UNSEEN" / "X" atom / kerberos_response

   search          ::= "SEARCH" SPACE 1#("ALL" / "ANSWERED" /
                       "BCC" SPACE istring / "BEFORE" SPACE date /
                       "BODY" SPACE istring / "CC" SPACE istring / "DELETED" /
                       "FLAGGED" / "FROM" space istring /
                       "KEYWORD" SPACE user_flag / "NEW" / "OLD" /
                       "ON" SPACE date / "RECENT" / "SEEN" /
                       "SINCE" SPACE date / "SUBJECT" SPACE istring /
                       "TEXT" SPACE istring / "TO" SPACE istring /
                       "UIDBEFORE" SPACE uniqueid / "UIDAFTER" SPACE uniqueid
/
                       "UNANSWERED" / "UNDELETED" / "UNFLAGGED" /
                       "UNKEYWORD" SPACE user_flag / "UNSEEN")

   select          ::= "SELECT" SPACE mailbox

   sequence        ::= number / (sequence "," sequence) / (number ":" number)
                       ;; identifies a set of messages by consecutive numbers
                       ;; from 1 to the number of messages in the mailbox.
                       ;; Comma delimits individual numbers, colon delimits
                       ;; between two numbers inclusive.
                       ;; Example: 2,4:7,9,12:15 is 2,4,5,6,7,9,12,13,14,15

   SPACE           ::= <ASCII SP, space, 0x20>

   specials        ::= "(" / ")" / "{" / qspecials

   store           ::= "STORE" SPACE sequence SPACE store_att

   store_att       ::= "+FLAGS" SPACE flags / "-FLAGS" SPACE flags /
                       "FLAGS" SPACE flags

   string          ::= quoted_string / literal

   subscribe       ::= subscribe_mailbox / subscribe_bboard

   subscribe_bboard
                   ::= "SUBSCRIBE" SPACE "BBOARD" SPACE mailbox_bboard

   subscribe_mailbox
                   ::= "SUBSCRIBE" SPACE "MAILBOX" SPACE mailbox
```

```
   system_flag     ::= "\Answered" / "\Flagged" / "\Deleted" / "\Seen" /
                       system_flag_new

   system_flag_new ::= "\" atom
                       ;; future expansion

   tag             ::= 1*<any CHAR except "*", "+", specials, SPACE, and CTLs>

   text_line       ::= 1*<any CHAR except CR and LF>

   time            ::= 2DIGIT ":" 2DIGIT ":" 2DIGIT
                       ;; hours minutes seconds

   uid             ::= "UID" SPACE (uid_after / copy / fetch / store)
                       ;; uniqueids used instead of message numbers

   uid_after       ::= "AFTER" SPACE uniqueid

   uniqueid        ::= number
                       ;;; strictly ascending

   unsubscribe     ::= unsubscribe_mailbox / unsubscribe_bboard

   unsubscribe_bboard
                   ::= "UNSUBSCRIBE" SPACE "BBOARD" SPACE mailbox_bboard

   unsubscribe_mailbox
                   ::= "UNSUBSCRIBE" SPACE "MAILBOX" SPACE mailbox_mailbox

   userid          ::= astring

   user_flag       ::= atom

   x_command       ::= "X" atom <optional arguments>
                       ;; experimental expansion commands

   zone            ::= ("+" / "-") 4DIGIT
                       ;; Signed four-digit value of hhmm representing
                       ;; hours and minutes west of Greenwich (that is,
                       ;; (the amount that the given time differs from
                       ;; Universal Time).  Subtracting the timezone
                       ;; from the given time will give the UT form.
                       ;; The Universal Time zone is "+0000".
```

```
   zone_old        ::= "UT" / "GMT" / "Z" /                ;; +0000
                       "AST" / "EST" / "CST" / "MST" /     ;; -0400 to -0700
                       "PST" / "YST" / "HST" / "BST" /     ;; -0800 to -1100
                       "ADT" / "EDT" / "CDT" / "MDT" /     ;; -0300 to -0600
                       "PDT" / "YDT" / "HDT" / "BDT" /     ;; -0700 to -1000
                       "A" / "B" / "C" / "D" / "E" / "F" / ;; +0100 to +0600
                       "G" / "H" / "I" / "K" / "L" / "M" / ;; +0700 to +1200
                       "N" / "O" / "P" / "Q" / "R" / "S" / ;; -0100 to -0600
                       "T" / "U" / "V" / "W" / "X" / "Y"   ;; -0700 to -1200
                       ;; Obsolete
```

A protocol session is as follows:

```
 Server: greeting
 *<Client: request (first part, if it contains a literal)
   *<Server: ready
     Client: request (next part)
    >
   Server: response
  >
```

Compatibility Notes

This is a summary of hints and recommendations to enable an IMAP2bis implementation, written to this specification, to interoperate with implementations that conform to earlier specifications. None of these hints and recommendations are required by this specification; implementors must decide for themselves whether they want their implementation to fail if it encounters old software.

IMAP2bis has been designed to be upwards compatible with earlier specifications. IMAP2bis facilities that were not in earlier specifications should be invisible to clients unless the client asks for the facility.

This information may not be complete; it reflects current knowledge of server and client implementations as well as "folklore" acquired in the evolution of the protocol.

IMAP2bis client interoperability with old servers

In general, a client should be able to discover whether a server supports a facility by trial-and-error; if an attempt to use a facility generates a BAD response, the client can assume that the server does not support the facility.

Some servers may disable certain commands as a matter of intentional site policy. For example, a bboard-only server may disable the SELECT command. Such servers should return a NO response to disabled commands instead of a BAD response.

A quick way to check whether a server implementation supports this specification is to try a UID FETCH 0 UID command. An OK or NO response would indicate a server that conforms to this specification; a BAD response would indicate an older server.

The CREATE, DELETE, and RENAME commands are new in IMAP2bis, and may not be present in old servers. A safe mechanism to test whether these commands are present is to try a CREATE INBOX command. If the response is NO, these commands are supported by the server. If the response is BAD, they are not. If the response is OK, the server's implementation is broken, since creating INBOX is not permitted.

The FIND MAILBOXES and FIND BBOARDS commands are new in RFC 1176. A BAD response to these commands indicates a server that does not support any form of FIND. It also indicates a server that does not support SUBSCRIBE and UNSUBSCRIBE. Note that the definition of the FIND MAILBOXES and FIND BBOARDS commands in

RFC 1176 differs from the definition in this specification; in RFC 1176 these commands were defined as returning a list of mailboxes or bulletin boards with no clear specification of whether the returned values were "subscribed" or "all possible" names.

The FIND ALL.MAILBOXES and FIND ALL.BBOARDS commands are new in IMAP2bis. A BAD response to these commands indicates a server that does not support a concept of subscriptions to a mailbox or bulletin board. The server may support FIND MAILBOXES and FIND BBOARDS using the older RFC 1176 semantics.

The SUBSCRIBE and UNSUBSCRIBE commands are new in IMAP2bis. A server that supports FIND ALL.MAILBOXES and FIND ALL.BBOARDS will also support the SUBSCRIBE and UNSUBSCRIBE commands.

The EXAMINE command is new in IMAP2bis. A BAD response to this command indicates a server that does not support an explicit read-only mode of access, and a SELECT command should be used instead.

Older server implementations may automatically create the destination mailbox on COPY if that mailbox does not already exist. This was how a new mailbox was created in older specifications. If the server does not support the CREATE command (see above for how to test for this), it will probably create a mailbox on COPY.

The APPEND command is new in IMAP2bis. A way to see if this command is implemented is to try to append a zero-length stream to a mailbox name that is known not to exist (or at least, highly unlikely to exist) on the remote system.

Although IMAP2bis clients SHOULD avoid asking for the same data more than once (by having a client-based cache of data returned by the server), this is not a requirement of the protocol. However, IMAP2bis clients MUST cache data from the EXISTS and RECENT unsolicited responses. Only the SELECT command is guaranteed to return EXISTS/RECENT information.

The BODY, BODY[section], and FULL fetch data items are new in IMAP2bis. A BAD response to an attempt to fetch either data item indicates a server that does not support server-based MIME parsing.

The BODYSTRUCTURE fetch data item is new in IMAP2bis. A BAD response to an attempt to fetch this data item indicates a server that does not support extensible results from server-

based MIME parsing. The server may be running an earlier, experimental version of IMAP2bis and support the older, non-extensible, BODY fetch data item. A client should attempt this data item before deciding that the server does not support MIME.

The use of nested part 0 of a part of type MESSAGE in a BODY or BODYSTRUCTURE fetch to get only the RFC 822 header of the message is new, and is not in earlier, experimental versions of IMAP2bis. A server that returns NIL is probably running the earlier version; with such servers the only way to obtain the RFC 822 header is to fetch the entire nested message.

The RFC822.HEADER.LINES and RFC822.HEADER.LINES.NOT fetch data items are new in IMAP2bis. A BAD response to an attempt to fetch this data item indicates a server that does not support selective header fetching. A client should use RFC822.HEADER and remove the unwanted information.

The UID fetch data item and the UID commands are new in IMAP2bis. A BAD response to an attempt to use these indicates a server that does not support unique identifiers.

The PARTIAL command is new in IMAP2bis. If this command causes a BAD response, then the client should use the appropriate FETCH command and ignore the unwanted data.

IMAP2bis client implementations must accept all responses and data formats documented in this specification, including those labeled as obsolete. This includes the COPY and STORE unsolicited responses and the old format of dates and times.

Older server implementations may not provide a way to set flags on APPEND. Client implementations which receive a BAD response to an APPEND command with flags should retry the command without flags.

Older server implementations may not preserve flags on COPY. Some server implementations may not permit the preservation of certain flags on COPY or their setting with APPEND as site policy. Older server implementations may attempt to preserve the internal date on COPY, and may cause a mailbox to be ordered in other than strictly ascending internal date/time order. Client implementations should not depend on any of these behaviors.

Older server implementations may send a TRYCREATE special information token inside a separate unsolicited OK response

instead of inside the NO response.

IMAP2bis server interoperability with old clients

In general, there should be no interoperation problem between a server conforming to this specification and a well-written client that conforms to an earlier specification. Known problems are noted below:

Clients written to use undocumented private server extensions that are not in any published specification may work poorly with server implementations that do not have those extensions.

Poor wording in the description of the CHECK command in earlier specifications implied that a CHECK command is the way to get the current number of messages in the mailbox. This is incorrect. A CHECK command does not necessarily result in an EXISTS response. Clients must remember the most recent EXISTS value sent from the server, and should not generate unnecessary CHECK commands.

An incompatibility exists with COPY in IMAP2bis. COPY in IMAP2bis servers does not automatically create the destination mailbox if that mailbox does not already exist. This may cause problems with old clients that expect automatic mailbox creation in COPY.

The PREAUTH unsolicited response is new in IMAP2bis. It is highly unlikely that an old client would ever see this response.

The COPY unsolicited response is obsolete. Old clients must not depend on receiving this response.

The STORE unsolicited response is obsolete. Old clients must not object to receiving a FETCH response instead of this response.

The format of dates and times has changed. Old clients should accept a four-digit year instead of a two-digit year, and a signed four-digit timezone value instead of a timezone name. In particular, client implementations must not treat a date/time as a fixed format string and assumed that the time begins at a particular octet.

Acknowledgements

Bill Yeager and Rich Acuff contributed invaluable suggestions in the evolution of IMAP2 from the original IMAP. James Rice pointed out several ambiguities in the previous IMAP2 specification.

My colleagues on the Pine team -- Steve Hubert, Laurence Lundblade, David Miller, and Mike Seibel -- worked long and hard to create a fantastic email user agent with worldwide popularity. Without their efforts, IMAP2 would have languished in obscurity. Terry Gray, our boss, provided much-needed moral support and guidance, while refusing to let us get away with "good enough" when "great" was possible.

John G. Myers and Chris Newman carefully examined the formal grammar and identified numerous mistakes and omissions in the drafts of this specification. They also provided invaluable input towards the overall architecture of the present protocol, and endured long meetings to reach the present protocol.

The present protocol would not have come into existence without the assistance of the rest of the IETF IMAP2 working group, in particular Ned Freed and Adam Treister.

Any mistakes, flaws, or sins of omission in this IMAP2bis protocol specification are, however, strictly my own; and the mention of any name above does not imply an endorsement.

Security Considerations

Security issues are discussed in this memo only as far as authentication to access a server are concerned.

Author's Address

Mark R. Crispin
Networks and Distributed Computing, JE-30
University of Washington
Seattle, WA 98195

Phone: (206) 543-5762

EMail: MRC@CAC.Washington.EDU