Network Working Group
Request for Comments: 1730
Category: Standards Track

M. Crispin
University of Washington

December 1994

INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4

Status of this Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Abstract

The Internet Message Access Protocol, Version 4 (IMAP4) allows a client to access and manipulate electronic mail messages on a server. IMAP4 permits manipulation of remote message folders, called "mailboxes", in a way that is functionally equivalent to local mailboxes. IMAP4 also provides the capability for an offline client to resynchronize with the server (see also [IMAP-DISC]).

IMAP4 includes operations for creating, deleting, and renaming mailboxes; checking for new messages; permanently removing messages; setting and clearing flags; RFC 822 and MIME parsing; searching; and selective fetching of message attributes, texts, and portions thereof. Messages in IMAP4 are accessed by the use of numbers. These numbers are either message sequence numbers (relative position from 1 to the number of messages in the mailbox) or unique identifiers (immutable, strictly ascending values assigned to each message, but which are not necessarily contiguous).

IMAP4 supports a single server. A mechanism for supporting multiple IMAP4 servers is discussed in [IMSP].

IMAP4 does not specify a means of posting mail; this function is handled by a mail transfer protocol such as [SMTP].

IMAP4 is designed to be upwards compatible from the [IMAP2] protocol. Compatibility issues are discussed in [IMAP-COMPAT].

Table of Contents

IMAP4 Protocol Specification

1. Organization of this Document

1.1. How to Read This Document

This document is written from the point of view of the implementor of an IMAP4 client or server. Beyond the protocol overview in section 2, it is not optimized for someone trying to understand the operation of the protocol. The material in sections 3 through 5 provides the general context and definitions with which IMAP4 operates.

Sections 6, 7, and 9 describe the IMAP commands, responses, and syntax, respectively. The relationships among these are such that it is almost impossible to understand any of them separately. In particular, one should not attempt to deduce command syntax from the command section alone; one should instead refer to the formal syntax section.

1.2. Conventions Used in this Document

In examples, "C:" and "S:" indicate lines sent by the client and server respectively.

2. Protocol Overview

2.1. Link Level

The IMAP4 protocol assumes a reliable data stream such as provided by TCP. When TCP is used, an IMAP4 server listens on port 143.

2.2. Commands and Responses

An IMAP4 session consists of the establishment of a client/server connection, an initial greeting from the server, and client/server interactions. These client/server interactions consist of a client command, server data, and a server completion result response.

All interactions transmitted by client and server are in the form of lines; that is, strings that end with a CRLF. The protocol receiver of an IMAP4 client or server is either reading a line, or is reading a sequence of octets with a known count followed by a line.

2.2.1. Client Protocol Sender and Server Protocol Receiver

The client command begins an operation. Each client command is prefixed with a identifier (typically a short alphanumeric string, e.g. A0001, A0002, etc.) called a "tag". A different tag is generated by the client for each command.

There are two cases in which a line from the client does not represent a complete command. In one case, a command argument is quoted with an octet count (see the description of literal in String under Data Formats); in the other case, the command arguments require server feedback (see the AUTHENTICATE command). In either case, the server sends a command continuation request response if it is ready for the octets (if appropriate) and the remainder of the command. This response is prefixed with the token "+".

> Note: If, instead, the server detected an error in the command, it sends a BAD completion response with tag matching the command (as described below) to reject the command and prevent the client from sending any more of the command.
>
> It is also possible for the server to send a completion response for some other command (if multiple commands are in progress), or untagged data. In either case, the command continuation request is still pending; the client takes the appropriate action for the response, and reads another response from the server.

The protocol receiver of an IMAP4 server reads a command line from the client, parses the command and its arguments, and transmits server data and a server command completion result response.

2.2.2. Server Protocol Sender and Client Protocol Receiver

Data transmitted by the server to the client and status responses that do not indicate command completion are prefixed with the token "*", and are called untagged responses.

Server data may be sent as a result of a client command, or may be sent unilaterally by the server. There is no syntactic difference between server data that resulted from a specific command and server data that were sent unilaterally.

The server completion result response indicates the success or failure of the operation. It is tagged with the same tag as the client command which began the operation. Thus, if more than one

command is in progress, the tag in a server completion response identifies the command to which the response applies. There are three possible server completion responses: OK (indicating success), NO (indicating failure), or BAD (indicating protocol error such as unrecognized command or command syntax error).

The protocol receiver of an IMAP4 client reads a response line from the server. It then takes action on the response based upon the first token of the response, which may be a tag, a "*", or a "+". As described above.

A client MUST be prepared to accept any server response at all times. This includes server data that it may not have requested. Server data SHOULD be recorded, so that the client can reference its recorded copy rather than sending a command to the server to request the data. In the case of certain server data, recording the data is mandatory.

This topic is discussed in greater detail in the Server Responses section.

3. State and Flow Diagram

An IMAP4 server is in one of four states. Most commands are valid in only certain states. It is a protocol error for the client to attempt a command while the command is in an inappropriate state. In this case, a server will respond with a BAD or NO (depending upon server implementation) command completion result.

3.1. Non-Authenticated State

In non-authenticated state, the user must supply authentication credentials before most commands will be permitted. This state is entered when a connection starts unless the connection has been pre-authenticated.

3.2. Authenticated State

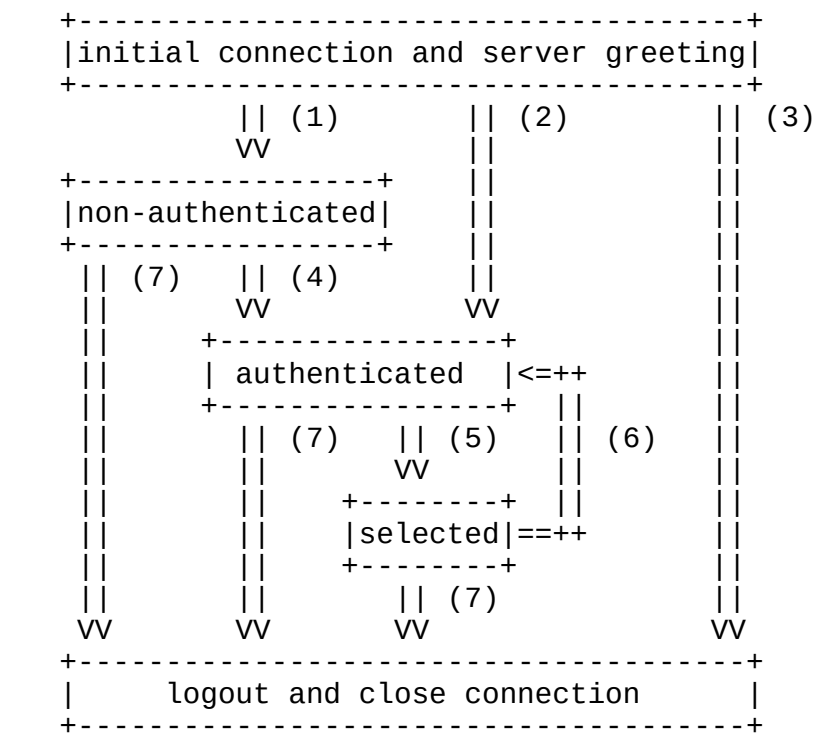
In authenticated state, the user is authenticated and must select a mailbox to access before commands that affect messages will be permitted. This state is entered when a pre-authenticated connection starts, when acceptable authentication credentials have been provided, or after an error in selecting a mailbox.

3.3. Selected State

In selected state, a mailbox has been selected to access. This state is entered when a mailbox has been successfully selected.

3.4. Logout State

In logout state, the session is being terminated, and the server will close the connection. This state can be entered as a result of a client request or by unilateral server decision.



(1) connection without pre-authentication (OK greeting)
(2) pre-authenticated connection (PREAUTH greeting)
(3) rejected connection (BYE greeting)
(4) successful LOGIN or AUTHENTICATE command
(5) successful SELECT or EXAMINE command
(6) CLOSE command, or failed SELECT or EXAMINE command
(7) LOGOUT command, server shutdown, or connection closed

4. Data Formats

IMAP4 uses textual commands and responses. Data in IMAP4 can be in one of several forms: atom, number, string, parenthesized list, or NIL.

4.1. Atom

An atom consists of one or more non-special characters.

4.2. Number

A number consists of one or more digit characters, and represents a numeric value.

4.3. String

A string is in one of two forms: literal and quoted string. The literal form is the general form of string. The quoted string form is an alternative that avoids the overhead of processing a literal at the cost of restrictions of what may be in a quoted string.

A literal is a sequence of zero or more octets (including CR and LF), prefix-quoted with an octet count in the form of an open brace ("{"), the number of octets, close brace ("}"), and CRLF. In the case of literals transmitted from server to client, the CRLF is immediately followed by the octet data. In the case of literals transmitted from client to server, the client must wait to receive a command continuation request (described later in this document) before sending the octet data (and the remainder of the command).

A quoted string is a sequence of zero or more 7-bit characters, excluding CR and LF, with double quote (<">) characters at each end.

The empty string is respresented as either "" (a quoted string with zero characters between double quotes) or as {0} followed by CRLF (a literal with an octet count of 0).

> Note: Even if the octet count is 0, a client transmitting a literal must wait to receive a command continuation request.

4.3.1. 8-bit and Binary Strings

8-bit textual and binary mail is supported through the use of [MIME-1] encoding. IMAP4 implementations MAY transmit 8-bit or multi-octet characters in literals, but should do so only when the character set is identified.

Although a BINARY body encoding is defined, unencoded binary strings are not permitted. A "binary string" is any string with NUL characters. Implementations MUST encode binary data into a textual form such as BASE64 before transmitting the data. A string with an excessive amount of CTL characters may also be considered to be binary, although this is not required.

4.4. Parenthesized List

Data structures are represented as a "parenthesized list"; a sequence of data items, delimited by space, and bounded at each end by parentheses. A parenthesized list may itself contain other parenthesized lists, using multiple levels of parentheses to indicate nesting.

The empty list is represented as () -- a parenthesized list with no members.

4.5. NIL

The special atom "NIL" represents the non-existence of a particular data item that is represented as a string or parenthesized list, as distinct from the empty string "" or the empty parenthesized list ().

5. Operational Considerations

5.1. Mailbox Naming

The interpretation of mailbox names is implementation-dependent. However, the mailbox name INBOX is a special name reserved to mean "the primary mailbox for this user on this server". If it is desired to export hierarchical mailbox names, mailbox names must be left-to-right hierarchical using a single character to separate levels of hierarchy. The same hierarchy separator character is used for all levels of hierarchy within a single name.

5.2. Mailbox Size and Message Status Updates

At any time, a server can send data that the client did not request. Sometimes, such behavior is required. For example, agents other than the server may add messages to the mailbox (e.g. new mail delivery), change the flags of message in the mailbox (e.g. simultaneous access to the same mailbox by multiple agents), or even remove messages from the mailbox. A server MUST send mailbox size updates automatically if a mailbox size change is observed during the processing of a command. A server SHOULD send message flag updates automatically, without requiring the client to request such updates explicitly. Special rules exist for server notification of a client about the removal of messages to prevent synchronization errors; see the description of the EXPUNGE response for more details.

Regardless of what implementation decisions a client may take on remembering data from the server, a client implementation MUST record mailbox size updates. It MUST NOT assume that any command after initial mailbox selection will return the size of the mailbox.

5.3. Response when no Command in Progress

Server implementations are permitted to send an untagged response (except for EXPUNGE) while there is no command in progress. Server implementations that send such responses MUST deal with flow control considerations. Specifically, they must either (1) verify that the size of the data does not exceed the underlying transport’s available window size, or (2) use non-blocking writes.

5.4. Autologout Timer

If a server has an inactivity autologout timer, that timer MUST be of at least 30 minutes' duration. The receipt of ANY command from the client during that interval should suffice to reset the autologout timer.

5.5. Multiple Commands in Progress

The client is not required to wait for the completion result response of a command before sending another command, subject to flow control constraints on the underlying data stream. Similarly, a server is not required to process a command to completion before beginning processing of the next command, unless an ambiguity would result because of a command that would affect the results of other commands. If there is such an ambiguity, the server executes commands to completion in the order given by the client.

6. Client Commands

IMAP4 commands are described in this section. Commands are organized by the state in which the command is permitted. Commands which are permitted in multiple states are listed in the minimum permitted state (for example, commands valid in authenticated and selected state are listed in the authenticated state commands).

Command arguments, identified by "Arguments:" in the command descriptions below, are described by function, not by syntax. The precise syntax of command arguments is described in the Formal Syntax section.

Some commands cause specific server data to be returned; these are identified by "Data:" in the command descriptions below. See the response descriptions in the Responses section for information on these responses, and the Formal Syntax section for the precise syntax of these responses. It is possible for server data to be transmitted as a result of any command; thus, commands that do not specifically require server data specify "no specific data for this command" instead of "none".

The "Result:" in the command description refers to the possible tagged status responses to a command, and any special interpretation of these status responses.

6.1. Client Commands - Any State

The following commands are valid in any state: CAPABILITY, NOOP, and LOGOUT.

6.1.1. CAPABILITY Command

Arguments: none

Data: mandatory untagged response: CAPABILITY

Result: OK - capability completed
BAD - command unknown or arguments invalid

The CAPABILITY command requests a listing of capabilities that the server supports. The server MUST send a single untagged CAPABILITY response with "IMAP4" as the first listed capability before the (tagged) OK response. This listing of capabilities is not dependent upon connection state or user. It is therefore not necessary to issue a CAPABILITY command more than once in a session.

Capability names other than "IMAP4" refer to extensions, revisions, or amendments to this specification. See the documentation of the CAPABILITY response for additional information. No capabilities are enabled without explicit client action to invoke the capability. See the section entitled "Client Commands - Experimental/Expansion" for information about the form of site or implementation-specific capabilities.

Example: C: abcd CAPABILITY
S: * CAPABILITY IMAP4
S: abcd OK CAPABILITY completed

6.1.2. NOOP Command

Arguments: none

Data: no specific data for this command (but see below)

Result: OK - noop completed
BAD - command unknown or arguments invalid

The NOOP command always succeeds. It does nothing.

Since any command can return a status update as untagged data, the NOOP command can be used as a periodic poll for new messages or message status updates during a period of inactivity. The NOOP command can also be used to reset any inactivity autologout timer on the server.

Example: C: a002 NOOP
S: a002 OK NOOP completed
. . .
C: a047 NOOP
S: * 22 EXPUNGE
S: * 23 EXISTS
S: * 3 RECENT
S: * 14 FETCH (FLAGS (\Seen \Deleted))
S: a047 OK NOOP completed

6.1.3. LOGOUT Command

Arguments: none

Data: mandatory untagged response: BYE

Result: OK - logout completed
BAD - command unknown or arguments invalid

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

Example: C: A023 LOGOUT
S: * BYE IMAP4 Server logging out
S: A023 OK LOGOUT completed
(Server and client then close the connection)

6.2. Client Commands - Non-Authenticated State

In non-authenticated state, the AUTHENTICATE or LOGIN command establishes authentication and enter authenticated state. The AUTHENTICATE command provides a general mechanism for a variety of authentication techniques, whereas the LOGIN command uses the traditional user name and plaintext password pair.

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

Once authenticated (including as anonymous), it is not possible to re-enter non-authenticated state.

In addition to the universal commands (CAPABILITY, NOOP, and LOGOUT), the following commands are valid in non-authenticated state: AUTHENTICATE and LOGIN.

6.2.1. AUTHENTICATE Command

Arguments: authentication mechanism name

Data: continuation data may be requested

Result: OK - authenticate completed, now in authenticated state
NO - authenticate failure: unsupported authentication mechanism, credentials rejected
BAD - command unknown or arguments invalid, authentication exchange cancelled

The AUTHENTICATE command indicates an authentication mechanism, such as described in [IMAP-AUTH], to the server. If the server supports the requested authentication mechanism, it performs an authentication protocol exchange to authenticate and identify the user. Optionally, it also negotiates a protection mechanism for subsequent protocol interactions. If the requested authentication mechanism is not supported, the server should reject the AUTHENTICATE command by sending a tagged NO response.

The authentication protocol exchange consists of a series of server challenges and client answers that are specific to the authentication mechanism. A server challenge consists of a command continuation request response with the "+" token followed by a BASE64 encoded string. The client answer consists of a line consisting of a BASE64 encoded string. If the client wishes to cancel an authentication exchange, it should issue a line with a single "*". If the server receives such an answer, it must reject the AUTHENTICATE command by sending a tagged BAD response.

A protection mechanism provides integrity and privacy protection to the protocol session. If a protection mechanism is negotiated, it is applied to all subsequent data sent over the connection. The protection mechanism takes effect immediately following the CRLF that concludes the authentication exchange for the client, and the CRLF of the tagged OK response for the server. Once the protection mechanism is in effect, the stream of command and response octets is processed into buffers of ciphertext. Each buffer is transferred over the connection as a stream of octets prepended with a four octet field in network byte order that represents the length of the following data. The maximum ciphertext buffer length is defined by the protection mechanism.

The server is not required to support any particular authentication mechanism, nor are authentication mechanisms required to support any protection mechanisms. If an AUTHENTICATE command fails with a NO response, the client may try another

authentication mechanism by issuing another AUTHENTICATE command, or may attempt to authenticate by using the LOGIN command. In other words, the client may request authentication types in decreasing order of preference, with the LOGIN command as a last resort.

```
Example:    S: * OK KerberosV4 IMAP4 Server
            C: A001 AUTHENTICATE KERBEROS_V4
            S: + AmFYig==
            C: BAcAQU5EUkVXLkNNVS5FRFUAOCAsho84kLN3/IJmrMG+25a4DT
               +nZImJjnTNHJUtxAA+o0KPKfHEcAFs9a3CL5Oebe/ydHJUwYFd
               WwuQ1MWiy6IesKvjL5rL9WjXUb9MwT9bpObYLGOKi1Qh
            S: + or//EoAADZI=
            C: DiAF5A4gA+oOIALuBkAAmw==
            S: A001 OK Kerberos V4 authentication successful
```

Note: the line breaks in the first client answer are for editorial clarity and are not in real authenticators.

6.2.2. LOGIN Command

```
Arguments:  user name
            password

Data:       no specific data for this command

Result:     OK - login completed, now in authenticated state
            NO - login failure: user name or password rejected
            BAD - command unknown or arguments invalid
```

The LOGIN command identifies the user to the server and carries the plaintext password authenticating this user.

```
Example:    C: a001 LOGIN SMITH SESAME
            S: a001 OK LOGIN completed
```

6.3. Client Commands - Authenticated State

In authenticated state, commands that manipulate mailboxes as atomic entities are permitted. Of these commands, the SELECT and EXAMINE commands will select a mailbox for access and enter selected state.

In addition to the universal commands (CAPABILITY, NOOP, and LOGOUT), the following commands are valid in authenticated state: SELECT, EXAMINE, CREATE, DELETE, RENAME, SUBSCRIBE, UNSUBSCRIBE, LIST, LSUB, and APPEND.

6.3.1. SELECT Command

Arguments: mailbox name

Data: mandatory untagged responses: FLAGS, EXISTS, RECENT
optional OK untagged responses: UNSEEN, PERMANENTFLAGS

Result: OK - select completed, now in selected state
NO - select failure, now in authenticated state: no such mailbox, can't access mailbox
BAD - command unknown or arguments invalid

The SELECT command selects a mailbox so that messages in the mailbox can be accessed. Before returning an OK to the client, the server MUST send the following untagged data to the client:

FLAGS Defined flags in the mailbox

<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

OK [UIDVALIDITY <n>]
The unique identifier validity value. See the description of the UID command for more detail.

to define the initial state of the mailbox at the client. If it is not possible to determine the messages that were added since the previous time a mailbox was read, then all messages SHOULD be considered recent.

The server SHOULD also send an UNSEEN response code in an OK untagged response, indicating the message sequence number of the first unseen message in the mailbox.

If the client can not change the permanent state of one or more of the flags listed in the FLAGS untagged response, the server SHOULD send a PERMANENTFLAGS response code in an OK untagged response, listing the flags that the client may change permanently.

Only one mailbox may be selected at a time in a session; simultaneous access to multiple mailboxes requires multiple

sessions. The SELECT command automatically deselects any currently selected mailbox before attempting the new selection. Consequently, if a mailbox is selected and a SELECT command that fails is attempted, no mailbox is selected.

If the user is permitted to modify the mailbox, the server SHOULD prefix the text of the tagged OK response with the "[READ-WRITE]" response code.

If the user is not permitted to modify the mailbox but is permitted read access, the mailbox is selected as read-only, and the server MUST prefix the text of the tagged OK response to SELECT with the "[READ-ONLY]" response code. Read-only access through SELECT differs from the EXAMINE command in that certain read-only mailboxes may permit the change of permanent state on a per-user (as opposed to global) basis. Netnews messages marked in a user's .newsrc file are an example of such per-user permanent state that can be modified with read-only mailboxes.

```
Example:    C: A142 SELECT INBOX
            S: * 172 EXISTS
            S: * 1 RECENT
            S: * OK [UNSEEN 12] Message 12 is first unseen
            S: * OK [UIDVALIDITY 3857529045] UIDs valid
            S: * FLAGS (\Answered \Flagged \Deleted \Seen \Draft)
            S: * OK [PERMANENTFLAGS (\Deleted \Seen \*)] Limited
            S: A142 OK [READ-WRITE] SELECT completed
```

6.3.2. EXAMINE Command

```
Arguments:  mailbox name

Data:       mandatory untagged responses: FLAGS, EXISTS, RECENT
            optional OK untagged responses: UNSEEN, PERMANENTFLAGS

Result:     OK - examine completed, now in selected state
            NO - examine failure, now in authenticated state: no
                 such mailbox, can't access mailbox
            BAD - command unknown or arguments invalid
```

The EXAMINE command is identical to SELECT and returns the same output; however, the selected mailbox is identified as read-only. No changes to the permanent state of the mailbox, including per-user state, are permitted.

The text of the tagged OK response to the EXAMINE command MUST begin with the "[READ-ONLY]" response code.

```
Example:    C: A932 EXAMINE blurdybloop
            S: * 17 EXISTS
            S: * 2 RECENT
            S: * OK [UNSEEN 8] Message 8 is first unseen
            S: * OK [UIDVALIDITY 3857529045] UIDs valid
            S: * FLAGS (\Answered \Flagged \Deleted \Seen \Draft)
            S: * OK [PERMANENTFLAGS ()] No permanent flags permitted
            S: A932 OK [READ-ONLY] EXAMINE completed
```

6.3.3. CREATE Command

```
Arguments:  mailbox name

Data:       no specific data for this command

Result:     OK - create completed
            NO - create failure: can't create mailbox with that name
            BAD - command unknown or arguments invalid
```

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

If the mailbox name is suffixed with the server's hierarchy separator character (as returned from the server by a LIST command), this is a declaration that the client may, in the future, create mailbox names under this name in the hierarchy. Server implementations that do not require this declaration MUST ignore it.

If a new mailbox is created with the same name as a mailbox which was deleted, its unique identifiers MUST be greater than any unique identifiers used in the previous incarnation of the mailbox UNLESS the new incarnation has a different unique identifier validity value. See the description of the UID command for more detail.

```
Example:    C: A003 CREATE owatagusiam/
            S: A003 OK CREATE completed
            C: A004 CREATE owatagusiam/blurdybloop
            S: A004 OK CREATE completed
```

Note: the interpretation of this example depends on whether "/" was returned as the hierarchy separator from LIST. If "/" is the hierarchy separator, a new level of hierarchy named "owatagusiam" with a member called "blurdybloop" is created. Otherwise, two mailboxes at the same hierarchy level are created.

6.3.4. DELETE Command

Arguments: mailbox name

Data: no specific data for this command

Result: OK - delete completed
NO - delete failure: can't delete mailbox with that name
BAD - command unknown or arguments invalid

The DELETE command permanently removes the mailbox with the given name. A tagged OK response is returned only if the mailbox has been deleted. It is an error to attempt to delete INBOX or a mailbox name that does not exist. Any error in deletion will return a tagged NO response.

The value of the highest-used unique indentifier of the deleted mailbox MUST be preserved so that a new mailbox created with the same name will not reuse the identifiers of the former incarnation, UNLESS the new incarnation has a different unique identifier validity value. See the description of the UID command for more detail.

Example: C: A683 DELETE blurdybloop
S: A683 OK DELETE completed

6.3.5. RENAME Command

Arguments: existing mailbox name
new mailbox name

Data: no specific data for this command

Result: OK - rename completed
NO - rename failure: can't rename mailbox with that name, can't rename to mailbox with that name
BAD - command unknown or arguments invalid

The RENAME command changes the name of a mailbox. A tagged OK response is returned only if the mailbox has been renamed. It is an error to attempt to rename from a mailbox name that does not exist or to a mailbox name that already exists. Any error in renaming will return a tagged NO response.

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

Example: C: Z4S9 RENAME blurdybloop owatagusiam
S: Z4S9 OK RENAME completed

6.3.6. SUBSCRIBE Command

Arguments: mailbox

Data: no specific data for this command

Result: OK - subscribe completed
NO - subscribe failure: can't subscribe to that name
BAD - command unknown or arguments invalid

The SUBSCRIBE command adds the specified mailbox name to the server's set of "active" or "subscribed" mailboxes as returned by the LSUB command. This command returns a tagged OK response only if the subscription is successful.

Example: C: A002 SUBSCRIBE #news.comp.mail.mime
S: A002 OK SUBSCRIBE completed

6.3.7. UNSUBSCRIBE Command

Arguments: mailbox name

Data: no specific data for this command

Result: OK - unsubscribe completed
NO - unsubscribe failure: can't unsubscribe that name
BAD - command unknown or arguments invalid

The UNSUBSCRIBE command removes the specified mailbox name from the server's set of "active" or "subscribed" mailboxes as returned by the LSUB command. This command returns a tagged OK response only if the unsubscription is successful.

Example: C: A002 UNSUBSCRIBE #news.comp.mail.mime
S: A002 OK UNSUBSCRIBE completed

6.3.8. LIST Command

Arguments: reference name
mailbox name with possible wildcards

Data: untagged responses: LIST

Result: OK - list completed
NO - list failure: can't list that reference or name
BAD - command unknown or arguments invalid

The LIST command returns a subset of names from the complete set of all names available to the user. Zero or more untagged LIST replies are returned, containing the name attributes, hierarchy delimiter, and name; see the description of the LIST reply for more detail.

An empty ("" string) reference name argument indicates that the mailbox name is interpreted as by SELECT. The returned mailbox names MUST match the supplied mailbox name pattern. A non-empty reference name argument is the name of a mailbox or a level of mailbox hierarchy, and indicates a context in which the mailbox name is interpreted in an implementation-defined manner.

The reference and mailbox name arguments are interpreted, in an implementation-dependent fashion, into a canonical form that represents an unambiguous left-to-right hierarchy. The returned mailbox names will be in the interpreted form.

Any part of the reference argument that is included in the interpreted form SHOULD prefix the interpreted form. It should also be in the same form as the reference name argument. This rule permits the client to determine if the returned mailbox name is in the context of the reference argument, or if something about the mailbox argument overrode the reference argument. Without this rule, the client would have to have knowledge of the server's naming semantics including what characters are "breakouts" that override a naming context.

For example, here are some examples of how references and mailbox names might be interpreted on a UNIX-based server:

Reference	Mailbox Name	Interpretation
~smith/Mail/	foo.*	~smith/Mail/foo.*
archive/	%	archive/%
#news.	comp.mail.*	#news.comp.mail.*
~smith/Mail/	/usr/doc/foo	/usr/doc/foo
archive/	~fred/Mail/*	~fred/Mail/*

The first three examples demonstrate interpretations in the context of the reference argument. Note that "~smith/Mail" should not be transformed into something like "/u2/users/smith/Mail", or it would be impossible for the client to determine that the interpretation was in the context of the reference.

The character "*" is a wildcard, and matches zero or more characters at this position. The character "%" is similar to "*", but it does not match a hierarchy delimiter. If the "%" wildcard is the last character of a mailbox name argument, matching levels of hierarchy are also returned. If these levels of hierarchy are not also selectable mailboxes, they are returned with the \Noselect mailbox name attribute (see the description of the LIST response for more detail).

Server implementations are permitted to "hide" otherwise accessible mailboxes from the wildcard characters, by preventing certain characters or names from matching a wildcard in certain situations. For example, a UNIX-based server might restrict the interpretation of "*" so that an initial "/" character does not match.

The special name INBOX is included in the output from LIST if it matches the input arguments and INBOX is supported by this server 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.

Example:
```
C: A002 LIST "~/Mail/" "%"
S: * LIST (\Noselect) "/" ~/Mail/foo
S: * LIST () "/" ~/Mail/meetings
S: A002 OK LIST completed
```

6.3.9. LSUB Command

Arguments: reference name
mailbox name with possible wildcards

Data: untagged responses: LSUB

Result: OK - lsub completed
NO - lsub failure: can't list that reference or name
BAD - command unknown or arguments invalid

The LSUB command returns a subset of names from the set of names that the user has declared as being "active" or "subscribed". Zero or more untagged LSUB replies are returned. The arguments to LSUB are in the same form as those for LIST.

Example: C: A002 LSUB "#news." "comp.mail.*"
S: * LSUB () "." #news.comp.mail.mime
S: * LSUB () "." #news.comp.mail.misc
S: A002 OK LSUB completed

6.3.10. APPEND Command

Arguments: mailbox name
optional flag parenthesized list
optional date/time string
message literal

Data: no specific data for this command

Result: OK - append completed
NO - append error: can't append to that mailbox, error in flags or date/time or message text
BAD - command unknown or arguments invalid

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. 8-bit characters are permitted in the message. A server implementation that is unable to preserve 8-bit data properly MUST be able to reversibly convert 8-bit APPEND data to 7-bit using [MIME-1] encoding.

If a flag parenthesized list or date_time are specified, that data SHOULD be set in the resulting message; otherwise, the defaults of empty flags and the current date/time are used.

If the append is unsuccessful for any reason, the mailbox MUST be restored to its state before the APPEND attempt; no partial appending is permitted. If the mailbox is currently selected, the normal new mail actions should occur.

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 response code "[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.

```
Example:    C: A003 APPEND saved-messages (\Seen) {310}
            C: Date: Mon, 7 Feb 1994 21:52:25 -0800 (PST)
            C: From: Fred Foobar <foobar@Blurdybloop.COM>
            C: Subject: afternoon meeting
            C: To: mooch@owatagu.siam.edu
            C: Message-Id: <B27397-0100000@Blurdybloop.COM>
            C: MIME-Version: 1.0
            C: Content-Type: TEXT/PLAIN; CHARSET=US-ASCII
            C:
            C: Hello Joe, do you think we can meet at 3:30 tomorrow?
            C:
            S: A003 OK APPEND completed
```

Note: the APPEND command is not used for message delivery, because it does not provide a mechanism to transfer [SMTP] envelope information.

6.4. Client Commands - Selected State

In selected state, commands that manipulate messages in a mailbox are permitted.

In addition to the universal commands (CAPABILITY, NOOP, and LOGOUT), and the authenticated state commands (SELECT, EXAMINE, CREATE, DELETE, RENAME, SUBSCRIBE, UNSUBSCRIBE, LIST, LSUB, FIND ALL.MAILBOXES, FIND MAILBOXES, and APPEND), the following commands are valid in the selected state: CHECK, CLOSE, EXPUNGE, SEARCH, FETCH, PARTIAL, STORE, COPY, and UID.

6.4.1. CHECK Command

Arguments: none

Data: no specific data for this command

Result: OK - check completed
BAD - command unknown or arguments invalid

The CHECK command requests a checkpoint of the currently selected mailbox. A checkpoint refers to any implementation-dependent housekeeping associated with the mailbox (e.g. resolving the server's in-memory state of the mailbox with the state on its disk) that is not normally executed as part of each command. A checkpoint may take a non-instantaneous amount of real time to complete. If a server implementation has no such housekeeping considerations, CHECK is equivalent to NOOP.

There is no guarantee that an EXISTS untagged response will happen as a result of CHECK. NOOP, not CHECK, should be used for new mail polling.

Example: C: FXXZ CHECK
S: FXXZ OK CHECK Completed

6.4.2. CLOSE Command

Arguments: none

Data: no specific data for this command

Result: OK - close completed, now in authenticated state
NO - close failure: no mailbox selected
BAD - command unknown or arguments invalid

The CLOSE command permanently removes from the currently selected mailbox all messages that have the \Deleted flag set, and returns to authenticated state from selected state. No untagged EXPUNGE responses are sent.

No messages are removed, and no error is given, if the mailbox is selected by an EXAMINE command or is otherwise selected read-only.

Even when a mailbox is selected, it is not required to send a CLOSE command before a SELECT, EXAMINE, or LOGOUT command. The SELECT, EXAMINE, and LOGOUT commands implicitly close the currently selected mailbox without doing an expunge. However,

when many messages are deleted, a CLOSE-LOGOUT or CLOSE-SELECT sequence is considerably faster than an EXPUNGE-LOGOUT or EXPUNGE-SELECT because no untagged EXPUNGE responses (which the client would probably ignore) are sent.

Example: C: A341 CLOSE
S: A341 OK CLOSE completed

6.4.3. EXPUNGE Command

Arguments: none

Data: untagged responses: EXPUNGE

Result: OK - expunge completed
NO - expunge failure: can't expunge (e.g. permission denied)
BAD - command unknown or arguments invalid

The EXPUNGE command permanently removes from the currently selected mailbox all messages that have the \Deleted flag set. Before returning an OK to the client, an untagged EXPUNGE response is sent for each message that is removed.

Example: C: A202 EXPUNGE
S: * 3 EXPUNGE
S: * 3 EXPUNGE
S: * 5 EXPUNGE
S: * 8 EXPUNGE
S: A202 OK EXPUNGE completed

Note: in this example, messages 3, 4, 7, and 11 had the \Deleted flag set. See the description of the EXPUNGE response for further explanation.

6.4.4. SEARCH Command

Arguments: optional character set specification
searching criteria (one or more)

Data: mandatory untagged response: SEARCH

Result: OK - search completed
NO - search error: can't search that character set or criteria
BAD - command unknown or arguments invalid

The SEARCH command searches the mailbox for messages that match the given searching criteria. Searching criteria consist of one or more search keys. The untagged SEARCH response from the server contains a listing of message sequence numbers corresponding to those messages that match the searching criteria.

When multiple keys are specified, the result is the intersection (AND function) of all the messages that match those keys. For example, the criteria DELETED FROM "SMITH" SINCE 1-Feb-1994 refers to all deleted messages from Smith that were placed in the mailbox since February 1, 1994. A search key may also be a parenthesized list of one or more search keys (e.g. for use with the OR and NOT keys).

Server implementations MAY exclude [MIME-1] body parts with terminal content types other than TEXT and MESSAGE from consideration in SEARCH matching.

The optional character set specification consists of the word "CHARSET" followed by a registered MIME character set. It indicates the character set of the strings that appear in the search criteria. [MIME-2] strings that appear in RFC 822/MIME message headers, and [MIME-1] content transfer encodings, MUST be decoded before matching. Except for US-ASCII, it is not required that any particular character set be supported. If the server does not support the specified character set, it MUST return a tagged NO response (not a BAD).

In all search keys that use strings, a message matches the key if the string is a substring of the field. The matching is case-insensitive.

The defined search keys are as follows. Refer to the Formal Syntax section for the precise syntactic definitions of the arguments.

<message set> Messages with message sequence numbers corresponding to the specified message sequence number set

ALL Messages in the mailbox; the default initial key for ANDing.

ANSWERED Messages with the \Answered flag set.

BCC <string> 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 <string> Messages that contain the specified string in the body of the message.

CC <string> Messages that contain the specified string in the envelope structure's CC field.

DELETED Messages with the \Deleted flag set.

DRAFT Messages with the \Draft flag set.

FLAGGED Messages with the \Flagged flag set.

FROM <string> Messages that contain the specified string in the envelope structure's FROM field.

HEADER <field-name> <string>
Messages that have a header with the specified field-name (as defined in [RFC-822]) and that contains the specified string in the [RFC-822] field-body.

KEYWORD <flag> Messages with the specified keyword set.

LARGER <n> Messages with an RFC822.SIZE larger than the specified number of octets.

NEW Messages that have the \Recent flag set but not the \Seen flag. This is functionally equivalent to "(RECENT UNSEEN)".

NOT <search-key>
Messages that do not match the specified search key.

OLD Messages that do not have the \Recent flag set. This is functionally equivalent to "NOT RECENT" (as opposed to "NOT NEW").

ON <date> Messages whose internal date is within the specified date.

OR <search-key1> <search-key2>
Messages that match either search key.

RECENT Messages that have the \Recent flag set.

SEEN Messages that have the \Seen flag set.

SENTBEFORE <date>
Messages whose [RFC-822] Date: header is earlier than the specified date.

SENTON <date> Messages whose [RFC-822] Date: header is within the specified date.

SENTSINCE <date>
Messages whose [RFC-822] Date: header is within or later than the specified date.

SINCE <date> Messages whose internal date is within or later than the specified date.

SMALLER <n> Messages with an RFC822.SIZE smaller than the specified number of octets.

SUBJECT <string>
Messages that contain the specified string in the envelope structure’s SUBJECT field.

TEXT <string> Messages that contain the specified string in the header or body of the message.

TO <string> Messages that contain the specified string in the envelope structure’s TO field.

UID <message set>
Messages with unique identifiers corresponding to the specified unique identifier set.

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

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

UNDRAFT Messages that do not have the \Draft flag set.

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

UNKEYWORD <flag>
Messages that do not have the specified keyword set.

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

Example: C: A282 SEARCH FLAGGED SINCE 1-Feb-1994 NOT FROM "Smith"
S: * SEARCH 2 84 882
S: A282 OK SEARCH completed

6.4.5. FETCH Command

Arguments: message set
message data item names

Data: untagged responses: FETCH

Result: OK - fetch completed
NO - fetch error: can't fetch that data
BAD - command unknown or arguments invalid

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 a parenthesized 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.

Here is an example of a complex message with its associated section specifications:

```
 0    ([RFC-822] header of the message)
      MULTIPART/MIXED
 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 responses.

BODY.PEEK[<section>]

An alternate form of BODY[section] that does not implicitly set the \Seen flag.

BODYSTRUCTURE The [MIME-1] body structure of the message. This is computed by the server by parsing the [MIME-1] header lines.

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

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

FLAGS The flags that are set for this message.

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

INTERNALDATE The date and time of final delivery of the message as defined by RFC 821.

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 responses. This is the concatenation of RFC822.HEADER and RFC822.TEXT.

RFC822.PEEK An alternate form of RFC822 that does not implicitly set the \Seen flag.

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_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_list. The matching is case-insensitive but otherwise exact. The delimiting blank line between the header and the body is always included.

RFC822.HEADER.LINES.NOT <header_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_list. The matching is case-insensitive but otherwise exact. The delimiting blank line between the header and the body is always included.

RFC822.SIZE The number of octets 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 responses.

RFC822.TEXT.PEEK
An alternate form of RFC822.TEXT that does not implicitly set the \Seen flag.

UID The unique identifier for the message.

Example: C: A654 FETCH 2:4 (FLAGS RFC822.HEADER.LINES (DATE FROM))
S: * 2 FETCH ....
S: * 3 FETCH ....
S: * 4 FETCH ....
S: A003 OK FETCH completed

6.4.6. PARTIAL Command

Arguments: message sequence number
message data item name
position of first octet
number of octets

Data: untagged responses: FETCH

Result: OK - partial completed
NO - partial error: can't fetch that data
BAD - command unknown or arguments invalid

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. 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, BODY[section], as well as any .PEEK forms of these.

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. It is not possible to stream multiple PARTIAL commands of the same data item without processing and synchronizing at each step, since streamed commands may be executed out of order.

There is no requirement that partial fetches follow any sequence. For example, if a partial fetch of octets 1 through 10000 breaks in an awkward place for BASE64 decoding, it is permitted to continue with a partial fetch of 9987 through 19987, etc.

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

```
Example:    C: A005 PARTIAL 4 RFC822 1 1024
            S: * 1 FETCH (RFC822 {1024}
            S: Return-Path: <gray@cac.washington.edu>
            S: ...
            S: .........  FLAGS (\Seen))
            S: A005 OK PARTIAL completed
```

6.4.7. STORE Command

```
Arguments:  message set
            message data item name
            value for message data item

Data:       untagged responses: FETCH

Result:     OK - store completed
            NO - store error: can't store that data
            BAD - command unknown or arguments invalid
```

The STORE command alters data associated with a message in the mailbox. Normally, STORE will return the updated value of the data with an untagged FETCH response. A suffix of ".SILENT" in

the data item name prevents the untagged FETCH, and the server should assume that the client has determined the updated value itself or does not care about the updated value.

The currently defined data items that can be stored are:

FLAGS <flag list>
Replace the flags for the message with the argument. The new value of the flags are returned as if a FETCH of those flags was done.

FLAGS.SILENT <flag list>
Equivalent to FLAGS, but without returning a new value.

+FLAGS <flag list>
Add the argument to the flags for the message. The new value of the flags are returned as if a FETCH of those flags was done.

+FLAGS.SILENT <flag list>
Equivalent to +FLAGS, but without returning a new value.

-FLAGS <flag list>
Remove the argument from the flags for the message. The new value of the flags are returned as if a FETCH of those flags was done.

-FLAGS.SILENT <flag list>
Equivalent to -FLAGS, but without returning a new value.

```
Example:    C: A003 STORE 2:4 +FLAGS (\Deleted)
            S: * 2 FETCH FLAGS (\Deleted \Seen)
            S: * 3 FETCH FLAGS (\Deleted)
            S: * 4 FETCH FLAGS (\Deleted \Flagged \Seen)
            S: A003 OK STORE completed
```

6.4.8. COPY Command

Arguments: message set
mailbox name

Data: no specific data for this command

Result: OK - copy completed
NO - copy error: can't copy those messages or to that name
BAD - command unknown or arguments invalid

The COPY command copies the specified message(s) to the specified destination mailbox. The flags and internal date 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 response code "[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, server implementations MUST restore the destination mailbox to its state before the COPY attempt.

Example: C: A003 COPY 2:4 MEETING
S: A003 OK COPY completed

6.4.9. UID Command

Arguments: command name
command arguments

Data: untagged responses: FETCH, SEARCH

Result: OK - UID command completed
NO - UID command error
BAD - command unknown or arguments invalid

The UID command has two forms. In the first form, it takes as its arguments a COPY, FETCH, or STORE command with arguments appropriate for the associated command. However, the numbers in the message set argument are unique identifiers instead of message sequence numbers.

In the second form, the UID command takes a SEARCH command with SEARCH command arguments. The interpretation of the arguments is the same as with SEARCH; however, the numbers returned in a SEARCH response for a UID SEARCH command are unique identifiers instead of message sequence numbers. For example, the command UID SEARCH 1:100 UID 443:557 returns the unique identifiers corresponding to the intersection of the message sequence number set 1:100 and the UID set 443:557.

A unique identifier of a message is a number, and is guaranteed not to refer to any other message in the mailbox. Unique identifiers are assigned in a strictly ascending fashion for each message added to the mailbox. Unlike message sequence numbers, unique identifiers persist across sessions. This permits a client to resynchronize its state from a previous session with the server (e.g. disconnected or offline access clients); this is discussed further in [IMAP-DISC].

Associated with every mailbox is a unique identifier validity value, which is sent in an UIDVALIDITY response code in an OK untagged response at message selection time. If unique identifiers from an earlier session fail to persist to this session, the unique identifier validity value MUST be greater than in the earlier session.

> Note: An example of a good value to use for the unique identifier validity value would be a 32-bit representation of the creation date/time of the mailbox. It is alright to use a constant such as 1, but only if it guaranteed that unique identifers will never be reused, even in the case of a mailbox being deleted and a new mailbox by the same name created at some future time.

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

The number after the "*" in an untagged FETCH response is always a message sequence number, not a unique identifier, even for a UID command response. However, server implementations MUST implicitly include the UID message data item as part of any FETCH response caused by a UID command, regardless of whether UID was specified as a message data item to the FETCH.

```
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: A999 UID FETCH completed
```

6.5. Client Commands - Experimental/Expansion

6.5.1. X<atom> Command

Arguments: implementation defined

Data: implementation defined

Result: OK - command completed
NO - failure
BAD - command unknown or arguments invalid

Any command prefixed with an X is an experimental command. Commands which are not part of this specification, or a standard or standards-track revision of this specification, MUST use the X prefix.

Any added untagged responses issued by an experimental command MUST also be prefixed with an X. Server implementations MUST NOT send any such untagged responses, unless the client requested it by issuing the associated experimental command.

```
Example:    C: a441 CAPABILITY
            S: * CAPABILITY IMAP4 XPIG-LATIN
            S: a441 OK CAPABILITY completed
            C: A442 XPIG-LATIN
            S: * XPIG-LATIN ow-nay eaking-spay ig-pay atin-lay
            S: A442 OK XPIG-LATIN ompleted-cay
```

7. Server Responses

Server responses are in three forms: status responses, server data, and command continuation request.

Server response data, identified by "Data:" in the response descriptions below, are described by function, not by syntax. The precise syntax of server response data is described in the Formal Syntax section.

The client MUST be prepared to accept any response at all times.

Status responses that are tagged indicate the completion result of a client command, and have a tag matching the command.

Some status responses, and all server data, are untagged. An untagged response is indicated by the token "*" instead of a tag. Untagged status responses indicate server greeting, or server status that does not indicate the completion of a command. For historical reasons, untagged server data responses are also called "unsolicited data", although strictly speaking only unilateral server data is truly "unsolicited".

Certain server data MUST be recorded by the client when it is received; this is noted in the description of that data. Such data conveys critical information which affects the interpretation of all subsequent commands and responses (e.g. updates reflecting the creation or destruction of messags).

Other server data SHOULD be recorded for later reference; if the client does not need to record the data, or if recording the data has no obvious purpose (e.g. a SEARCH response when no SEARCH command is in progress), the data SHOULD be ignored.

An example of unilateral untagged responses occurs when the IMAP connection is in selected state. In selected state, the server checks the mailbox for new messages as part of the execution of each command. If new messages are found, the server sends untagged EXISTS and RECENT responses reflecting the new size of the mailbox. Server implementations that offer multiple simultaneous access to the same mailbox should also send appropriate unilateral untagged FETCH and EXPUNGE responses if another agent changes the state of any message flags or expunges any messages.

Command continuation request responses use the token "+" instead of a tag. These responses are sent by the server to indicate acceptance of an incomplete client command and readiness for the remainder of the command.

7.1. Server Responses - Status Responses

Status responses may include an optional response code. A response code consists of data inside square brackets in the form of an atom, possibly followed by a space and arguments. The response code contains additional information or status codes for client software beyond the OK/NO/BAD condition, and are defined when there is a specific action that a client can take based upon the additional information.

The currently defined response codes are:

ALERT The human-readable text contains a special alert that MUST be presented to the user in a fashion that calls the user's attention to the message.

PARSE The human-readable text represents an error in parsing the [RFC-822] or [MIME-1] headers of a message in the mailbox.

PERMANENTFLAGS Followed by a parenthesized list of flags, indicates which of the known flags that the client may change permanently. Any flags that are in the FLAGS untagged response, but not the PERMANENTFLAGS list, can not be set permanently. If the client attempts to STORE a flag that is not in the PERMANENTFLAGS list, the server will either reject it with a NO reply or store the state for the remainder of the current session only. The PERMANENTFLAGS list may also include the special flag *, which indicates that it is possible to create new keywords by attempting to store those flags in the mailbox.

READ-ONLY The mailbox is selected read-only, or its access while selected has changed from read-write to read-only.

READ-WRITE The mailbox is selected read-write, or its access while selected has changed from read-only to read-write.

TRYCREATE An APPEND or COPY attempt is failing because the target mailbox does not exist (as opposed to some other reason). This is a hint to the client that the operation may succeed if the mailbox is first created by the CREATE command.

UIDVALIDITY Followed by a decimal number, indicates the unique identifier validity value. See the description of the UID command for more detail.

UNSEEN Followed by a decimal number, indicates the number of the first message without the \Seen flag set.

Additional response codes defined by particular client or server implementations should be prefixed with an "X" until they are added to a revision of this protocol. Client implementations should ignore response codes that they do not recognize.

7.1.1. OK Response

Data: optional response code
human-readable text

The OK response indicates an information message from the server. When tagged, it indicates successful completion of the associated command. The human-readable text may be presented to the user as an information message. The untagged form indicates an information-only message; the nature of the information may be indicated by a response code.

The untagged form is also used as one of three possible greetings at session startup. It indicates that the session is not yet authenticated and that a LOGIN command is needed.

Example:
```
S: * OK IMAP4 server ready
C: A001 LOGIN fred blurdybloop
S: * OK [ALERT] System shutdown in 10 minutes
S: A001 OK LOGIN Completed
```

7.1.2. NO Response

Data: optional response code
human-readable text

The NO response indicates an operational error message from the server. When tagged, it indicates unsuccessful completion of the associated command. The untagged form indicates a warning; the command may still complete successfully. The human-readable text describes the condition.

```
Example:    C: A222 COPY 1:2 owatagusiam
            S: * NO Disk is 98% full, please delete unnecessary data
            S: A222 OK COPY completed
            C: A222 COPY 3:200 blurdybloop
            S: * NO Disk is 98% full, please delete unnecessary data
            S: * NO Disk is 99% full, please delete unnecessary data
            S: A222 NO COPY failed: disk is full
```

7.1.3. BAD Response

Data: optional response code
human-readable text

The BAD response indicates an error message from the server. When tagged, it reports a protocol-level error in the client's command; the tag indicates the command that caused the error. The untagged form indicates a protocol-level error for which the associated command can not be determined; it may also indicate an internal server failure. The human-readable text describes the condition.

```
Example:    C: ...very long command line...
            S: * BAD Command line too long
            C: ...empty line...
            S: * BAD Empty command line
            C: A443 EXPUNGE
            S: * BAD Disk crash, attempting salvage to a new disk!
            S: * OK Salvage successful, no data lost
            S: A443 OK Expunge completed
```

7.1.4. PREAUTH Response

Data: optional response code
human-readable text

The PREAUTH response is always untagged, and 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.

```
Example:    S: * PREAUTH IMAP4 server ready and logged in as Smith
```

7.1.5. BYE Response

Data: optional response code
human-readable text

The BYE response is always untagged, and indicates that the server is about to close the connection. The human-readable text may be displayed to the user in a status report by the client. The BYE response 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 server implementations as an announcement of an inactivity autologout.

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

Example: S: * BYE Autologout; idle for too long

7.2. Server Responses - Server and Mailbox Status

These responses are always untagged. This is how server data are transmitted from the server to the client, often as a result of a command with the same name.

7.2.1. CAPABILITY Response

Data: capability listing

The CAPABILITY response occurs as a result of a CAPABILITY command. The capability listing contains a space-separated listing of capability names that the server supports. The first name in the capability listing MUST be the atom "IMAP4".

A capability name other than IMAP4 indicates that the server supports an extension, revision, or amendment to the IMAP4 protocol. Server responses MUST conform to this document until the client issues a command that uses the associated capability.

Capability names MUST either begin with "X" or be standard or standards-track IMAP4 extensions, revisions, or amendments registered with IANA. A server MUST NOT offer unregistered or non-standard capability names, unless such names are prefixed with an "X".

Client implementations SHOULD NOT require any capability name other than "IMAP4", and MUST ignore any unknown capability names.

Example: S: * CAPABILITY IMAP4 XPIG-LATIN

7.2.2. LIST Response

Data: name attributes
hierarchy delimiter
name

The LIST response occurs as a result of a LIST command. It returns a single name that matches the LIST specification. There may be multiple LIST responses for a single LIST command.

Four name attributes are defined:

\Noinferiors It is not possible for any child levels of hierarchy to exist under this name; no child levels exist now and none can be created in the future.

\Noselect It is not possible to use this name as a selectable mailbox.

\Marked The mailbox has been marked "interesting" by the server; the mailbox probably contains messages that have been added since the last time the mailbox was selected.

\Unmarked The mailbox does not contain any additional messages since the last time the mailbox was selected.

If it is not feasible for the server to determine whether the mailbox is "interesting" or not, or if the name is a \Noselect name, the server should not send either \Marked or \Unmarked.

The hierarchy delimiter is a character used to delimit levels of hierarchy in a mailbox name. A client may use it to create child mailboxes, and to search higher or lower levels of naming hierarchy. All children of a top-level hierarchy node must use the same separator character. A NIL hierarchy delimiter means that no hierarchy exists; the name is a "flat" name.

The name represents an unambiguous left-to-right hierarchy, and MUST be valid for use as a reference in LIST and LSUB commands. Unless \Noselect is indicated, the name must also be valid as an argument for commands, such as SELECT, that accept mailbox names.

Example: S: * LIST (\Noselect) "/" ~/Mail/foo

7.2.3. LSUB Response

Data: name attributes
hierarchy delimiter
name

The LSUB response occurs as a result of an LSUB command. It returns a single name that matches the LSUB specification. There may be multiple LSUB responses for a single LSUB command. The data is identical in format to the LIST response.

Example: S: * LSUB () "." #news.comp.mail.misc

7.2.4. SEARCH Response

Data: zero or more numbers

The SEARCH response occurs as a result of a SEARCH or UID SEARCH command. The number(s) refer to those messages that match the search criteria. For SEARCH, these are message sequence numbers; for UID SEARCH, these are unique identifiers. Each number is delimited by a space.

Example: S: * SEARCH 2 3 6

7.2.5. FLAGS Response

Data: flag parenthesized list

The FLAGS response occurs as a result of a SELECT or EXAMINE command. The flag parenthesized list identifies the flags (at a minimum, the system-defined flags) that are applicable for this mailbox. Flags other than the system flags may also exist, depending on server implementation.

The update from the FLAGS response MUST be recorded by the client.

Example: S: * FLAGS (\Answered \Flagged \Deleted \Seen \Draft)

7.3. Server Responses - Message Status

These responses are always untagged. This is how message data are transmitted from the server to the client, often as a result of a command with the same name. Immediately following the "*" token is a number that represents either a message sequence number or a message count.

7.3.1. EXISTS Response

Data: none

The EXISTS response reports the number of messages in the mailbox. This response occurs as a result of a SELECT or EXAMINE command, and if the size of the mailbox changes (e.g. new mail).

The update from the EXISTS response MUST be recorded by the client.

Example: S: * 23 EXISTS

7.3.2. RECENT Response

Data: none

The RECENT response reports the number of messages that have arrived since the previous time a SELECT command was done on this mailbox. This response occurs as a result of a SELECT or EXAMINE command, and if the size of the mailbox changes (e.g. new mail).

The update from the RECENT response MUST be recorded by the client.

Example: S: * 5 RECENT

7.3.3. EXPUNGE Response

Data: none

The EXPUNGE response reports that the specified message sequence number has been permanently removed from the mailbox. The message sequence number for each successive message in the mailbox is immediately decremented by 1, and this decrement is reflected in message sequence numbers in subsequent responses (including other untagged EXPUNGE responses).

As a result of the immediate decrement rule, message sequence numbers that appear in a set of successive EXPUNGE responses depend upon whether the messages are removed starting from lower numbers to higher numbers, or from higher numbers to lower numbers. For example, if the last 5 messages in a 9-message mailbox are expunged; a "lower to higher" server will send five untagged EXPUNGE responses for message sequence number 5, whereas a "higher to lower server" will send successive untagged EXPUNGE responses for message sequence numbers 9, 8, 7, 6, and 5.

An EXPUNGE response MUST NOT be sent when no command is in progress; nor while responding to a FETCH, STORE, or SEARCH command. This rule is necessary to prevent a loss of synchronization of message sequence numbers between client and server.

The update from the EXPUNGE response MUST be recorded by the client.

Example: S: * 44 EXPUNGE

7.3.4. FETCH Response

Data: message data

The FETCH response returns data about a message to the client. The data are pairs of data item names and their values in parentheses. This response occurs as the result of a FETCH or STORE command, as well as by unilateral server decision (e.g. flag updates).

The current data items are:

BODY A form of BODYSTRUCTURE without 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.

8-bit textual data is permitted if a character set identifier is part of the body parameter parenthesized list for this section.

Non-textual data such as binary data must be transfer encoded into a textual form such as BASE64 prior to being sent to the client. To derive the

original binary data, the client must decode the transfer encoded string.

BODYSTRUCTURE A parenthesized 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 parenthesis nesting. Instead of a body type as the first element of the parenthesized list there is a nested body. The second element of the parenthesized list is the multipart subtype (mixed, digest, parallel, alternative, etc.).

Extension data follows the multipart subtype. Extension data is never returned with the 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 multipart body part are in the following order:

body parameter parenthesized list
A parenthesized 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-1].

Any following extension data are not yet defined in this version of the protocol. Such extension data may consist of zero or more NILs, strings, numbers, or potentially nested parenthesized lists of such data. Client implementations that do a BODYSTRUCTURE fetch MUST be prepared to accept such extension data. Server implementations MUST NOT send such extension data until it has been defined by a revision of this 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-1].

body subtype
A string giving the content subtype name as defined in [MIME-1].

body parameter parenthesized list
A parenthesized 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-1].

body id
A string giving the content id as defined in [MIME-1].

body description
A string giving the content description as defined in [MIME-1].

body encoding
A string giving the content transfer encoding as defined in [MIME-1].

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 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-1].

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

ENVELOPE A parenthesized 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 parenthesized lists of address structures.

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

[RFC-822] group syntax is indicated by a special form of address structure in which the host name field 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 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 A parenthesized list of flags that are set for this message. This may include keywords as well as 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

\Draft Message has not completed composition (marked as a draft).

as well as the following special flag, which may be fetched but not stored:

\Recent Message has arrived since the previous time this mailbox was selected.

INTERNALDATE A string containing the date and time of final delivery of the message as defined by [SMTP].

RFC822 A string expressing the message in [RFC-822] format. The header portion of the message must be 7-bit. 8-bit characters are permitted only in the non-header portion of the message only if there are [MIME-1] data in the message that identify the character set of the message.

RFC822.HEADER A string expressing the [RFC-822] format header of the message, including the delimiting blank line between the header and the body. The entire string must be 7-bit; 8-bit characters are not permitted in headers. RFC822.HEADER is used to return data for the RFC822.HEADER, RFC822.HEADER.LINES, and RFC822.HEADER.LINES.NOT FETCH data items. Note that a blank line is always included regardless of header line restrictions.

RFC822.SIZE A number expressing the number of octets in the message in [RFC-822] format.

RFC822.TEXT A string expressing the text body of the message, omitting the [RFC-822] header. 8-bit characters are permitted only if there are [MIME-1] data in the message that identify the character set of the message.

UID A number expressing the unique identifier of the message.

Example: S: * 23 FETCH (FLAGS (\Seen) RFC822.SIZE 44827)

7.3.5. Obsolete Responses

In addition to the responses listed in here, client implementations MUST accept and implement the obsolete responses described in Appendix B.

7.4. Server Responses - Command Continuation Request

The command completion request response is indicated by a "+" token instead of a tag. This form of response indicates that the server is ready to accept the continuation of a command from the client. The remainder of this response is a line of text.

This response is used in the AUTHORIZATION command to transmit server data to the client, and request additional client data. This response is also used if an argument to any command is a literal.

The client is not permitted to send the octets of the literal unless the server indicates that it expects it. This permits the server to process commands and reject errors on a line-by-line basis. The remainder of the command, including the CRLF that terminates a command, follows the octets of the literal. If there are any additional command arguments the literal octets are followed by a space and those arguments.

```
Example:    C: A001 LOGIN {11}
            S: + Ready for additional command text
            C: FRED FOOBAR {7}
            S: + Ready for additional command text
            C: fat man
            S: A001 OK LOGIN completed
            C: A044 BLURDYBLOOP {102856}
            S: A044 BAD No such command as "BLURDYBLOOP"
```

8. Sample IMAP4 session

The following is a transcript of an IMAP4 session. A long line in this sample is broken for editorial clarity.

```
S:   * OK IMAP4 Service Ready
C:   a001 login mrc secret
S:   a001 OK LOGIN completed
C:   a002 select inbox
S:   * 18 EXISTS
S:   * FLAGS (\Answered \Flagged \Deleted \Seen \Draft)
S:   * 2 RECENT
S:   * OK [UNSEEN 17] Message 17 is the first unseen message
S:   * OK [UIDVALIDITY 3857529045] UIDs valid
S:   a002 OK [READ-WRITE] SELECT completed
C:   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)"
      "IMAP4 WG mtg summary and minutes"
      (("Terry Gray" NIL "gray" "cac.washington.edu"))
      (("Terry Gray" NIL "gray" "cac.washington.edu"))
      (("Terry Gray" NIL "gray" "cac.washington.edu"))
      ((NIL NIL "imap" "cac.washington.edu"))
      ((NIL NIL "minutes" "CNRI.Reston.VA.US")
      ("John Klensin" NIL "KLENSIN" "INFOODS.MIT.EDU")) NIL NIL
      "<B27397-0100000@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 {346}
S:   Date: Wed, 14 Jul 1993 02:23:25 -0700 (PDT)
S:   From: Terry Gray <gray@cac.washington.edu>
S:   Subject: IMAP4 WG mtg summary and minutes
S:   To: imap@cac.washington.edu
S:   cc: minutes@CNRI.Reston.VA.US, John Klensin <KLENSIN@INFOODS.MIT.ED
U>
S:   Message-Id: <B27397-0100000@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 IMAP4 server terminating connection
S:   a006 OK LOGOUT completed
```

9. 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 are case-insensitive. The use of upper or lower case characters to define token strings is for editorial clarity only. Implementations MUST accept these strings in a case-insensitive fashion.

Syntax marked as obsolete may be encountered with implementations written for an earlier version of this protocol (e.g. IMAP2). New implementations SHOULD accept obsolete syntax as input, but MUST NOT otherwise use such syntax.

```
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

alpha           ::= "A" / "B" / "C" / "D" / "E" / "F" / "G" / "H" /
                    "I" / "J" / "K" / "L" / "M" / "N" / "O" / "P" /
                    "Q" / "R" / "S" / "T" / "U" / "V" / "W" / "X" /
                    "Y" / "Z" /
                    "a" / "b" / "c" / "d" / "e" / "f" / "g" / "h" /
                    "i" / "j" / "k" / "l" / "m" / "n" / "o" / "p" /
                    "q" / "r" / "s" / "t" / "u" / "v" / "w" / "x" /
                    "y" / "z" /
                    ;; Case-sensitive

append          ::= "APPEND" SPACE mailbox [SPACE flag_list]
                    [SPACE date_time] SPACE literal

astring         ::= atom / string
```

```
   atom            ::= 1*ATOM_CHAR

   ATOM_CHAR       ::= <any CHAR except atom_specials>

   atom_specials   ::= "(" / ")" / "{" / SPACE / CTLs / list_wildcards /
                       quoted_specials

   authenticate    ::= "AUTHENTICATE" SPACE auth_type *(CRLF base64)

   auth_type       ::= atom

   base64          ::= *(4base64_char) [base64_terminal]

   base64_char     ::= alpha / digit / "+" / "/"

   base64_terminal ::= (2base64_char "==") / (3base64_char "=")

   body            ::= "(" body_type_1part / body_type_mpart ")"

   body_extension  ::= nstring / number / "(" 1#body_extension ")"
                       ;; Future expansion.  Client implementations
                       ;; MUST accept body_extension fields.  Server
                       ;; implementations MUST NOT generate
                       ;; body_extension fields except as defined by
                       ;; future standard or standards-track
                       ;; revisions of this specification.

   body_ext_1part  ::= body_fld_md5 [SPACE 1#body_extension]
                       ;; MUST NOT be returned on non-extensible
                       ;; "BODY" fetch

   body_ext_mpart  ::= body_fld_param [SPACE 1#body_extension]]
                       ;; MUST NOT be returned on non-extensible
                       ;; "BODY" fetch

   body_fields     ::= body_fld_param SPACE body_fld_id SPACE
                       body_fld_desc SPACE body_fld_enc SPACE
                       body_fld_octets

   body_fld_desc   ::= nstring

   body_fld_enc    ::= (<"> ("7BIT" / "8BIT" / "BINARY" / "BASE64"/
                       "QUOTED-PRINTABLE") <">) / string

   body_fld_id     ::= nstring

   body_fld_lines  ::= number
```

```
   body_fld_md5    ::= nstring

   body_fld_octets ::= number

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

   body_fld_subtyp ::= string

   body_type_1part ::= (body_type_basic / body_type_msg / body_type_text)
                       [SPACE body_ext_1part]

   body_type_basic ::= (<"> ("APPLICATION" / "AUDIO" / "IMAGE" /
                       "MESSAGE" / "VIDEO") <">) / string) SPACE
                       body_fld_subtyp SPACE body_fields
                       ;; MESSAGE subtype MUST NOT be "RFC822"

   body_type_mpart ::= 1*body SPACE body_fld_subtyp
                       [SPACE body_ext_mpart]

   body_type_msg   ::= <"> "MESSAGE" <"> SPACE <"> "RFC822" <"> SPACE
                       body_fields SPACE envelope SPACE body SPACE
                       body_fld_lines

   body_type_text  ::= <"> "TEXT" <"> SPACE body_fld_subtyp SPACE
                        body_fields SPACE body_fld_lines

   capability      ::= atom
                       ;; Must begin with "X" or be registered with
                       ;; IANA as standard or standards-track

   capability_data ::= "CAPABILITY" SPACE "IMAP4" [SPACE 1#capability]

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

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

   command         ::= tag SPACE (command_any / command_auth /
                       command_nonauth / command_select) CRLF
                       ;; Modal based on state

   command_any     ::= "CAPABILITY" / "LOGOUT" / "NOOP" / x_command
                       ;; Valid in all states

   command_auth    ::= append / create / delete / examine / find / list /
                       lsub / rename / select / subscribe / unsubscribe /
                       ;; Valid only in Authenticated or Selected state
```

```
   command_nonauth ::= login / authenticate
                       ;; Valid only when in Non-Authenticated state

   command_select  ::= "CHECK" / "CLOSE" / "EXPUNGE" /
                        copy / fetch / partial / store / uid / search
                       ;; Valid only when in Selected state

   continue_req    ::= "+" SPACE (resp_text / base64)

   copy            ::= "COPY" SPACE set SPACE mailbox

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

   create          ::= "CREATE" SPACE mailbox
                       ;; Use of INBOX gives a NO error

   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 digit) / 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
                       ;; Use of INBOX gives a NO error

   digit           ::= "0" / digit_nz

   digit_nz        ::= "1" / "2" / "3" / "4" / "5" / "6" / "7" / "8" /
                       "9"

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

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

   env_date        ::= nstring

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

   env_in-reply-to ::= nstring

   env_message-id  ::= nstring

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

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

   env_subject     ::= nstring

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

   examine         ::= "EXAMINE" SPACE mailbox

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

   fetch_att       ::= "BODY" / "BODYSTRUCTURE" /
                       "BODY" [".PEEK"] "[" section "]" / "ENVELOPE" /
                       "FLAGS" / "INTERNALDATE" / "UID" /
                       "RFC822" (([".TEXT"] [".PEEK"]) / ".SIZE" /
                       (".HEADER" [".LINES" [".NOT"] SPACE header_list])

   find            ::= "FIND" SPACE ["ALL."] "MAILBOXES" SPACE
                       list_mailbox
                       ;; OBSOLETE
```

```
   flag            ::= "\Answered" / "\Flagged" / "\Deleted" /
                       "\Seen" / "\Draft" / flag_keyword  /
                       flag_extension

   flag_extension  ::= "\" atom
                       ;; Future expansion.  Client implementations
                       ;; MUST accept flag_extension flags.  Server
                       ;; implementations MUST NOT generate
                       ;; flag_extension flags except as defined by
                       ;; future standard or standards-track
                       ;; revisions of this specification.

   flag_keyword    ::= atom

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

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

   header_line     ::= astring

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

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

   list            ::= "LIST" SPACE mailbox SPACE list_mailbox

   list_mailbox    ::= 1*(ATOM_CHAR / list_wildcards) / string

   list_wildcards  ::= "%" / "*"

   literal         ::= "{" number "}" CRLF *CHAR8
                       ;; Number represents the number of CHAR8 octets

   login           ::= "LOGIN" SPACE userid SPACE password

   lsub            ::= "LSUB" SPACE mailbox SPACE list_mailbox

   mailbox         ::= "INBOX" / astring
                       ;; INBOX is case-insensitive; other names may be
                       ;; case-sensitive depending on implementation.

   mailbox_data    ::=  "FLAGS" SPACE flag_list /
                        "LIST" SPACE mailbox_list /
                        "LSUB" SPACE mailbox_list /
                        "MAILBOX" SPACE text /
                        "SEARCH" [SPACE 1#nz_number] /
                        number SPACE "EXISTS" / number SPACE "RECENT"
```

```
   mailbox_list    ::= "(" #("\Marked" / "\Noinferiors" /
                       "\Noselect" / "\Unmarked" / flag_extension) ")"
                       SPACE (<"> QUOTED_CHAR <"> / nil) SPACE mailbox

   message_data    ::= nz_number SPACE ("EXPUNGE" /
                       ("FETCH" SPACE msg_fetch) / msg_obsolete)

   msg_fetch       ::= "(" 1#("BODY" SPACE body /
                       "BODYSTRUCTURE" SPACE body /
                       "BODY[" section "]" SPACE nstring /
                       "ENVELOPE" SPACE envelope /
                       "FLAGS" SPACE "(" #(flag / "\Recent") ")" /
                       "INTERNALDATE" SPACE date_time /
                       "RFC822" [".HEADER" / ".TEXT"] SPACE nstring /
                       "RFC822.SIZE" SPACE number /
                       "UID" SPACE uniqueid) ")"

   msg_obsolete    ::= "COPY" / ("STORE" SPACE msg_fetch)
                       ;; OBSOLETE untagged data responses

   nil             ::= "NIL"

   nstring         ::= string / nil

   number          ::= 1*digit
                       ;; Unsigned 32-bit integer
                       ;; (0 <= n < 4,294,967,296)

   nz_number       ::= digit_nz *digit
                       ;; Non-zero unsigned 32-bit integer
                       ;; (0 < n < 4,294,967,296)

   partial         ::= "PARTIAL" SPACE nz_number SPACE
                       ("BODY" [".PEEK"] "[" section "]" /
                       "RFC822" (([".TEXT"] [".PEEK"]) / ".HEADER")
                       SPACE number SPACE number

   password        ::= astring

   quoted          ::= <"> *QUOTED_CHAR <">

   QUOTED_CHAR     ::= <any TEXT_CHAR except quoted_specials> /
                       "\" quoted_specials

   quoted_specials ::= <"> / "\"

   rename          ::= "RENAME" SPACE mailbox SPACE mailbox
                       ;; Use of INBOX as a destination gives a NO error
```

```
   response        ::= *response_data response_done

   response_data   ::= "*" SPACE (resp_cond_state / resp_cond_bye /
                       mailbox_data / message_data / capability_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_text_code "]" SPACE] (text_mime2 / text)

   resp_text_code  ::= "ALERT" / "PARSE" /
                       "PERMANENTFLAGS" SPACE "(" #(flag / "\*") ")" /
                       "READ-ONLY" / "READ-WRITE" / "TRYCREATE" /
                       "UIDVALIDITY" SPACE nz_number /
                       "UNSEEN" SPACE nz_number /
                       atom [SPACE 1*<any TEXT_CHAR except "]">]

   search          ::= "SEARCH" SPACE ["CHARSET" SPACE astring SPACE]
                       search_criteria
                       ;; Character set must be registered with IANA
                       ;; as a MIME character set

   search_criteria ::= 1#search_key

   search_key      ::= search_new / search_old

   search_new      ::= "DRAFT" /
                       "HEADER" SPACE header_line SPACE astring /
                       "LARGER" SPACE number / "NOT" SPACE search_key /
                       "OR" SPACE search_key SPACE search_key /
                       "SENTBEFORE" SPACE date / "SENTON" SPACE date /
                       "SENTSINCE" SPACE date / "SMALLER" SPACE number /
                       "UID" SPACE set / "UNDRAFT" / set /
                       "(" search_criteria ")"
                       ;; New in IMAP4
```

```
   search_old      ::= "ALL" / "ANSWERED" / "BCC" SPACE astring /
                       "BEFORE" SPACE date / "BODY" SPACE astring /
                       "CC" SPACE astring / "DELETED" / "FLAGGED" /
                       "FROM" SPACE astring /
                       "KEYWORD" SPACE flag_keyword / "NEW" / "OLD" /
                       "ON" SPACE date / "RECENT" / "SEEN" /
                       "SINCE" SPACE date / "SUBJECT" SPACE astring /
                       "TEXT" SPACE astring / "TO" SPACE astring /
                       "UNANSWERED" / "UNDELETED" / "UNFLAGGED" /
                       "UNKEYWORD" SPACE flag_keyword / "UNSEEN"
                       ;; Defined in [IMAP2]

   section         ::= "0" / nz_number ["." section]

   select          ::= "SELECT" SPACE mailbox

   sequence_num    ::= nz_number / "*"
                       ;; * is the largest number in use.  For message
                       ;; sequence numbers, it is the number of messages
                       ;; in the mailbox.  For unique identifiers, it is
                       ;; the unique identifier of the last message in
                       ;; the mailbox.

   set             ::= sequence_num / (sequence_num ":" sequence_num) /
                       (set "," set)
                       ;; Identifies a set of messages.  For message
                       ;; sequence numbers, these are 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:* is 2,4,5,6,7,9,12,13,
                       ;; 14,15 for a mailbox with 15 messages.

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

   store           ::= "STORE" SPACE set SPACE store_att_flags

   store_att_flags ::= (["+" / "-"] "FLAGS" [".SILENT"]) SPACE
                       (flag_list / #flag)

   string          ::= quoted / literal

   subscribe       ::= ("SUBSCRIBE" SPACE mailbox) / subscribe_obs

   subscribe_obs   ::= "SUBSCRIBE" SPACE "MAILBOX" SPACE mailbox
                       ;;OBSOLETE
```

```
   tag             ::= 1*<any ATOM_CHAR except "+">

   text            ::= 1*TEXT_CHAR

   text_mime2       ::= "=?" <charset> "?" <encoding> "?"
                        <encoded-text> "?="
                        ;; Syntax defined in [MIME-2]

   TEXT_CHAR       ::= <any CHAR except CR and LF>

   time            ::= 2digit ":" 2digit ":" 2digit
                       ;; Hours minutes seconds

   uid             ::= "UID" SPACE (copy / fetch / search / store)
                       ;; Unique identifiers used instead of message
                       ;; sequence numbers

   uniqueid        ::= nz_number
                       ;; Strictly ascending

   unsubscribe     ::= ("UNSUBSCRIBE" SPACE mailbox) / unsubscribe_obs

   unsubscribe_obs ::= "UNSUBSCRIBE" SPACE "MAILBOX" SPACE mailbox
                       ;;OBSOLETE

   userid          ::= astring

   x_command       ::= "X" atom <experimental command arguments>

   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" / "EDT" /                     ;; -0400
                       "EST" / "CDT" /                     ;; -0500
                       "CST" / "MDT" /                     ;; -0600
                       "MST" / "PDT" /                     ;; -0700
                       "PST" / "YDT" /                     ;; -0800
                       "YST" / "HDT" /                     ;; -0900
                       "HST" / "BDT" /                     ;; -1000
                       "BST" /                             ;; -1100
                       "A" / "B" / "C" / "D" / "E" / "F" / ;; +1 to +6
                       "G" / "H" / "I" / "K" / "L" / "M" / ;; +7 to +12
                       "N" / "O" / "P" / "Q" / "R" / "S" / ;; -1 to -6
                       "T" / "U" / "V" / "W" / "X" / "Y"   ;; -7 to -12
                       ;; OBSOLETE
```

10. Author's Note

This document is a revision or rewrite of earlier documents, and supercedes the protocol specification in those documents: IMAP4 Internet drafts, the IMAP2bis Internet drafts, unpublished IMAP2bis.TXT document, RFC 1176, and RFC 1064.

11. Security Considerations

IMAP4 protocol transactions, including electronic mail data, are sent in the clear over the network unless the optional privacy protection is negotiated in the AUTHENTICATE command.

A server error message for an AUTHENTICATE command which fails due to invalid credentials should not detail why the credentials are invalid.

Use of the LOGIN command sends passwords in the clear. This can be avoided by using the AUTHENTICATE command instead.

A server error message for a failing LOGIN command should not specify that the user name, as opposed to the password, is invalid.

Additional security considerations are discussed in the section discussing the AUTHENTICATE and LOGIN commands.

12. 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

Appendices

A. Obsolete Commands

The following commands are OBSOLETE. It is NOT required to support any of these commands in new server implementations. These commands are documented here for the benefit of implementors who may wish to support them for compatibility with old client implementations.

The section headings of these commands are intended to correspond with where they would be located in the main document if they were not obsoleted.

A.6.3.OBS.1. FIND ALL.MAILBOXES Command

Arguments: mailbox name with possible wildcards

Data: untagged responses: MAILBOX

Result: OK - find completed
NO - find failure: can't list that name
BAD - command unknown or arguments invalid

The FIND ALL.MAILBOXES command returns a subset of names from the complete set of all names available to the user. It returns zero or more untagged MAILBOX replies. The mailbox argument to FIND ALL.MAILBOXES is similar to that for LIST with an empty reference, except that the characters "%" and "?" match a single character.

Example: C: A002 FIND ALL.MAILBOXES *
S: * MAILBOX blurdybloop
S: * MAILBOX INBOX
S: A002 OK FIND ALL.MAILBOXES completed

A.6.3.OBS.2. FIND MAILBOXES Command

Arguments: mailbox name with possible wildcards

Data: untagged responses: MAILBOX

Result: OK - find completed
NO - find failure: can't list that name
BAD - command unknown or arguments invalid

The FIND MAILBOXES command returns a subset of names from the set of names that the user has declared as being "active" or

"subscribed". It returns zero or more untagged MAILBOX replies. The mailbox argument to FIND MAILBOXES is similar to that for LSUB with an empty reference, except that the characters "%" and "?" match a single character.

Example: C: A002 FIND MAILBOXES *
S: * MAILBOX blurdybloop
S: * MAILBOX INBOX
S: A002 OK FIND MAILBOXES completed

A.6.3.OBS.3. SUBSCRIBE MAILBOX Command

Arguments: mailbox name

Data: no specific data for this command

Result: OK - subscribe completed
NO - subscribe failure: can't subscribe to that name
BAD - command unknown or arguments invalid

The SUBSCRIBE MAILBOX command is identical in effect to the SUBSCRIBE command. A server which implements this command must be able to distinguish between a SUBSCRIBE MAILBOX command and a SUBSCRIBE command with a mailbox name argument of "MAILBOX".

Example: C: A002 SUBSCRIBE MAILBOX #news.comp.mail.mime
S: A002 OK SUBSCRIBE MAILBOX to #news.comp.mail.mime completed
C: A003 SUBSCRIBE MAILBOX
S: A003 OK SUBSCRIBE to MAILBOX completed

A.6.3.OBS.4. UNSUBSCRIBE MAILBOX Command

Arguments: mailbox name

Data: no specific data for this command

Result: OK - unsubscribe completed
NO - unsubscribe failure: can't unsubscribe that name
BAD - command unknown or arguments invalid

The UNSUBSCRIBE MAILBOX command is identical in effect to the UNSUBSCRIBE command. A server which implements this command must be able to distinguish between a UNSUBSCRIBE MAILBOX command and an UNSUBSCRIBE command with a mailbox name argument of "MAILBOX".

```
   Example:    C: A002 UNSUBSCRIBE MAILBOX #news.comp.mail.mime
               S: A002 OK UNSUBSCRIBE MAILBOX from #news.comp.mail.mime
               completed
               C: A003 UNSUBSCRIBE MAILBOX
               S: A003 OK UNSUBSCRIBE from MAILBOX completed
```

B. Obsolete Responses

The following responses are OBSOLETE. Except as noted below, these responses MUST NOT be transmitted by new server implementations.

The section headings of these responses are intended to correspond with where they would be located in the main document if they were not obsoleted.

B.7.2.OBS.1. MAILBOX Response

Data: name

The MAILBOX response MUST NOT be transmitted by server implementations except in response to the obsolete FIND MAILBOXES and FIND ALL.MAILBOXES commands. Client implementations that do not use these commands MAY ignore this response. It is documented here for the benefit of implementors who may wish to support it for compatibility with old client implementations.

This response occurs as a result of the FIND MAILBOXES and FIND ALL.MAILBOXES commands. It returns a single name that matches the FIND specification. There are no attributes or hierarchy delimiter.

Example: S: * MAILBOX blurdybloop

B.7.3.OBS.1. COPY Response

Data: none

The COPY response MUST NOT be transmitted by new server implementations. Client implementations MUST ignore the COPY response. It is documented here for the benefit of client implementors who may encounter this response from old server implementations.

In some experimental versions of this protocol, this response was returned in response to a COPY command to indicate on a per-message basis that the message was copied successfully.

Example: S: * 44 COPY

B.7.3.OBS.2. STORE Response

Data: message data

The STORE response MUST NOT be transmitted by new server implementations. Client implementations MUST treat the STORE response as equivalent to the FETCH response. It is documented here for the benefit of client implementors who may encounter this response from old server implementations.

In some experimental versions of this protocol, this response was returned instead of FETCH in response to a STORE command to report the new value of the flags.

Example: S: * 69 STORE (FLAGS (\Deleted))

C. References

[IMAP-AUTH] Myers, J., "IMAP4 Authentication Mechanism", RFC 1731. Carnegie-Mellon University, December 1994.

[IMAP-COMPAT] Crispin, M. "IMAP4 Compatibility with IMAP2 and IMAP2bis", RFC 1732, University of Washington, December 1994.

[IMAP-DISC] Austein, R. "Synchronization Operations for Disconnected IMAP4 Clients", Work in Progress.

[IMAP-MODEL] Crispin, M. "Distributed Electronic Mail Models in IMAP4", RFC 1733, University of Washington, December 1994.

[IMAP-NAMING] Crispin, M. "Mailbox Naming Convention in IMAP4", Work in Progress.

[IMAP2] Crispin, M., "Interactive Mail Access Protocol - Version 2", RFC 1176, University of Washington, August 1990.

[IMSP] Myers, J. "IMSP -- Internet Message Support Protocol", Work in Progress.

[MIME-1] Borenstein, N., and Freed, N., "MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies", RFC 1521, Bellcore, Innosoft, September 1993.

[MIME-2] Moore, K., "MIME (Multipurpose Internet Mail Extensions) Part Two: Message Header Extensions for Non-ASCII Text", RFC 1522, University of Tennessee, September 1993.

[RFC-822] Crocker, D., "Standard for the Format of ARPA Internet Text Messages", STD 11, RFC 822, University of Delaware, August 1982.

[SMTP] Postel, Jonathan B. "Simple Mail Transfer Protocol", STD 10, RFC 821, USC/Information Sciences Institute, August 1982.

E. IMAP4 Keyword Index