Network Working Group
Request for Comments: 2060
Obsoletes: 1730
Category: Standards Track

M. Crispin
University of Washington

December 1996

INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1

Status of this Memo

Abstract

The Internet Message Access Protocol, Version 4rev1 (IMAP4rev1) allows a client to access and manipulate electronic mail messages on a server. IMAP4rev1 permits manipulation of remote message folders, called "mailboxes", in a way that is functionally equivalent to local mailboxes. IMAP4rev1 also provides the capability for an offline client to resynchronize with the server (see also [IMAP-DISC]).

IMAP4rev1 includes operations for creating, deleting, and renaming mailboxes; checking for new messages; permanently removing messages; setting and clearing flags; [RFC-822] and [MIME-IMB] parsing; searching; and selective fetching of message attributes, texts, and portions thereof. Messages in IMAP4rev1 are accessed by the use of numbers. These numbers are either message sequence numbers or unique identifiers.

IMAP4rev1 supports a single server. A mechanism for accessing configuration information to support multiple IMAP4rev1 servers is discussed in [ACAP].

IMAP4rev1 does not specify a means of posting mail; this function is handled by a mail transfer protocol such as [SMTP].

IMAP4rev1 is designed to be upwards compatible from the [IMAP2] and unpublished IMAP2bis protocols. In the course of the evolution of IMAP4rev1, some aspects in the earlier protocol have become obsolete. Obsolete commands, responses, and data formats which an IMAP4rev1 implementation may encounter when used with an earlier implementation are described in [IMAP-OBSOLETE].

Other compatibility issues with IMAP2bis, the most common variant of the earlier protocol, are discussed in [IMAP-COMPAT]. A full discussion of compatibility issues with rare (and presumed extinct) variants of [IMAP2] is in [IMAP-HISTORICAL]; this document is primarily of historical interest.

Table of Contents

IMAP4rev1 Protocol Specification

1. How to Read This Document

1.1. Organization of This Document

This document is written from the point of view of the implementor of an IMAP4rev1 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 IMAP4rev1 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, do not attempt to deduce command syntax from the command section alone; 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.

The following terms are used in this document to signify the requirements of this specification.

1) MUST, or the adjective REQUIRED, means that the definition is an absolute requirement of the specification.

2) MUST NOT that the definition is an absolute prohibition of the specification.

3) SHOULD means that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications MUST be understood and carefully weighed before choosing a different course.

4) SHOULD NOT means that there may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications SHOULD be understood and the case carefully weighed before implementing any behavior described with this label.

5) MAY, or the adjective OPTIONAL, means that an item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because the vendor feels that it enhances the product while another vendor may omit the same item. An implementation which does not include a particular option MUST be prepared to interoperate with another implementation which does include the option.

"Can" is used instead of "may" when referring to a possible circumstance or situation, as opposed to an optional facility of the protocol.

"User" is used to refer to a human user, whereas "client" refers to the software being run by the user.

"Connection" refers to the entire sequence of client/server interaction from the initial establishment of the network connection until its termination. "Session" refers to the sequence of client/server interaction from the time that a mailbox is selected (SELECT or EXAMINE command) until the time that selection ends (SELECT or EXAMINE of another mailbox, CLOSE command, or connection termination).

Characters are 7-bit US-ASCII unless otherwise specified. Other character sets are indicated using a "CHARSET", as described in [MIME-IMT] and defined in [CHARSET]. CHARSETs have important additional semantics in addition to defining character set; refer to these documents for more detail.

2. Protocol Overview

2.1. Link Level

The IMAP4rev1 protocol assumes a reliable data stream such as provided by TCP. When TCP is used, an IMAP4rev1 server listens on port 143.

2.2. Commands and Responses

An IMAP4rev1 connection consists of the establishment of a client/server network 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 IMAP4rev1 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 an 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. In all cases, the client MUST send a complete command (including receiving all command continuation request responses and command continuations for the command) before initiating a new command.

The protocol receiver of an IMAP4rev1 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 IMAP4rev1 client reads a response line from the server. It then takes action on the response based upon the first token of the response, which can be a tag, a "*", or a "+".

A client MUST be prepared to accept any server response at all times. This includes server data that was not 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, the data MUST be recorded.

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

2.3. Message Attributes

In addition to message text, each message has several attributes associated with it. These attributes may be retrieved individually or in conjunction with other attributes or message texts.

2.3.1. Message Numbers

Messages in IMAP4rev1 are accessed by one of two numbers; the unique identifier and the message sequence number.

2.3.1.1. Unique Identifier (UID) Message Attribute

A 32-bit value assigned to each message, which when used with the unique identifier validity value (see below) forms a 64-bit value

that is permanently guaranteed not to refer to any other message in the mailbox. Unique identifiers are assigned in a strictly ascending fashion in the mailbox; as each message is added to the mailbox it is assigned a higher UID than the message(s) which were added previously.

Unlike message sequence numbers, unique identifiers are not necessarily contiguous. Unique identifiers also 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 mailbox selection time. If unique identifiers from an earlier session fail to persist to this session, the unique identifier validity value MUST be greater than the one used in the earlier session.

Note: Unique identifiers MUST be strictly ascending in the mailbox at all times. If the physical message store is re-ordered by a non-IMAP agent, this requires that the unique identifiers in the mailbox be regenerated, since the former unique identifers are no longer strictly ascending as a result of the re-ordering. Another instance in which unique identifiers are regenerated is if the message store has no mechanism to store unique identifiers. Although this specification recognizes that this may be unavoidable in certain server environments, it STRONGLY ENCOURAGES message store implementation techniques that avoid this problem.

Another cause of non-persistance is if the mailbox is deleted and a new mailbox with the same name is created at a later date, Since the name is the same, a client may not know that this is a new mailbox unless the unique identifier validity is different. A good value to use for the unique identifier validity value is 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 identifiers will never be reused, even in the case of a mailbox being deleted (or renamed) and a new mailbox by the same name created at some future time.

The unique identifier of a message MUST NOT change during the session, and SHOULD NOT change between sessions. However, if it is not possible to preserve the unique identifier of a message in a subsequent session, each subsequent session MUST have a new unique identifier validity value that is larger than any that was used previously.

2.3.1.2. Message Sequence Number Message Attribute

A relative position from 1 to the number of messages in the mailbox. This position MUST be ordered by ascending unique identifier. As each new message is added, it is assigned a message sequence number that is 1 higher than the number of messages in the mailbox before that new message was added.

Message sequence numbers can be reassigned during the session. For example, when a message is permanently removed (expunged) from the mailbox, the message sequence number for all subsequent messages is decremented. Similarly, a new message can be assigned a message sequence number that was once held by some other message prior to an expunge.

In addition to accessing messages by relative position in the mailbox, message sequence numbers can be used in mathematical calculations. For example, if an untagged "EXISTS 11" is received, and previously an untagged "8 EXISTS" was received, three new messages have arrived with message sequence numbers of 9, 10, and 11. Another example; if message 287 in a 523 message mailbox has UID 12345, there are exactly 286 messages which have lesser UIDs and 236 messages which have greater UIDs.

2.3.2. Flags Message Attribute

A list of zero or more named tokens associated with the message. A flag is set by its addition to this list, and is cleared by its removal. There are two types of flags in IMAP4rev1. A flag of either type may be permanent or session-only.

A system flag is a flag name that is pre-defined in this specification. All system flags begin with "\". Certain system flags (\Deleted and \Seen) have special semantics described elsewhere. The currently-defined system flags are:

\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).

\Recent Message is "recently" arrived in this mailbox. This session is the first session to have been notified about this message; subsequent sessions will not see \Recent set for this message. This flag can not be altered by the client.

If it is not possible to determine whether or not this session is the first session to be notified about a message, then that message SHOULD be considered recent.

If multiple connections have the same mailbox selected simultaneously, it is undefined which of these connections will see newly-arrives messages with \Recent set and which will see it without \Recent set.

A keyword is defined by the server implementation. Keywords do not begin with "\". Servers MAY permit the client to define new keywords in the mailbox (see the description of the PERMANENTFLAGS response code for more information).

A flag may be permanent or session-only on a per-flag basis. Permanent flags are those which the client can add or remove from the message flags permanently; that is, subsequent sessions will see any change in permanent flags. Changes to session flags are valid only in that session.

Note: The \Recent system flag is a special case of a session flag. \Recent can not be used as an argument in a STORE command, and thus can not be changed at all.

2.3.3. Internal Date Message Attribute

The internal date and time of the message on the server. This is not the date and time in the [RFC-822] header, but rather a date and time which reflects when the message was received. In the case of messages delivered via [SMTP], this SHOULD be the date and time of final delivery of the message as defined by [SMTP]. In the case of messages delivered by the IMAP4rev1 COPY command, this SHOULD be the internal date and time of the source message. In the case of messages delivered by the IMAP4rev1 APPEND command, this SHOULD be the date and time as specified in the APPEND command description. All other cases are implementation defined.

2.3.4. [RFC-822] Size Message Attribute

The number of octets in the message, as expressed in [RFC-822] format.

2.3.5. Envelope Structure Message Attribute

A parsed representation of the [RFC-822] envelope information (not to be confused with an [SMTP] envelope) of the message.

2.3.6. Body Structure Message Attribute

A parsed representation of the [MIME-IMB] body structure information of the message.

2.4. Message Texts

In addition to being able to fetch the full [RFC-822] text of a message, IMAP4rev1 permits the fetching of portions of the full message text. Specifically, it is possible to fetch the [RFC-822] message header, [RFC-822] message body, a [MIME-IMB] body part, or a [MIME-IMB] header.

3. State and Flow Diagram

An IMAP4rev1 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 client 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

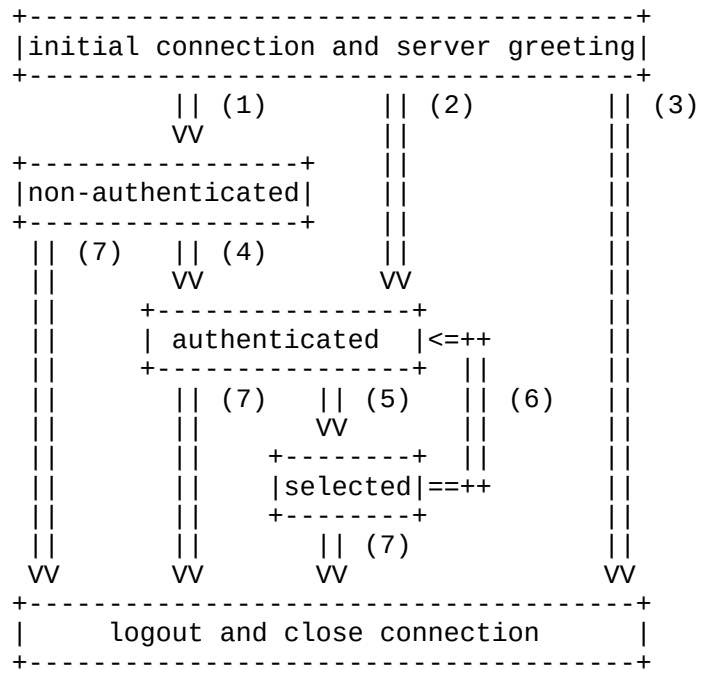
In authenticated state, the client is authenticated and MUST select a mailbox to access before commands that affect messages will be permitted. This state is entered when a pre-authenticated connection starts, when acceptable authentication credentials have been provided, or after an error in selecting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

IMAP4rev1 uses textual commands and responses. Data in IMAP4rev1 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 limitations of characters that can be used 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 represented 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 a [MIME-IMB] content transfer encoding. IMAP4rev1 implementations MAY transmit 8-bit or multi-octet characters in literals, but SHOULD do so only when the [CHARSET] 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.

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 can 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 case-insensitive mailbox name INBOX is a special name reserved to mean "the primary mailbox for this user on this server".

5.1.1. Mailbox Hierarchy Naming

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.

5.1.2. Mailbox Namespace Naming Convention

By convention, the first hierarchical element of any mailbox name which begins with "#" identifies the "namespace" of the remainder of the name. This makes it possible to disambiguate between different types of mailbox stores, each of which have their own namespaces.

For example, implementations which offer access to USENET newsgroups MAY use the "#news" namespace to partition the USENET newsgroup namespace from that of other mailboxes. Thus, the comp.mail.misc newsgroup would have an mailbox name of "#news.comp.mail.misc", and the name "comp.mail.misc" could refer to a different object (e.g. a user's private mailbox).

5.1.3. Mailbox International Naming Convention

By convention, international mailbox names are specified using a modified version of the UTF-7 encoding described in [UTF-7]. The purpose of these modifications is to correct the following problems with UTF-7:

1) UTF-7 uses the "+" character for shifting; this conflicts with the common use of "+" in mailbox names, in particular USENET newsgroup names.

2) UTF-7's encoding is BASE64 which uses the "/" character; this conflicts with the use of "/" as a popular hierarchy delimiter.

3) UTF-7 prohibits the unencoded usage of "\"; this conflicts with the use of "\" as a popular hierarchy delimiter.

4) UTF-7 prohibits the unencoded usage of "~"; this conflicts with the use of "~" in some servers as a home directory indicator.

5) UTF-7 permits multiple alternate forms to represent the same string; in particular, printable US-ASCII chararacters can be represented in encoded form.

In modified UTF-7, printable US-ASCII characters except for "&" represent themselves; that is, characters with octet values 0x20-0x25 and 0x27-0x7e. The character "&" (0x26) is represented by the two-octet sequence "&-".

All other characters (octet values 0x00-0x1f, 0x7f-0xff, and all Unicode 16-bit octets) are represented in modified BASE64, with a further modification from [UTF-7] that "," is used instead of "/". Modified BASE64 MUST NOT be used to represent any printing US-ASCII character which can represent itself.

"&" is used to shift to modified BASE64 and "-" to shift back to US-ASCII. All names start in US-ASCII, and MUST end in US-ASCII (that is, a name that ends with a Unicode 16-bit octet MUST end with a "-").

For example, here is a mailbox name which mixes English, Japanese, and Chinese text: ~peter/mail/&ZeVnLIqe-/&U,BTFw-

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 detail.

Regardless of what implementation decisions a client makes 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 MAY send another command without waiting for the completion result response of a command, subject to ambiguity rules (see below) and flow control constraints on the underlying data stream. Similarly, a server MAY begin processing another command before processing the current command to completion, subject to ambiguity rules. However, any command continuation request responses and command continuations MUST be negotiated before any subsequent command is initiated.

The exception is if an ambiguity would result because of a command that would affect the results of other commands. Clients MUST NOT send multiple commands without waiting if an ambiguity would result. If the server detects a possible ambiguity, it MUST execute commands to completion in the order given by the client.

The most obvious example of ambiguity is when a command would affect the results of another command; for example, a FETCH of a message's flags and a STORE of that same message's flags.

A non-obvious ambiguity occurs with commands that permit an untagged EXPUNGE response (commands other than FETCH, STORE, and SEARCH), since an untagged EXPUNGE response can invalidate sequence numbers in a subsequent command. This is not a problem for FETCH, STORE, or SEARCH commands because servers are prohibited from sending EXPUNGE responses while any of those commands are in progress. Therefore, if the client sends any command other than FETCH, STORE, or SEARCH, it MUST wait for a response before sending a command with message sequence numbers.

For example, the following non-waiting command sequences are invalid:

```
FETCH + NOOP + STORE
STORE + COPY + FETCH
COPY + COPY
CHECK + FETCH
```

The following are examples of valid non-waiting command sequences:

```
FETCH + STORE + SEARCH + CHECK
STORE + COPY + EXPUNGE
```

6. Client Commands

IMAP4rev1 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 responses to be returned; these are identified by "Responses:" 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 responses 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

Responses: REQUIRED 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 "IMAP4rev1" as one of the listed capabilities 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 connection.

A capability name which begins with "AUTH=" indicates that the server supports that particular authentication mechanism. All such names are, by definition, part of this specification. For example, the authorization capability for an experimental "blurdybloop" authenticator would be "AUTH=XBLURDYBLOOP" and not "XAUTH=BLURDYBLOOP" or "XAUTH=XBLURDYBLOOP".

Other capability names refer to extensions, revisions, or amendments to this specification. See the documentation of the CAPABILITY response for additional information. No capabilities, beyond the base IMAP4rev1 set defined in this specification, 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 IMAP4rev1 AUTH=KERBEROS_V4
S: abcd OK CAPABILITY completed

6.1.2. NOOP Command

Arguments: none

Responses: no specific responses 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

Responses: REQUIRED 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 connection. 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 IMAP4rev1 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

Responses: continuation data can 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 client. It MAY also negotiate an OPTIONAL 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 issues 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 connection. 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.

Authentication mechanisms are OPTIONAL. Protection mechanisms are also OPTIONAL; an authentication mechanism MAY be implemented without any protection mechanism. 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 IMAP4rev1 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

Responses:  no specific responses 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 client 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, STATUS, and APPEND.

6.3.1. SELECT Command

Arguments: mailbox name

Responses: REQUIRED 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. See the description of the FLAGS response for more detail.

<n> EXISTS The number of messages in the mailbox. See the description of the EXISTS response for more detail.

<n> RECENT The number of messages with the \Recent flag set. See the description of the RECENT response for more detail.

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.

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 can change permanently.

Only one mailbox can be selected at a time in a connection; simultaneous access to multiple mailboxes requires multiple connections. 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 client 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 client 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 server-based .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

Responses: REQUIRED 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

Responses: no specific responses 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 intends to create mailbox names under this name in the hierarchy. Server implementations that do not require this declaration MUST ignore it.

If the server's hierarchy separator character appears elsewhere in the name, the server SHOULD create any superior hierarchical names that are needed for the CREATE command to complete successfully. In other words, an attempt to create "foo/bar/zap" on a server in which "/" is the hierarchy separator character SHOULD create foo/ and foo/bar/ if they do not already exist.

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

Responses: no specific responses 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.

The DELETE command MUST NOT remove inferior hierarchical names. For example, if a mailbox "foo" has an inferior "foo.bar" (assuming "." is the hierarchy delimiter character), removing "foo" MUST NOT remove "foo.bar". It is an error to attempt to delete a name that has inferior hierarchical names and also has the \Noselect mailbox name attribute (see the description of the LIST response for more details).

It is permitted to delete a name that has inferior hierarchical names and does not have the \Noselect mailbox name attribute. In this case, all messages in that mailbox are removed, and the name will acquire the \Noselect mailbox name attribute.

The value of the highest-used unique identifier 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.

```
   Examples:   C: A682 LIST "" *
               S: * LIST () "/" blurdybloop
               S: * LIST (\Noselect) "/" foo
               S: * LIST () "/" foo/bar
               S: A682 OK LIST completed
               C: A683 DELETE blurdybloop
               S: A683 OK DELETE completed
               C: A684 DELETE foo
               S: A684 NO Name "foo" has inferior hierarchical names
               C: A685 DELETE foo/bar
               S: A685 OK DELETE Completed
               C: A686 LIST "" *
               S: * LIST (\Noselect) "/" foo
               S: A686 OK LIST completed
               C: A687 DELETE foo
               S: A687 OK DELETE Completed


               C: A82 LIST "" *
               S: * LIST () "." blurdybloop
               S: * LIST () "." foo
               S: * LIST () "." foo.bar
               S: A82 OK LIST completed
               C: A83 DELETE blurdybloop
               S: A83 OK DELETE completed
               C: A84 DELETE foo
               S: A84 OK DELETE Completed
               C: A85 LIST "" *
               S: * LIST () "." foo.bar
               S: A85 OK LIST completed
               C: A86 LIST "" %
               S: * LIST (\Noselect) "." foo
               S: A86 OK LIST completed
```

6.3.5. RENAME Command

```
   Arguments:  existing mailbox name
               new mailbox name

   Responses:  no specific responses 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.

If the name has inferior hierarchical names, then the inferior hierarchical names MUST also be renamed. For example, a rename of "foo" to "zap" will rename "foo/bar" (assuming "/" is the hierarchy delimiter character) to "zap/bar".

The value of the highest-used unique identifier of the old mailbox name 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.

Renaming INBOX is permitted, and has special behavior. It moves all messages in INBOX to a new mailbox with the given name, leaving INBOX empty. If the server implementation supports inferior hierarchical names of INBOX, these are unaffected by a rename of INBOX.

Examples:
```
C: A682 LIST "" *
S: * LIST () "/" blurdybloop
S: * LIST (\Noselect) "/" foo
S: * LIST () "/" foo/bar
S: A682 OK LIST completed
C: A683 RENAME blurdybloop sarasoop
S: A683 OK RENAME completed
C: A684 RENAME foo zowie
S: A684 OK RENAME Completed
C: A685 LIST "" *
S: * LIST () "/" sarasoop
S: * LIST (\Noselect) "/" zowie
S: * LIST () "/" zowie/bar
S: A685 OK LIST completed
```

```
C: Z432 LIST "" *
S: * LIST () "." INBOX
S: * LIST () "." INBOX.bar
S: Z432 OK LIST completed
C: Z433 RENAME INBOX old-mail
S: Z433 OK RENAME completed
C: Z434 LIST "" *
S: * LIST () "." INBOX
S: * LIST () "." INBOX.bar
S: * LIST () "." old-mail
S: Z434 OK LIST completed
```

6.3.6. SUBSCRIBE Command

Arguments: mailbox

Responses: no specific responses 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.

A server MAY validate the mailbox argument to SUBSCRIBE to verify that it exists. However, it MUST NOT unilaterally remove an existing mailbox name from the subscription list even if a mailbox by that name no longer exists.

Note: this requirement is because some server sites may routinely remove a mailbox with a well-known name (e.g. "system-alerts") after its contents expire, with the intention of recreating it when new contents are appropriate.

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

6.3.7. UNSUBSCRIBE Command

Arguments: mailbox name

Responses: no specific responses 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

Responses: 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 client. 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.

The LIST command SHOULD return its data quickly, without undue delay. For example, it SHOULD NOT go to excess trouble to calculate \Marked or \Unmarked status or perform other processing; if each name requires 1 second of processing, then a list of 1200 names would take 20 minutes!

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.

An empty ("" string) mailbox name argument is a special request to return the hierarchy delimiter and the root name of the name given in the reference. The value returned as the root MAY be null if the reference is non-rooted or is null. In all cases, the hierarchy delimiter is returned. This permits a client to get the hierarchy delimiter even when no mailboxes by that name currently exist.

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 details).

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 INBOX is supported by this server for this user and if the uppercase string "INBOX" matches the interpreted reference and mailbox name arguments with wildcards as described above. 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: A101 LIST "" ""
S: * LIST (\Noselect) "/" ""
S: A101 OK LIST Completed
C: A102 LIST #news.comp.mail.misc ""
S: * LIST (\Noselect) "." #news.
S: A102 OK LIST Completed
C: A103 LIST /usr/staff/jones ""
S: * LIST (\Noselect) "/" /
S: A103 OK LIST Completed
C: A202 LIST ~/Mail/ %
S: * LIST (\Noselect) "/" ~/Mail/foo
S: * LIST () "/" ~/Mail/meetings
S: A202 OK LIST completed
```

6.3.9. LSUB Command

Arguments: reference name
mailbox name with possible wildcards

Responses: 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.

A server MAY validate the subscribed names to see if they still exist. If a name does not exist, it SHOULD be flagged with the \Noselect attribute in the LSUB response. The server MUST NOT

unilaterally remove an existing mailbox name from the subscription list even if a mailbox by that name no longer exists.

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. STATUS Command

Arguments: mailbox name
status data item names

Responses: untagged responses: STATUS

Result: OK - status completed
NO - status failure: no status for that name
BAD - command unknown or arguments invalid

The STATUS command requests the status of the indicated mailbox. It does not change the currently selected mailbox, nor does it affect the state of any messages in the queried mailbox (in particular, STATUS MUST NOT cause messages to lose the \Recent flag).

The STATUS command provides an alternative to opening a second IMAP4rev1 connection and doing an EXAMINE command on a mailbox to query that mailbox's status without deselecting the current mailbox in the first IMAP4rev1 connection.

Unlike the LIST command, the STATUS command is not guaranteed to be fast in its response. In some implementations, the server is obliged to open the mailbox read-only internally to obtain certain status information. Also unlike the LIST command, the STATUS command does not accept wildcards.

The currently defined status data items that can be requested are:

MESSAGES The number of messages in the mailbox.

RECENT The number of messages with the \Recent flag set.

UIDNEXT The next UID value that will be assigned to a new message in the mailbox. It is guaranteed that this value will not change unless new messages are added to the mailbox; and that it will change when new messages are added even if those new messages are subsequently expunged.

UIDVALIDITY The unique identifier validity value of the mailbox.

UNSEEN The number of messages which do not have the \Seen flag set.

Example: C: A042 STATUS blurdybloop (UIDNEXT MESSAGES)
S: * STATUS blurdybloop (MESSAGES 231 UIDNEXT 44292)
S: A042 OK STATUS completed

6.3.11. APPEND Command

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

Responses: no specific responses 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 to the end of the specified destination mailbox. This argument SHOULD be 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 a [MIME-IMB] content transfer encoding.

Note: There MAY be exceptions, e.g. draft messages, in which required [RFC-822] header lines are omitted in the message literal argument to APPEND. The full implications of doing so MUST be understood and carefully weighed.

If a flag parenthesized list is specified, the flags SHOULD be set in the resulting message; otherwise, the flag list of the resulting message is set empty by default.

If a date_time is specified, the internal date SHOULD be set in the resulting message; otherwise, the internal date of the resulting message is set to the current date and time by default.

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 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.

If the mailbox is currently selected, the normal new mail actions SHOULD occur. Specifically, the server SHOULD notify the client immediately via an untagged EXISTS response. If the server does not do so, the client MAY issue a NOOP command (or failing that, a CHECK command) after one or more APPEND commands.

```
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, STATUS, and APPEND), the following commands are valid in the selected state: CHECK, CLOSE, EXPUNGE, SEARCH, FETCH, STORE, COPY, and UID.

6.4.1. CHECK Command

Arguments: none

Responses: no specific responses 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

Responses: no specific responses 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 if a mailbox is selected, a SELECT, EXAMINE, or LOGOUT command MAY be issued without previously issuing a CLOSE 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

Responses: 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 [CHARSET] specification
searching criteria (one or more)

Responses: REQUIRED untagged response: SEARCH

Result: OK - search completed
NO - search error: can't search that [CHARSET] 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 can 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-IMB] body parts with terminal content media types other than TEXT and MESSAGE from consideration in SEARCH matching.

The OPTIONAL [CHARSET] specification consists of the word "CHARSET" followed by a registered [CHARSET]. It indicates the [CHARSET] of the strings that appear in the search criteria. [MIME-IMB] content transfer encodings, and [MIME-HDRS] strings in [RFC-822]/[MIME-IMB] headers, MUST be decoded before comparing text in a [CHARSET] other than US-ASCII. US-ASCII MUST be supported; other [CHARSET]s MAY be supported. If the server does not support the specified [CHARSET], 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 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 [RFC-822] 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 [RFC-822] 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

   Responses:  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 can 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>]<<partial>>
: The text of a particular body section. The section specification is a set of zero or more part specifiers delimited by periods. A part specifier is either a part number or one of the following: HEADER, HEADER.FIELDS, HEADER.FIELDS.NOT, MIME, and TEXT. An empty section specification refers to the entire message, including the header.

  Every message has at least one part number. Non-[MIME-IMB] messages, and non-multipart [MIME-IMB] messages with no encapsulated message, 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.

A part of type MESSAGE/RFC822 also has nested part numbers, referring to parts of the MESSAGE part's body.

The HEADER, HEADER.FIELDS, HEADER.FIELDS.NOT, and TEXT part specifiers can be the sole part specifier or can be prefixed by one or more numeric part specifiers, provided that the numeric part specifier refers to a part of type MESSAGE/RFC822. The MIME part specifier MUST be prefixed by one or more numeric part specifiers.

The HEADER, HEADER.FIELDS, and HEADER.FIELDS.NOT part specifiers refer to the [RFC-822] header of the message or of an encapsulated [MIME-IMT] MESSAGE/RFC822 message. HEADER.FIELDS and HEADER.FIELDS.NOT are followed by a list of field-name (as defined in [RFC-822]) names, and return a subset of the header. The subset returned by HEADER.FIELDS contains only those header fields with a field-name that matches one of the names in the list; similarly, the subset returned by HEADER.FIELDS.NOT contains only the header fields with a non-matching field-name. The field-matching is case-insensitive but otherwise exact. In all cases, the delimiting blank line between the header and the body is always included.

The MIME part specifier refers to the [MIME-IMB] header for this part.

The TEXT part specifier refers to the text body of the message, omitting the [RFC-822] header.

Here is an example of a complex message
with some of its part specifiers:

```
 HEADER     ([RFC-822] header of the message)
 TEXT       MULTIPART/MIXED
 1          TEXT/PLAIN
 2          APPLICATION/OCTET-STREAM
 3          MESSAGE/RFC822
 3.HEADER   ([RFC-822] header of the message)
 3.TEXT     ([RFC-822] text body of the message)
 3.1        TEXT/PLAIN
 3.2        APPLICATION/OCTET-STREAM
 4          MULTIPART/MIXED
 4.1        IMAGE/GIF
 4.1.MIME   ([MIME-IMB] header for the IMAGE/GIF)
 4.2        MESSAGE/RFC822
 4.2.HEADER ([RFC-822] header of the message)
 4.2.TEXT   ([RFC-822] text body of the message)
 4.2.1      TEXT/PLAIN
 4.2.2      MULTIPART/ALTERNATIVE
 4.2.2.1    TEXT/PLAIN
 4.2.2.2    TEXT/RICHTEXT
```

It is possible to fetch a substring of the designated text. This is done by appending an open angle bracket ("<"), the octet position of the first desired octet, a period, the maximum number of octets desired, and a close angle bracket (">") to the part specifier. If the starting octet is beyond the end of the text, an empty string is returned.

Any partial fetch that attempts to read beyond the end of the text is truncated as appropriate. A partial fetch that starts at octet 0 is returned as a partial fetch, even if this truncation happened.

> Note: this means that BODY[]<0.2048> of a 1500-octet message will return BODY[]<0> with a literal of size 1500, not BODY[].

> Note: a substring fetch of a HEADER.FIELDS or HEADER.FIELDS.NOT part specifier is calculated after subsetting the header.

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

BODYSTRUCTURE The [MIME-IMB] body structure of the message. This is computed by the server by parsing the [MIME-IMB] header fields in the [RFC-822] header and [MIME-IMB] headers.

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 internal date of the message.

RFC822 Functionally equivalent to BODY[], differing in the syntax of the resulting untagged FETCH data (RFC822 is returned).

RFC822.HEADER Functionally equivalent to BODY.PEEK[HEADER], differing in the syntax of the resulting untagged FETCH data (RFC822.HEADER is returned).

RFC822.SIZE The [RFC-822] size of the message.

RFC822.TEXT Functionally equivalent to BODY[TEXT], differing in the syntax of the resulting untagged FETCH data (RFC822.TEXT is returned).

UID The unique identifier for the message.

```
   Example:    C: A654 FETCH 2:4 (FLAGS BODY[HEADER.FIELDS (DATE FROM)])
               S: * 2 FETCH ....
               S: * 3 FETCH ....
               S: * 4 FETCH ....
               S: A654 OK FETCH completed
```

6.4.6. STORE Command

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

   Responses:  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.

> Note: regardless of whether or not the ".SILENT" suffix was used, the server SHOULD send an untagged FETCH response if a change to a message's flags from an external source is observed. The intent is that the status of the flags is determinate without a race condition.

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

6.4.7. COPY Command

Arguments: message set
mailbox name

Responses: no specific responses 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 end of 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

6.4.8. UID Command

Arguments: command name
command arguments

Responses: 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.

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 a UID was specified as a message data item to the FETCH.

```
   Example:    C: A999 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

   Responses:  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, a standard or standards-track revision of this specification, or an IESG-approved experimental protocol, 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 IMAP4rev1 AUTH=KERBEROS_V4 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. The information contained in a server response, identified by "Contents:" in the response descriptions below, is described by function, not by syntax. The precise syntax of server responses is described in the Formal Syntax section.

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

Status responses can be tagged or untagged. Tagged status responses indicate the completion result (OK, NO, or BAD status) 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 example, an impending system shutdown alert). 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 messages).

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 server data occurs when the IMAP connection is in selected state. In selected state, the server checks the mailbox for new messages as part of command execution. Normally, this is part of the execution of every command; hence, a NOOP command suffices to check for new messages. 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 are OK, NO, BAD, PREAUTH and BYE. OK, NO, and BAD may be tagged or untagged. PREAUTH and BYE are always untagged.

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.

NEWNAME Followed by a mailbox name and a new mailbox name. A SELECT or EXAMINE is failing because the target mailbox name no longer exists because it was renamed to the new mailbox name. This is a hint to the client that the operation can succeed if the SELECT or EXAMINE is reissued with the new mailbox name.

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

PERMANENTFLAGS Followed by a parenthesized list of flags, indicates which of the known flags that the client can 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 can 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 can succeed if the mailbox is first created by the CREATE command.

UIDVALIDITY Followed by a decimal number, indicates the unique identifier validity value.

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

Contents: 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 connection startup. It indicates that the connection is not yet authenticated and that a LOGIN command is needed.

Example: S: * OK IMAP4rev1 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

Contents: 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 can 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: A223 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: A223 NO COPY failed: disk is full
```

7.1.3. BAD Response

Contents: 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 can 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

Contents: OPTIONAL response code
human-readable text

The PREAUTH response is always untagged, and is one of three possible greetings at connection startup. It indicates that the connection has already been authenticated by external means and thus no LOGIN command is needed.

```
Example:    S: * PREAUTH IMAP4rev1 server logged in as Smith
```

7.1.5. BYE Response

Contents: 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 is sent under one of four conditions:

1) as part of a normal logout sequence. The server will close the connection after sending the tagged OK response to the LOGOUT command.

2) as a panic shutdown announcement. The server closes the connection immediately.

3) as an announcement of an inactivity autologout. The server closes the connection immediately.

4) as one of three possible greetings at connection startup, indicating that the server is not willing to accept a connection from this client. The server closes the connection immediately.

The difference between a BYE that occurs as part of a normal LOGOUT sequence (the first case) and a BYE that occurs because of a failure (the other three cases) is that the connection closes immediately in the failure case.

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 and mailbox status data are transmitted from the server to the client. Many of these responses typically result from a command with the same name.

7.2.1. CAPABILITY Response

Contents: 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 capability listing MUST include the atom "IMAP4rev1".

A capability name which begins with "AUTH=" indicates that the server supports that particular authentication mechanism.

Other capability names indicate that the server supports an extension, revision, or amendment to the IMAP4rev1 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 IMAP4rev1 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 "IMAP4rev1", and MUST ignore any unknown capability names.

Example: S: * CAPABILITY IMAP4rev1 AUTH=KERBEROS_V4 XPIG-LATIN

7.2.2. LIST Response

Contents: 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 can 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 can 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

Contents: 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 can 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 STATUS Response

Contents: name
status parenthesized list

The STATUS response occurs as a result of an STATUS command. It returns the mailbox name that matches the STATUS specification and the requested mailbox status information.

Example: S: * STATUS blurdybloop (MESSAGES 231 UIDNEXT 44292)

7.2.5. SEARCH Response

Contents: 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

7.2.6. FLAGS Response

Contents: 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 can 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 - Mailbox Size

These responses are always untagged. This is how changes in the size of the mailbox are trasnmitted from the server to the client. Immediately following the "*" token is a number that represents a message count.

7.3.1. EXISTS Response

Contents: 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

Contents: none

The RECENT response reports the number of messages with the \Recent flag set. This response occurs as a result of a SELECT or EXAMINE command, and if the size of the mailbox changes (e.g. new mail).

Note: It is not guaranteed that the message sequence numbers of recent messages will be a contiguous range of the highest n messages in the mailbox (where n is the value reported by the RECENT response). Examples of situations in which this is not the case are: multiple clients having the same mailbox open (the first session to be notified will see it as recent, others will probably see it as non-recent), and when the mailbox is re-ordered by a non-IMAP agent.

The only reliable way to identify recent messages is to look at message flags to see which have the \Recent flag set, or to do a SEARCH RECENT.

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

Example: S: * 5 RECENT

7.4. 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 a message sequence number.

7.4.1. EXPUNGE Response

Contents: 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

7.4.2. FETCH Response

Contents: 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>]<<origin_octet>>
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.

If the origin octet is specified, this string is a substring of the entire body contents, starting at that origin octet. This means that BODY[]<0> MAY be truncated, but BODY[] is NEVER truncated.

8-bit textual data is permitted if a [CHARSET] identifier is part of the body parameter parenthesized list for this section. Note that headers (part specifiers HEADER or MIME, or the header portion of a MESSAGE/RFC822 part), MUST be

7-bit; 8-bit characters are not permitted in headers. Note also that the blank line at the end of the header is always included in header data.

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 [MIME-IMB] body structure of a message. This is computed by the server by parsing the [MIME-IMB] header fields, defaulting various fields as necessary.

For example, a simple text message of 48 lines and 2279 octets can have a body structure of: ("TEXT" "PLAIN" ("CHARSET" "US-ASCII") NIL NIL "7BIT" 2279 48)

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.).

For example, a two part message consisting of a text and a BASE645-encoded text attachment can have a body structure of: (("TEXT" "PLAIN" ("CHARSET" "US-ASCII") NIL NIL "7BIT" 1152 23)("TEXT" "PLAIN" ("CHARSET" "US-ASCII" "NAME" "cc.diff") "<960723163407.20117h@cac.washington.edu>" "Compiler diff" "BASE64" 4554 73) "MIXED"))

Extension data follows the multipart subtype. Extension data is never returned with the BODY fetch, but can 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-IMB].

body disposition
A parenthesized list, consisting of a disposition type string followed by a parenthesized list of disposition attribute/value pairs. The disposition type and attribute names will be defined in a future standards-track revision to [DISPOSITION].

body language
A string or parenthesized list giving the body language value as defined in [LANGUAGE-TAGS].

Any following extension data are not yet defined in this version of the protocol. Such extension data can 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 media type name as defined in [MIME-IMB].

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

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-IMB].

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

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

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

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 content 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 can 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 body MD5 value as defined in [MD5].

body disposition
A parenthesized list with the same content and function as the body disposition for a multipart body part.

body language
A string or parenthesized list giving the body language value as defined in [LANGUAGE-TAGS].

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.

INTERNALDATE — A string representing the internal date of the message.

RFC822 — Equivalent to BODY[].

RFC822.HEADER — Equivalent to BODY.PEEK[HEADER].

RFC822.SIZE — A number expressing the [RFC-822] size of the message.

RFC822.TEXT — Equivalent to BODY[TEXT].

UID A number expressing the unique identifier of the message.

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

7.5. Server Responses - Command Continuation Request

The command continuation 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 IMAP4rev1 connection

The following is a transcript of an IMAP4rev1 connection. A long line in this sample is broken for editorial clarity.

```
S:   * OK IMAP4rev1 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 "17-Jul-1996 02:44:25 -0700"
      RFC822.SIZE 4286 ENVELOPE ("Wed, 17 Jul 1996 02:23:25 -0700 (PDT)"
      "IMAP4rev1 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 body[header]
S:    * 12 FETCH (BODY[HEADER] {350}
S:    Date: Wed, 17 Jul 1996 02:23:25 -0700 (PDT)
S:    From: Terry Gray <gray@cac.washington.edu>
S:    Subject: IMAP4rev1 WG mtg summary and minutes
S:    To: imap@cac.washington.edu
S:    cc: minutes@CNRI.Reston.VA.US, John Klensin <KLENSIN@INFOODS.MIT.EDU>
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 IMAP4rev1 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 one or more commas.

In the case of alternative or optional rules in which a later rule overlaps an earlier rule, the rule which is listed earlier MUST take priority. For example, "\Seen" when parsed as a flag is the \Seen flag name and not a flag_extension, even though "\Seen" could be parsed as a flag_extension. Some, but not all, instances of this rule are noted below.

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.

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

addr_adl        ::= nstring
                    ;; Holds route from [RFC-822] route-addr if
                    ;; non-NIL

addr_host       ::= nstring
                    ;; NIL indicates [RFC-822] group syntax.
                    ;; Otherwise, holds [RFC-822] domain name

addr_mailbox    ::= nstring
                    ;; NIL indicates end of [RFC-822] group; if
                    ;; non-NIL and addr_host is NIL, holds
                    ;; [RFC-822] group name.
                    ;; Otherwise, holds [RFC-822] local-part

addr_name       ::= nstring
                    ;; Holds phrase from [RFC-822] mailbox if
                    ;; non-NIL

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 / CTL / list_wildcards /
                    quoted_specials
```

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

auth_type       ::= atom
                    ;; Defined by [IMAP-AUTH]

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 body_fld_dsp
                    [SPACE body_fld_lang
                    [SPACE 1#body_extension]]]
                    ;; MUST NOT be returned on non-extensible
                    ;; "BODY" fetch

body_ext_mpart  ::= body_fld_param
                    [SPACE body_fld_dsp SPACE body_fld_lang
                    [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_dsp    ::= "(" string SPACE body_fld_param ")" / nil

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

body_fld_id     ::= nstring

body_fld_lang   ::= nstring / "(" 1#string ")"
```

```
body_fld_lines  ::= number

body_fld_md5    ::= nstring

body_fld_octets ::= number

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

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

body_type_basic ::= media_basic SPACE body_fields
                    ;; MESSAGE subtype MUST NOT be "RFC822"

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

body_type_msg   ::= media_message SPACE body_fields SPACE envelope
                    SPACE body SPACE body_fld_lines

body_type_text  ::= media_text SPACE body_fields SPACE body_fld_lines

capability      ::= "AUTH=" auth_type / atom
                    ;; New capabilities MUST begin with "X" or be
                    ;; registered with IANA as standard or
                    ;; standards-track

capability_data ::= "CAPABILITY" SPACE [1#capability SPACE] "IMAP4rev1"
                    [SPACE 1#capability]
                    ;; IMAP4rev1 servers which offer RFC 1730
                    ;; compatibility MUST list "IMAP4" as the first
                    ;; 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 / list / lsub /
                    rename / select / status / 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 / 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, 0x0D>

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       ::= 4digit

date_time       ::= <"> date_day_fixed "-" date_month "-" date_year
                    SPACE time SPACE zone <">

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       ::= "ENVELOPE" / "FLAGS" / "INTERNALDATE" /
                    "RFC822" [".HEADER" / ".SIZE" / ".TEXT"] /
                    "BODY" ["STRUCTURE"] / "UID" /
                    "BODY" [".PEEK"] section
                    ["<" number "." nz_number ">"]

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_fld_name ::= astring

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

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.  All case variants of
                    ;; INBOX (e.g. "iNbOx") MUST be interpreted as INBOX
                    ;; not as an astring.  Refer to section 5.1 for
                    ;; further semantic details of mailbox names.

mailbox_data    ::=  "FLAGS" SPACE flag_list /
                     "LIST" SPACE mailbox_list /
                     "LSUB" SPACE mailbox_list /
                     "MAILBOX" SPACE text /
                     "SEARCH" [SPACE 1#nz_number] /
                     "STATUS" SPACE mailbox SPACE
                     "(" #<status_att number ")" /
                     number SPACE "EXISTS" / number SPACE "RECENT"

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

media_basic     ::= (<"> ("APPLICATION" / "AUDIO" / "IMAGE" /
                    "MESSAGE" / "VIDEO") <">) / string)
                    SPACE media_subtype
                    ;; Defined in [MIME-IMT]

media_message   ::= <"> "MESSAGE" <"> SPACE <"> "RFC822" <">
```

```
                    ;; Defined in [MIME-IMT]

media_subtype   ::= string
                    ;; Defined in [MIME-IMT]

media_text      ::= <"> "TEXT" <"> SPACE media_subtype
                    ;; Defined in [MIME-IMT]

message_data    ::= nz_number SPACE ("EXPUNGE" /
                                     ("FETCH" SPACE msg_att))

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

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)

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        ::= *(continue_req / 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
                    ;; Server closes connection immediately

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

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

resp_text       ::= ["[" resp_text_code "]" SPACE] (text_mime2 / text)
                    ;; text SHOULD NOT begin with "[" or "="

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]
                    1#search_key
                    ;; [CHARSET] MUST be registered with IANA

search_key      ::= "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" /
                    ;; Above this line were in [IMAP2]
                    "DRAFT" /
                    "HEADER" SPACE header_fld_name 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 /
                    "(" 1#search_key ")"

section         ::= "[" [section_text / (nz_number *["." nz_number]
                    ["." (section_text / "MIME")])] "]"

section_text    ::= "HEADER" / "HEADER.FIELDS" [".NOT"]
                    SPACE header_list / "TEXT"

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>

status          ::= "STATUS" SPACE mailbox SPACE "(" 1#status_att ")"

status_att      ::= "MESSAGES" / "RECENT" / "UIDNEXT" / "UIDVALIDITY" /
                    "UNSEEN"

store           ::= "STORE" SPACE set SPACE store_att_flags

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

string          ::= quoted / literal

subscribe       ::= "SUBSCRIBE" SPACE mailbox

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

text            ::= 1*TEXT_CHAR
```

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

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

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".
```

10. Author's Note

This document is a revision or rewrite of earlier documents, and supercedes the protocol specification in those documents: RFC 1730, unpublished IMAP2bis.TXT document, RFC 1176, and RFC 1064.

11. Security Considerations

IMAP4rev1 protocol transactions, including electronic mail data, are sent in the clear over the network unless 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
University of Washington
4545 15th Aveneue NE
Seattle, WA 98105-4527

Phone: (206) 543-5762

EMail: MRC@CAC.Washington.EDU

Appendices

A. References

[ACAP] Myers, J. "ACAP -- Application Configuration Access Protocol", Work in Progress.

[CHARSET] Reynolds, J., and J. Postel, "Assigned Numbers", STD 2, RFC 1700, USC/Information Sciences Institute, October 1994.

[DISPOSITION] Troost, R., and Dorner, S., "Communicating Presentation Information in Internet Messages: The Content-Disposition Header", RFC 1806, June 1995.

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

[IMAP-COMPAT] Crispin, M., "IMAP4 Compatibility with IMAP2bis", RFC 2061, University of Washington, November 1996.

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

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

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

[IMAP-OBSOLETE] Crispin, M., "Internet Message Access Protocol - Obsolete Syntax", RFC 2062, University of Washington, November 1996.

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

[LANGUAGE-TAGS] Alvestrand, H., "Tags for the Identification of Languages", RFC 1766, March 1995.

[MD5] Myers, J., and M. Rose, "The Content-MD5 Header Field", RFC 1864, October 1995.

[MIME-IMB] Freed, N., and N. Borenstein, "MIME (Multipurpose Internet Mail Extensions) Part One: Format of Internet Message Bodies", RFC 2045, November 1996.

[MIME-IMT] Freed, N., and N. Borenstein, "MIME (Multipurpose Internet Mail Extensions) Part Two: Media Types", RFC 2046, November 1996.

[MIME-HDRS] Moore, K., "MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text", RFC 2047, November 1996.

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

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

[UTF-7] Goldsmith, D., and Davis, M., "UTF-7: A Mail-Safe Transformation Format of Unicode", RFC 1642, July 1994.

B. Changes from RFC 1730

1) The STATUS command has been added.

2) Clarify in the formal syntax that the "#" construct can never refer to multiple spaces.

3) Obsolete syntax has been moved to a separate document.

4) The PARTIAL command has been obsoleted.

5) The RFC822.HEADER.LINES, RFC822.HEADER.LINES.NOT, RFC822.PEEK, and RFC822.TEXT.PEEK fetch attributes have been obsoleted.

6) The "<" origin "." size ">" suffix for BODY text attributes has been added.

7) The HEADER, HEADER.FIELDS, HEADER.FIELDS.NOT, MIME, and TEXT part specifiers have been added.

8) Support for Content-Disposition and Content-Language has been added.

9) The restriction on fetching nested MULTIPART parts has been removed.

10) Body part number 0 has been obsoleted.

11) Server-supported authenticators are now identified by capabilities.

12) The capability that identifies this protocol is now called "IMAP4rev1". A server that provides backwards support for RFC 1730 SHOULD emit the "IMAP4" capability in addition to "IMAP4rev1" in its CAPABILITY response. Because RFC-1730 required "IMAP4" to appear as the first capability, it MUST listed first in the response.

13) A description of the mailbox name namespace convention has been added.

14) A description of the international mailbox name convention has been added.

15) The UID-NEXT and UID-VALIDITY status items are now called UIDNEXT and UIDVALIDITY. This is a change from the IMAP STATUS Work in Progress and not from RFC-1730

16) Add a clarification that a null mailbox name argument to the LIST command returns an untagged LIST response with the hierarchy delimiter and root of the reference argument.

17) Define terms such as "MUST", "SHOULD", and "MUST NOT".

18) Add a section which defines message attributes and more thoroughly details the semantics of message sequence numbers, UIDs, and flags.

19) Add a clarification detailing the circumstances when a client may send multiple commands without waiting for a response, and the circumstances in which ambiguities may result.

20) Add a recommendation on server behavior for DELETE and RENAME when inferior hierarchical names of the given name exist.

21) Add a clarification that a mailbox name may not be unilaterally unsubscribed by the server, even if that mailbox name no longer exists.

22) Add a clarification that LIST should return its results quickly without undue delay.

23) Add a clarification that the date_time argument to APPEND sets the internal date of the message.

24) Add a clarification on APPEND behavior when the target mailbox is the currently selected mailbox.

25) Add a clarification that external changes to flags should be always announced via an untagged FETCH even if the current command is a STORE with the ".SILENT" suffix.

26) Add a clarification that COPY appends to the target mailbox.

27) Add the NEWNAME response code.

28) Rewrite the description of the untagged BYE response to clarify its semantics.

29) Change the reference for the body MD5 to refer to the proper RFC.

30) Clarify that the formal syntax contains rules which may overlap, and that in the event of such an overlap the rule which occurs first takes precedence.

31) Correct the definition of body_fld_param.

32) More formal syntax for capability_data.

33) Clarify that any case variant of "INBOX" must be interpreted as INBOX.

34) Clarify that the human-readable text in resp_text should not begin with "[" or "=".

35) Change MIME references to Draft Standard documents.

36) Clarify \Recent semantics.

37) Additional examples.

C. Key Word Index