Abstract

The Internet Message Access Protocol, Version 4rev2 (IMAP4rev2) allows a client to access and manipulate electronic mail messages on a server. IMAP4rev2 permits manipulation of mailboxes (remote message folders) in a way that is functionally equivalent to local folders. IMAP4rev2 also provides the capability for an offline client to resynchronize with the server.

IMAP4rev2 includes operations for creating, deleting, and renaming mailboxes, checking for new messages, permanently removing messages, setting and clearing flags, RFC 5322 and RFC 2045 parsing, searching, and selective fetching of message attributes, texts, and portions thereof. Messages in IMAP4rev2 are accessed by the use of numbers. These numbers are either message sequence numbers or unique identifiers.

IMAP4rev2 does not specify a means of posting mail; this function is handled by a mail submission protocol such as RFC 6409.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on August 7, 2019.
Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the
document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal
Provisions Relating to IETF Documents
(https://trustee.ietf.org/license-info) in effect on the date of
publication of this document. Please review these documents
carefully, as they describe your rights and restrictions with respect
to this document. Code Components extracted from this document must
include Simplified BSD License text as described in Section 4.e of
the Trust Legal Provisions and are provided without warranty as
described in the Simplified BSD License.

This document may contain material from IETF Documents or IETF
Contributions published or made publicly available before November
10, 2008. The person(s) controlling the copyright in some of this
material may not have granted the IETF Trust the right to allow
modifications of such material outside the IETF Standards Process.
Without obtaining an adequate license from the person(s) controlling
the copyright in such materials, this document may not be modified
outside the IETF Standards Process, and derivative works of it may
not be created outside the IETF Standards Process, except to format
it for publication as an RFC or to translate it into languages other
than English.

Table of Contents

1. How to Read This Document .............................. 5
   1.1. Organization of This Document .................... 5
   1.2. Conventions Used in This Document ................ 5
   1.3. Special Notes to Implementors .................... 6
2. Protocol Overview ...................................... 7
   2.1. Link Level ....................................... 7
   2.2. Commands and Responses ............................ 7
       2.2.1. Client Protocol Sender and Server Protocol Receiver .. 7
       2.2.2. Server Protocol Sender and Client Protocol Receiver .. 8
   2.3. Message Attributes ................................ 9
       2.3.1. Message Numbers ................................ 9
       2.3.2. Flags Message Attribute ....................... 11
       2.3.3. Internal Date Message Attribute ................. 12
       2.3.4. [RFC-5322] Size Message Attribute ............ 13
       2.3.5. Envelope Structure Message Attribute ......... 13
       2.3.6. Body Structure Message Attribute ............. 13
   2.4. Message Texts .................................... 13
3. State and Flow Diagram ................................. 13
   3.1. Not Authenticated State ............................ 13
3.2. Authenticated State ........................................ 13
3.3. Selected State ............................................. 14
3.4. Logout State ................................................. 14
4. Data Formats .................................................. 16
4.1. Atom .......................................................... 16
4.1.1. Sequence set and UID set ................................. 16
4.2. Number ......................................................... 16
4.3. String ........................................................ 16
4.3.1. 8-bit and Binary Strings ................................. 17
4.4. Parenthesized List ........................................... 18
4.5. NIL .............................................................. 18
5. Operational Considerations ..................................... 18
5.1. Mailbox Naming ............................................... 18
5.1.1. Mailbox Hierarchy Naming ................................. 19
5.1.2. Namespaces ................................................. 19
5.2. Mailbox Size and Message Status Updates .................... 21
5.3. Response when no Command in Progress ....................... 21
5.4. Autologout Timer ............................................. 21
5.5. Multiple Commands in Progress (Command Pipelining) ...... 22
6. Client Commands ............................................... 23
6.1. Client Commands - Any State ................................. 23
6.1.1. CAPABILITY Command .................................... 24
6.1.2. NOOP Command ............................................. 25
6.1.3. LOGOUT Command ......................................... 25
6.2. Client Commands - Not Authenticated State ................ 26
6.2.1. STARTTLS Command ...................................... 26
6.2.2. AUTHENTICATE Command ................................. 27
6.2.3. LOGIN Command ........................................... 30
6.3. Client Commands - Authenticated State ..................... 31
6.3.1. ENABLE Command .......................................... 31
6.3.2. SELECT Command ......................................... 33
6.3.3. EXAMINE Command ........................................ 35
6.3.4. CREATE Command ......................................... 35
6.3.5. DELETE Command ......................................... 36
6.3.6. RENAME Command ......................................... 38
6.3.7. SUBSCRIBE Command ...................................... 40
6.3.8. UNSUBSCRIBE Command ................................... 41
6.3.9. LIST Command ............................................. 41
6.3.10. LSUB Command ........................................... 44
6.3.11. NAMESPACE Command ..................................... 45
6.3.12. STATUS Command ........................................ 49
6.3.13. APPEND Command ........................................ 51
6.3.14. IDLE Command ............................................ 53
6.4. Client Commands - Selected State ........................... 55
6.4.1. CHECK Command ........................................... 56
6.4.2. CLOSE Command ........................................... 56
6.4.3. UNSELECT Command ........................................ 57
6.4.4. EXPUNGE Command ........................................ 57
6.4.5. SEARCH Command ........................................ 58
6.4.6. FETCH Command ........................................ 63
6.4.7. STORE Command ........................................ 66
6.4.8. COPY Command ........................................ 68
6.4.9. MOVE Command ........................................ 69
6.4.10. UID Command .......................................... 70
6.5. Client Commands - Experimental/Expansion ............ 72
6.5.1. X<atom> Command ...................................... 72
7. Server Responses ........................................... 73
7.1. Server Responses - Status Responses .................. 74
7.1.1. OK Response .......................................... 81
7.1.2. NO Response .......................................... 81
7.1.3. BAD Response ......................................... 82
7.1.4. PREAUTH Response ..................................... 82
7.1.5. BYE Response ......................................... 83
7.2. Server Responses - Server and Mailbox Status ....... 83
7.2.1. The ENABLED Response ................................ 83
7.2.2. CAPABILITY Response ................................ 84
7.2.3. LIST Response ......................................... 85
7.2.4. LSUB Response ......................................... 87
7.2.5. NAMESPACE Response ................................... 88
7.2.6. STATUS Response ....................................... 88
7.2.7. ESEARCH Response .................................... 88
7.2.8. FLAGS Response ........................................ 89
7.3. Server Responses - Mailbox Size ....................... 89
7.3.1. EXISTS Response ....................................... 89
7.4. Server Responses - Message Status ..................... 90
7.4.1. EXPUNGE Response ..................................... 90
7.4.2. FETCH Response ....................................... 91
7.5. Server Responses - Command Continuation Request ..... 96
8. Sample IMAP4rev2 connection ............................... 96
9. Formal Syntax ............................................... 97
10. Author’s Note .............................................. 112
11. Security Considerations .................................. 112
11.1. STARTTLS Security Considerations .................... 112
11.2. COPYUID and APPENDUID response codes .............. 112
11.3. Other Security Considerations ......................... 113
12. IANA Considerations ...................................... 113
12.1. Updates to IMAP4 Capabilities registry ............... 114
12.2. GSSAPI/SASL service name .............................. 114
13. References ................................................. 114
13.1. Normative References ................................. 114
13.2. Informative References (related protocols) .......... 117
13.3. Informative References (historical aspects of IMAP and related protocols) .................. 118

Appendix A. Backward compatibility with IMAP4rev1 ........ 119
A.1. Mailbox International Naming Convention ............ 119
Appendix B. Changes from RFC 3501 / IMAP4rev1 .......... 121
Appendix C. Acknowledgement .................. 122
Index ................................................. 123
Authors’ Addresses ................................. 128

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 IMAP4rev2 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 IMAP4rev2 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

"Conventions" are basic principles or procedures. Document conventions are noted in this section.

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

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [KEYWORDS].

The word "can" (not "may") is used to refer 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.

There are several protocol conventions in IMAP. These refer to aspects of the specification which are not strictly part of the IMAP protocol, but reflect generally-accepted practice. Implementations need to be aware of these conventions, and avoid conflicts whether or not they implement the convention. For example, "&" may not be used as a hierarchy delimiter since it conflicts with the Mailbox International Naming Convention, and other uses of "&" in mailbox names are impacted as well.

1.3. Special Notes to Implementors

Implementors of the IMAP protocol are strongly encouraged to read the IMAP implementation recommendations document [IMAP-IMPLEMENTATION] in conjunction with this document, to help understand the intricacies of this protocol and how best to build an interoperable product.

IMAP4rev2 is designed to be upwards compatible from the [IMAP2] and unpublished IMAP2bis protocols. IMAP4rev2 is largely compatible with the IMAP4rev1 protocol described in RFC 3501 and the IMAP4 protocol described in RFC 1730; the exception being in certain facilities added in RFC 1730 that proved problematic and were subsequently removed. In the course of the evolution of IMAP4rev2, some aspects in the earlier protocols have become obsolete. Obsolete commands, responses, and data formats which an IMAP4rev2 implementation can 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.

IMAP was originally developed for the older [RFC-822] standard, and as a consequence several fetch items in IMAP incorporate "RFC822" in their name. With the exception of RFC822.SIZE, there are more modern replacements; for example, the modern version of RFC822.HEADER is BODY.PEEK[HEADER]. In all cases, "RFC822" should be interpreted as a reference to the updated [RFC-5322] standard.
2. Protocol Overview

2.1. Link Level

The IMAP4rev2 protocol assumes a reliable data stream such as that provided by TCP. When TCP is used, an IMAP4rev2 server listens on port 143.

2.2. Commands and Responses

An IMAP4rev2 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 IMAP4rev2 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.

Clients MUST follow the syntax outlined in this specification strictly. It is a syntax error to send a command with missing or extraneous spaces or arguments.

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 a 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 IMAP4rev2 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 a protocol error such as unrecognized command or command syntax error).

Servers SHOULD enforce the syntax outlined in this specification strictly. Any client command with a protocol syntax error, including (but not limited to) missing or extraneous spaces or arguments, SHOULD be rejected, and the client given a BAD server completion response.

The protocol receiver of an IMAP4rev2 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 can be retrieved individually or in conjunction with other attributes or message texts.

2.3.1. Message Numbers

Messages in IMAP4rev2 are accessed by one of two numbers; the unique identifier or the message sequence number.

2.3.1.1. Unique Identifier (UID) Message Attribute

An unsigned 32-bit value assigned to each message, which when used with the unique identifier validity value (see below) forms a 64-bit value that MUST NOT refer to any other message in the mailbox or any subsequent mailbox with the same name forever. 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.

The unique identifier of a message MUST NOT change during the session, and SHOULD NOT change between sessions. Any change of unique identifiers between sessions MUST be detectable using the UIDVALIDITY mechanism discussed below. Persistent unique identifiers are required for 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 are two 32-bit unsigned values which aid in unique identifier handling: the next unique identifier value (UIDNEXT) and the unique identifier validity value (UIDVALIDITY).

The next unique identifier value is the predicted value that will be assigned to a new message in the mailbox. Unless the unique identifier validity also changes (see below), the next unique identifier value MUST have the following two characteristics. First, the next unique identifier value MUST NOT change unless new messages are added to the mailbox; and second, the next unique identifier value MUST change whenever new messages are added to the mailbox, even if those new messages are subsequently expunged.

Note: The next unique identifier value is intended to provide a means for a client to determine whether any messages have been
delivered to the mailbox since the previous time it checked this value. It is not intended to provide any guarantee that any message will have this unique identifier. A client can only assume, at the time that it obtains the next unique identifier value, that messages arriving after that time will have a UID greater than or equal to that value.

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

Note: Ideally, unique identifiers SHOULD persist at all times. Although this specification recognizes that failure to persist can be unavoidable in certain server environments, it STRONGLY ENCOURAGES message store implementation techniques that avoid this problem. For example:

1. 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 identifiers are no longer strictly ascending as a result of the re-ordering.

2. If the message store has no mechanism to store unique identifiers, it must regenerate unique identifiers at each session, and each session must have a unique UIDVALIDITY value.

3. If the mailbox is deleted and a new mailbox with the same name is created at a later date, the server must either keep track of unique identifiers from the previous instance of the mailbox, or it must assign a new UIDVALIDITY value to the new instance of the mailbox. A good UIDVALIDITY value to use in this case 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.

4. The combination of mailbox name, UIDVALIDITY, and UID must refer to a single immutable message on that server forever. In particular, the internal date, [RFC-5322] size, envelope, body structure, and message texts (RFC822, RFC822.HEADER, RFC822.TEXT, and all BODY[...] fetch data items) must never

Melnikov & Leiba Expires August 7, 2019 [Page 10]
change. This does not include message numbers, nor does it include attributes that can be set by a STORE command (e.g., FLAGS).

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. The number of messages in the mailbox is also 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 "11 EXISTS" 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 IMAP4rev2. A flag of either type can be permanent or session-only.

A system flag is a flag name that is pre-defined in this specification and begin with ":". Certain system flags ("\Seen", "\Answered", "\Flagged", "\Deleted") have special semantics described elsewhere in this document. 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 This flag was in used in IMAP4rev1 and is now deprecated.

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). Some keywords that start with "$" are also defined in this specification.

This document defines several keywords that were not originally defined in RFC 3501, but which were found to be useful by client implementations. These keywords SHOULD be supported (i.e. allowed in APPEND, COPY, MOVE and SEARCH commands) by server implementations:

$Forwarded Message has been forwarded to another email address, embedded within or attached to a new message. An email client sets this keyword when it successfully forwards the message to another email address. Typical usage of this keyword is to show a different (or additional) icon for a message that has been forwarded. Once set, the flag SHOULD NOT be cleared.

$MDNSent Message Disposition Notification was generated and sent for this message.

A flag can 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, concurrent and subsequent sessions will see any change in permanent flags. Changes to session flags are valid only in that session.

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-5322] 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 IMAP4rev2 COPY or MOVE command, this SHOULD be the internal date and time of the source message. In the case of messages delivered by the IMAP4rev2 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-5322] Size Message Attribute

The number of octets in the message, as expressed in [RFC-5322] format.

2.3.5. Envelope Structure Message Attribute

A parsed representation of the [RFC-5322] header of the message. Note that the IMAP Envelope structure is not the same as an [SMTP] envelope.

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-5322] text of a message, IMAP4rev2 permits the fetching of portions of the full message text. Specifically, it is possible to fetch the [RFC-5322] message header, [RFC-5322] message body, a [MIME-IMB] body part, or a [MIME-IMB] header.

3. State and Flow Diagram

Once the connection between client and server is established, an IMAP4rev2 connection is in one of four states. The initial state is identified in the server greeting. Most commands are only valid in certain states. It is a protocol error for the client to attempt a command while the connection is in an inappropriate state, and the server will respond with a BAD or NO (depending upon server implementation) command completion result.

3.1. Not Authenticated State

In the not 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 the 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, after an error in selecting a mailbox, or after a successful CLOSE command.

3.3. Selected State

In a 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 the logout state, the connection is being terminated. This state can be entered as a result of a client request (via the LOGOUT command) or by unilateral action on the part of either the client or server.

If the client requests the logout state, the server MUST send an untagged BYE response and a tagged OK response to the LOGOUT command before the server closes the connection; and the client MUST read the tagged OK response to the LOGOUT command before the client closes the connection.

A server MUST NOT unilaterally close the connection without sending an untagged BYE response that contains the reason for having done so. A client SHOULD NOT unilaterally close the connection, and instead SHOULD issue a LOGOUT command. If the server detects that the client has unilaterally closed the connection, the server MAY omit the untagged BYE response and simply close its connection.
(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, unsolicited CLOSED response code or
    failed SELECT or EXAMINE command
(7) LOGOUT command, server shutdown, or connection closed
4. Data Formats

IMAP4rev2 uses textual commands and responses. Data in IMAP4rev2 can be in one of several forms: atom, number, string, parenthesized list, or NIL. Note that a particular data item may take more than one form; for example, a data item defined as using "astring" syntax may be either an atom or a string.

4.1. Atom

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

4.1.1. Sequence set and UID set

A set of messages can be referenced by a sequence set containing either message sequence numbers or unique identifiers. See Section 9 for details. Sequence sets can contain ranges (e.g. "5:50"), an enumeration of specific message/UID numbers, a special symbol "*", or a combination of the above.

A "UID set" is similar to the sequence set of unique identifiers; however, the "*" value for a sequence number is not permitted.

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 three forms: synchronizing literal, non-synchronizing literal or quoted string. The synchronizing literal form is the general form of string. The non-synchronizing literal form is also the general form, but has length limitation. The quoted string form is an alternative that avoids the overhead of processing a literal at the cost of limitations of characters which may be used.

When the distinction between synchronizing and non-synchronizing literals is not important, this document just uses the term "literal".

A synchronizing 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 synchronizing literals transmitted from server to client, the CRLF is immediately followed by the octet data. In the case of synchronizing 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).

The non-synchronizing literal is an alternate form of synchronizing
literal, and it may appear in communication from client to server
instead of the synchronizing form of literal. The non-synchronizing
literal form MUST NOT be sent from server to client. The non-
synchronizing literal is distinguished from the synchronizing literal
by having a plus ("+") between the octet count and the closing brace
("}"). The server does not generate a command continuation request
in response to a non-synchronizing literal, and clients are not
required to wait before sending the octets of a non-
synchronizing literal. Non-synchronizing literals MUST NOT be larger than 4096
octets. Any literal larger than 4096 bytes MUST be sent as a
synchronizing literal. (Non-synchronizing literals defined in this
document are the same as non-synchronizing literals defined by the
LITERAL- extension from [RFC7888]. See that document for details on
how to handle invalid non-synchronizing literals longer than 4096
octets and for interaction with other IMAP extensions.)

A quoted string is a sequence of zero or more Unicode characters,
excluding CR and LF, encoded in UTF-8, with double quote (<"">)
characters at each end.

The empty string is represented as "" (a quoted string with zero
characters between double quotes), as {0} followed by CRLF (a
synchronizing literal with an octet count of 0) or as {0+} followed
by CRLF (a non-synchronizing literal with an octet count of 0).

Note: Even if the octet count is 0, a client transmitting a
synchronizing 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. IMAP4rev2 implementations MAY
transmit 8-bit or multi-octet characters in literals, but SHOULD do
so only when the [CHARSET] is identified.

IMAP4rev2 is compatible with [I18N-HDRS]. As a result, the
identified charset for header-field values with 8-bit content is
UTF-8 [UTF-8]. IMAP4rev2 implementations MUST accept and MAY
transmit [UTF-8] text in quoted-strings as long as the string does
not contain NUL, CR, or LF. This differs from IMAP4rev1
implementations.
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 form "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 ()..

Note: NIL is never used for any data item which takes the form of an atom. For example, a mailbox name of "NIL" is a mailbox named NIL as opposed to a non-existent mailbox name. This is because mailbox uses "astring" syntax which is an atom or a string. Conversely, an addr-name of NIL is a non-existent personal name, because addr-name uses "nstring" syntax which is NIL or a string, but never an atom.

5. Operational Considerations

The following rules are listed here to ensure that all IMAP4rev2 implementations interoperate properly.

5.1. Mailbox Naming

In IMAP4rev2, Mailbox names are encoded in Net-Unicode [NET-UNICODE] (this differs from IMAP4rev1). Client implementations MAY attempt to create Net-Unicode mailbox names, and MUST interpret any 8-bit mailbox names returned by LIST or LSUB as [NET-UNICODE]. Server implementations MUST prohibit the creation of 8-bit mailbox names that do not comply with Net-Unicode (however, servers MAY accept a de-normalized UTF-8 mailbox name and convert it to Net-Unicode prior to mailbox creation).
The case-insensitive mailbox name INBOX is a special name reserved to mean "the primary mailbox for this user on this server". (Note that this special name may not exist on some servers for some users.) The interpretation of all other names is implementation-dependent.

In particular, this specification takes no position on case sensitivity in non-INBOX mailbox names. Some server implementations are fully case-sensitive in ASCII range; others preserve case of a newly-created name but otherwise are case-insensitive; and yet others coerce names to a particular case. Client implementations MUST interact with any of these.

There are certain client considerations when creating a new mailbox name:

1. Any character which is one of the atom-specials (see the Formal Syntax) will require that the mailbox name be represented as a quoted string or literal.

2. CTL and other non-graphic characters are difficult to represent in a user interface and are best avoided. Servers MAY refuse to create mailbox names containing Unicode CTL characters.

3. Although the list-wildcard characters ("%" and "*"), are valid in a mailbox name, it is difficult to use such mailbox names with the LIST and LSUB commands due to the conflict with wildcard interpretation.

4. Usually, a character (determined by the server implementation) is reserved to delimit levels of hierarchy.

5. Two characters, "#" and "&", have meanings by convention, and should be avoided except when used in that convention.

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. Namespaces

Personal Namespace: A namespace that the server considers within the personal scope of the authenticated user on a particular connection. Typically, only the authenticated user has access to mailboxes in their Personal Namespace. It is the part of the namespace that belongs to the user that is allocated for mailboxes. If an INBOX
exists for a user, it MUST appear within the user’s personal namespace. In the typical case, there SHOULD be only one Personal Namespace on a server.

Other Users’ Namespace: A namespace that consists of mailboxes from the Personal Namespaces of other users. To access mailboxes in the Other Users’ Namespace, the currently authenticated user MUST be explicitly granted access rights. For example, it is common for a manager to grant to their secretary access rights to their mailbox. In the typical case, there SHOULD be only one Other Users’ Namespace on a server.

Shared Namespace: A namespace that consists of mailboxes that are intended to be shared amongst users and do not exist within a user’s Personal Namespace.

The namespaces a server uses MAY differ on a per-user basis.

5.1.2.1. Historic 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 a mailbox name of "#news.comp.mail.misc", and the name "comp.mail.misc" can refer to a different object (e.g., a user’s private mailbox).

Namespaces that include the "#" character are not IMAP URL [IMAP-URL] friendly requiring the "#" character to be represented as %23 when within URLs. As such, server implementers MAY instead consider using namespace prefixes that do not contain the "#" character.

5.1.2.2. Common namespace models

Previous version of this protocol does not define a default server namespace. Two common namespace models have evolved:

The "Personal Mailbox" model, in which the default namespace that is presented consists of only the user’s personal mailboxes. To access shared mailboxes, the user must use an escape mechanism to reach another namespace.
The "Complete Hierarchy" model, in which the default namespace that is presented includes the user’s personal mailboxes along with any other mailboxes they have access to.

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 message delivery), change the flags of the messages 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. In particular, it is NOT permitted to send an EXISTS response that would reduce the number of messages in the mailbox; only the EXPUNGE response can do this.

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 the 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 applies to sessions after authentication, the duration of that timer MUST be at least 30 minutes. The receipt of ANY command from the client during that interval SHOULD suffice to reset the autologout timer.
5.5. Multiple Commands in Progress (Command Pipelining)

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, e.g., 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 the completion result response before sending a command with message sequence numbers.

Note: EXPUNGE responses are permitted while UID FETCH, UID STORE, and UID SEARCH are in progress. If the client sends a UID command, it MUST wait for a completion result response before sending a command which uses message sequence numbers (this may include UID SEARCH). Any message sequence numbers in an argument to UID SEARCH are associated with messages prior to the effect of any untagged EXPUNGE returned by the UID SEARCH.

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

UID SEARCH + UID SEARCH may be valid or invalid as a non-waiting command sequence, depending upon whether or not the second UID SEARCH contains message sequence numbers.

6. Client Commands

IMAP4rev2 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 9).

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.

The state of a connection is only changed by successful commands which are documented as changing state. A rejected command (BAD response) never changes the state of the connection or of the selected mailbox. A failed command (NO response) generally does not change the state of the connection or of the selected mailbox; the exception being the SELECT and EXAMINE commands.

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 "IMAP4rev2" as one of the listed capabilities before the (tagged) OK response.

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 IMAP4rev2 set defined in this specification, are enabled without explicit client action to invoke the capability.

Client and server implementations MUST implement the STARTTLS, LOGINDISABLED, and AUTH=PLAIN (described in [PLAIN]) capabilities. See the Security Considerations section for important information.

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 IMAP4rev2 STARTTLS AUTH=GSSAPI LOGINDISABLED
S: abcd OK CAPABILITY completed
C: efgf STARTTLS
S: efgf OK STARTTLS completed
<TLS negotiation, further commands are under [TLS] layer>
C: ijkl CAPABILITY
S: * CAPABILITY IMAP4rev2 AUTH=GSSAPI AUTH=PLAIN
S: ijkl 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 (this is the
preferred method to do this). 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: * 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 IMAP4rev2 Server logging out
          S: A023 OK LOGOUT completed
          (Server and client then close the connection)
6.2. Client Commands - Not Authenticated State

In the not authenticated state, the AUTHENTICATE or LOGIN command establishes authentication and enters the authenticated state. The AUTHENTICATE command provides a general mechanism for a variety of authentication techniques, privacy protection, and integrity checking; whereas the LOGIN command uses a traditional user name and plaintext password pair and has no means of establishing privacy protection or integrity checking.

The STARTTLS command is an alternate form of establishing session privacy protection and integrity checking, but does not by itself establish authentication or enter the authenticated state.

Server implementations MAY allow access to certain mailboxes without establishing authentication. This can be done by means of the ANONYMOUS [SASL] authenticator described in [ANONYMOUS]. An older convention is a LOGIN command using the userid "anonymous"; in this case, a password is required although the server may choose to accept any password. The restrictions placed on anonymous users are implementation-dependent.

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

In addition to the universal commands (CAPABILITY, NOOP, and LOGOUT), the following commands are valid in the not authenticated state: STARTTLS, AUTHENTICATE and LOGIN. See the Security Considerations section for important information about these commands.

6.2.1. STARTTLS Command

Arguments: none

Responses: no specific response for this command

Result: OK - starttls completed, begin TLS negotiation
BAD - command unknown or arguments invalid

A [TLS] negotiation begins immediately after the CRLF at the end of the tagged OK response from the server. Once a client issues a STARTTLS command, it MUST NOT issue further commands until a server response is seen and the [TLS] negotiation is complete.

The server remains in the non-authenticated state, even if client credentials are supplied during the [TLS] negotiation. This does not preclude an authentication mechanism such as EXTERNAL (defined in
(SASL) from using client identity determined by the [TLS] negotiation.

Once [TLS] has been started, the client MUST discard cached information about server capabilities and SHOULD re-issue the CAPABILITY command. This is necessary to protect against man-in-the-middle attacks which alter the capabilities list prior to STARTTLS. The server MAY advertise different capabilities, and in particular SHOULD NOT advertise the STARTTLS capability, after a successful STARTTLS command.

Example:

C: a001 CAPABILITY
S: * CAPABILITY IMAP4rev2 STARTTLS LOGINDISABLED
S: a001 OK CAPABILITY completed
C: a002 STARTTLS
S: a002 OK Begin TLS negotiation now
<TLS negotiation, further commands are under [TLS] layer>
C: a003 CAPABILITY
S: * CAPABILITY IMAP4rev2 AUTH=PLAIN
S: a003 OK CAPABILITY completed
C: a004 LOGIN joe password
S: a004 OK LOGIN completed

6.2.2. AUTHENTICATE Command

Arguments: SASL authentication mechanism name
OPTIONAL initial response

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 a [SASL] authentication mechanism 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 security layer 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 AUTHENTICATE command supports the optional "initial response" feature defined in Section 5.1 of [SASL]. The client doesn’t need to use it. If a SASL mechanism supports "initial response", but it is
not specified by the client, the server handles this as specified in Section 3 of [SASL].

The service name specified by this protocol’s profile of [SASL] is "imap".

The authentication protocol exchange consists of a series of server challenges and client responses 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 (see Section 4 of [RFC4648]) string. The client response consists of a single line consisting of a BASE64 encoded string. If the client wishes to cancel an authentication exchange, it issues a line consisting of a single "+". If the server receives such a response, or if it receives an invalid BASE64 string (e.g. characters outside the BASE64 alphabet, or non-terminal "="), it MUST reject the AUTHENTICATE command by sending a tagged BAD response.

As with any other client response, this initial response MUST be encoded as BASE64. It also MUST be transmitted outside of a quoted string or literal. To send a zero-length initial response, the client MUST send a single pad character ("="). This indicates that the response is present, but is a zero-length string.

When decoding the BASE64 data in the initial response, decoding errors MUST be treated as in any normal SASL client response, i.e. with a tagged BAD response. In particular, the server should check for any characters not explicitly allowed by the BASE64 alphabet, as well as any sequence of BASE64 characters that contains the pad character (‘=’) anywhere other than the end of the string (e.g., ":AAA" and "AAA=BBB" are not allowed).

If the client uses an initial response with a SASL mechanism that does not support an initial response, the server MUST reject the command with a tagged BAD response.

If a security layer is negotiated through the [SASL] authentication exchange, it 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.

While client and server implementations MUST implement the AUTHENTICATE command itself, it is not required to implement any authentication mechanisms other than the PLAIN mechanism described in [PLAIN]. Also, an authentication mechanism is not required to support any security layers.
Note: a server implementation MUST implement a configuration in which it does NOT permit any plaintext password mechanisms, unless either the STARTTLS command has been negotiated or some other mechanism that protects the session from password snooping has been provided. Server sites SHOULD NOT use any configuration which permits a plaintext password mechanism without such a protection mechanism against password snooping. Client and server implementations SHOULD implement additional [SASL] mechanisms that do not use plaintext passwords, such the GSSAPI mechanism described in [SASL] and/or the [DIGEST-MD5] mechanism.

Servers and clients can support multiple authentication mechanisms. The server SHOULD list its supported authentication mechanisms in the response to the CAPABILITY command so that the client knows which authentication mechanisms to use.

A server MAY include a CAPABILITY response code in the tagged OK response of a successful AUTHENTICATE command in order to send capabilities automatically. It is unnecessary for a client to send a separate CAPABILITY command if it recognizes these automatic capabilities. This should only be done if a security layer was not negotiated by the AUTHENTICATE command, because the tagged OK response as part of an AUTHENTICATE command is not protected by encryption/integrity checking. [SASL] requires the client to re-issue a CAPABILITY command in this case. The server MAY advertise different capabilities after a successful AUTHENTICATE command.

If an AUTHENTICATE command fails with a NO response, the client MAY try another authentication mechanism by issuing another AUTHENTICATE command. It MAY also attempt to authenticate by using the LOGIN command (see Section 6.2.3 for more detail). In other words, the client MAY request authentication types in decreasing order of preference, with the LOGIN command as a last resort.

The authorization identity passed from the client to the server during the authentication exchange is interpreted by the server as the user name whose privileges the client is requesting.
Example: S: * OK IMAP4rev2 Server
C: A001 AUTHENTICATE GSSAPI
S: +
C: YIIB+wYJKoZIhvcSAQICAQBuggHqMIIB5qAAdAgEFOQMCAQ6iBwMFACAAABACqgEnYlBIjCCACB6gAwIBaESGxB1Lndhch2hpbdm0
b2u4ZWRiOl0wK6AgEDEQWlhSmEaWhbcbsac2hpmdmFtcy5iyW
Mud2FzaGlue23Rvb15ZWhjgdMwgdCgAwIBaaEDEQEDooHDBIHA
cS1GSa5b+fXnPZnMxB9SjL801lj2SRyb+3501XM1jen/jJnkjX
AleKTez6BPzj8ud8EtoOuNfKgweViy/9B9bcy1uuAE2H1oy
C/PHXNNU92rBziJ8Lm0tTnc98kUpjXnH2hsMczS5Mx2GR6dGknt
I0iaGcReRJuUsW0uBmkKKhvMMdR9T3EzdpqsBd7jzCNMWo1tjhi
vd5zovQ1Fq02Wjc2+y46vKp/iXxUJuDiisyXF0Y8+5GtpAL
pHdSc/pIgmMIGjoMCAQGigZsEgZg2onmSuxoDHEAlw9bcW9n
FdfxDrdrphVGYVDRIZcCMCTzvUboqb5KjY1NJKJsfjRq1BYBEdE
NKfzKg5D1VBnrw81u0cP8NOQCLR5XkoMHC0Dr/80z1QzbNqhx
O6652NpftQLQjQvenwD113YpxOwDmxzWZVN/XrEQOwp6GCGgXTB
vCyLW1LnbaukdeYBkBHPjd8t/1x5yge=
S: + YggsGcGxGsIb3EgEcAG1Ab1kwV6ADAgEFOQMCAQ+iSz8BoAMC
AQGiQqRAtHTEuOP2Bxb9sBYFR4S1DZxmg391xmRBOhXKRdDA0
uHTCOT9Bq30sUTUXUl0kCsFl0a8j+gvD1gHuzgWHPSoQge=
C: S: + YDMGCSgGSIb3EgECAG1BAAD///////6jcyG4E3KkTzeBiVHe
ceP2CWY0SR0AQAgAAQEBAQ=
C: YDMGCSgGSIb3EgECAG1BAAD///////3LQBHXTpFfZgrejpLL1ImP
wkhbfa2QteAQAgGlyYWE=
S: A001 OK GSSAPI authentication successful

Note: The line breaks within server challenges and client responses are for editorial clarity and are not in real authenticators.

6.2.3. 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.

A server MAY include a CAPABILITY response code in the tagged OK response to a successful LOGIN command in order to send capabilities automatically. It is unnecessary for a client to send a separate CAPABILITY command if it recognizes these automatic capabilities.
Example: C: a001 LOGIN SMITH SESAME
S: a001 OK LOGIN completed

Note: Use of the LOGIN command over an insecure network (such as the Internet) is a security risk, because anyone monitoring network traffic can obtain plaintext passwords. The LOGIN command SHOULD NOT be used except as a last resort, and it is recommended that client implementations have a means to disable any automatic use of the LOGIN command.

Unless either the client is accessing IMAP service on IMAPS port [RFC8314], the STARTTLS command has been negotiated or some other mechanism that protects the session from password snooping has been provided, a server implementation MUST implement a configuration in which it advertises the LOGINDISABLED capability and does NOT permit the LOGIN command. Server sites SHOULD NOT use any configuration which permits the LOGIN command without such a protection mechanism against password snooping. A client implementation MUST NOT send a LOGIN command if the LOGINDISABLED capability is advertised.

6.3. Client Commands - Authenticated State

In the 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 the selected state.

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

6.3.1. ENABLE Command

Arguments: capability names
Responses: no specific responses for this command
Result: OK - Relevant capabilities enabled
BAD - No arguments, or syntax error in an argument

Several IMAP extensions allow the server to return unsolicited responses specific to these extensions in certain circumstances. However, servers cannot send those unsolicited responses (with the exception of response codes included in tagged or untagged OK/NO/BAD responses, which can always be sent) until they know that the clients support such extensions and thus won’t choke on the extension response data.
The ENABLE command provides an explicit indication from the client that it supports particular extensions.

The ENABLE command takes a list of capability names, and requests the server to enable the named extensions. Once enabled using ENABLE, each extension remains active until the IMAP connection is closed. For each argument, the server does the following:

- If the argument is not an extension known to the server, the server MUST ignore the argument.
- If the argument is an extension known to the server, and it is not specifically permitted to be enabled using ENABLE, the server MUST ignore the argument. (Note that knowing about an extension doesn’t necessarily imply supporting that extension.)
- If the argument is an extension that is supported by the server and that needs to be enabled, the server MUST enable the extension for the duration of the connection. Note that once an extension is enabled, there is no way to disable it.

If the ENABLE command is successful, the server MUST send an untagged ENABLED response Section 7.2.1.

Clients SHOULD only include extensions that need to be enabled by the server. For example, a client can enable IMAP4rev2 specific behaviour when both IMAP4rev1 and IMAP4rev2 are advertised in the CAPABILITY response. Future RFCs may add to this list.

The ENABLE command is only valid in the authenticated state, before any mailbox is selected. Clients MUST NOT issue ENABLE once they SELECT/EXAMINE a mailbox; however, server implementations don’t have to check that no mailbox is selected or was previously selected during the duration of a connection.

The ENABLE command can be issued multiple times in a session. It is additive; i.e., "ENABLE a b", followed by "ENABLE c" is the same as a single command "ENABLE a b c". When multiple ENABLE commands are issued, each corresponding ENABLED response SHOULD only contain extensions enabled by the corresponding ENABLE command.

There are no limitations on pipelining ENABLE. For example, it is possible to send ENABLE and then immediately SELECT, or a LOGIN immediately followed by ENABLE.

The server MUST NOT change the CAPABILITY list as a result of executing ENABLE; i.e., a CAPABILITY command issued right after an ENABLE command MUST list the same capabilities as a CAPABILITY
command issued before the ENABLE command. This is demonstrated in the following example:

```
C: t1 CAPABILITY
S: * CAPABILITY IMAP4rev1 ID LITERAL+ ENABLE X-GOOD-IDEA
S: t1 OK foo
C: t2 ENABLE CONDSTORE X-GOOD-IDEA
S: * ENABLED X-GOOD-IDEA
S: t2 OK foo
C: t3 CAPABILITY
S: * CAPABILITY IMAP4rev1 ID LITERAL+ ENABLE X-GOOD-IDEA
S: t3 OK foo again
```

In the following example, the client enables CONDSTORE:

```
C: a1 ENABLE CONDSTORE
S: * ENABLED CONDSTORE
S: a1 OK Conditional Store enabled
```

6.3.1.1. Note to Designers of Extensions That May Use the ENABLE Command

Designers of IMAP extensions are discouraged from creating extensions that require ENABLE unless there is no good alternative design. Specifically, extensions that cause potentially incompatible behavior changes to deployed server responses (and thus benefit from ENABLE) have a higher complexity cost than extensions that do not.

6.3.2. SELECT Command

Arguments: mailbox name

Responses: REQUIRED untagged responses: FLAGS, EXISTS
          REQUIRED OK untagged responses: PERMANENTFLAGS, UIDNEXT, UIDVALIDITY

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. Note that earlier versions of this protocol only required the FLAGS and EXISTS untagged data; consequently, client implementations SHOULD implement default behavior for missing data as discussed with the individual item.
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.

OK [PERMANENTFLAGS (<list of flags>)] A list of message flags that
the client can change permanently. If this is missing, the client
should assume that all flags can be changed permanently.

OK [UIDNEXT <n>] The next unique identifier value. Refer to
Section 2.3.1.1 for more information. If this is missing, the
client can not make any assumptions about the next unique
identifier value.

OK [UIDVALIDITY <n>] The unique identifier validity value. Refer to
Section 2.3.1.1 for more information. If this is missing, the
server does not support unique identifiers.

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. When deselecting a
selected mailbox, the server MUST return an untagged OK response with
the "[CLOSED]" response code when the currently selected mailbox is
closed (see Paragraph 10).

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:  * OK [UIDVALIDITY 3857529045] UIDs valid
S:  * OK [UIDNEXT 4392] Predicted next UID
S:  * FLAGS (\Answered \Flagged \Deleted \Seen \Draft)
S:  * OK [PERMANENTFLAGS (\Deleted \Seen \*)] Limited
S:  A142 OK [READ-WRITE] SELECT completed

6.3.3. EXAMINE Command

Arguments: mailbox name

Responses: REQUIRED untagged responses: FLAGS, EXISTS
           REQUIRED OK untagged responses: PERMANENTFLAGS,
           UIDNEXT, UIDVALIDITY

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:  * OK [UIDVALIDITY 3857529045] UIDs valid
S:  * OK [UIDNEXT 4392] Predicted next UID
S:  * FLAGS (\Answered \Flagged \Deleted \Seen \Draft)
S:  * OK [PERMANENTFLAGS ()] No permanent flags permitted
S:  A932 OK [READ-ONLY] EXAMINE completed

6.3.4. 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 a client attempts to create a UTF-8 mailbox name that is not a valid Net-Unicode name, the server MUST reject the creation or convert the name to Net-Unicode prior to creating the mailbox.

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 the declaration. In any case, the name created is without the trailing hierarchy delimiter.

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 be successfully completed. 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.

6.3.5.  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. If the server implementation does not permit deleting the name while inferior hierarchical names exists the \Noselect mailbox name attribute is set for that name. In any case, all messages in that mailbox are removed by the DELETE command.

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 () "/" blurybloop
S: * LIST (\Noselect)="/" foo
S: * LIST () "/" foo/bar
S: A682 OK LIST completed
C: A683 DELETE blurybloop
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 () "." blurybloop
S: * LIST () "." foo
S: * LIST () "." foo.bar
S: A82 OK LIST completed
C: A83 DELETE blurybloop
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

6.3.6. 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".

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

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

6.3.7. 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 a server site can choose to 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

6.3.8. 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

6.3.9. 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 the \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 the context in which the mailbox name is interpreted.

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 the empty string if the reference is non-rooted or is an empty string. In all cases, a hierarchy delimiter (or NIL if there is no hierarchy) is returned. This permits a client to get the hierarchy delimiter (or find out that the mailbox names are flat) even when no mailboxes by that name currently exist.

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

Note: The interpretation of the reference argument is implementation-defined. It depends upon whether the server implementation has a concept of the "current working directory" and leading "break out characters", which override the current working directory.

For example, on a server which exports a UNIX or NT filesystem, the reference argument contains the current working directory, and the mailbox name argument would contain the name as interpreted in the current working directory.

If a server implementation has no concept of break out characters, the canonical form is normally the reference name appended with the mailbox name. Note that if the server implements the namespace convention (Section 5.1.2.1), "#" is a break out character and must be treated as such.

If the reference argument is not a level of mailbox hierarchy (that is, it is a \NoInferiors name), and/or the reference argument does not end with the hierarchy delimiter, it is implementation-dependent how this is interpreted. For example, a reference of "foo/bar" and mailbox name of "rag/baz" could be interpreted as "foo/bar/rag/baz", "foo/barrag/baz", or "foo/rag/baz". A client SHOULD NOT use such a reference argument except at the explicit request of the user. A hierarchical browser MUST NOT make any assumptions about server interpretation of the reference
unless the reference is a level of mailbox hierarchy AND ends with
the hierarchy delimiter.

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:

<table>
<thead>
<tr>
<th>Reference</th>
<th>Mailbox Name</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>~smith/Mail/</td>
<td>foo.*</td>
<td>~smith/Mail/foo.*</td>
</tr>
<tr>
<td>archive/</td>
<td>%</td>
<td>archive/</td>
</tr>
<tr>
<td>#news.</td>
<td>comp.mail.*</td>
<td>#news.comp.mail.*</td>
</tr>
<tr>
<td>~smith/Mail/</td>
<td>/usr/doc/foo</td>
<td>/usr/doc/foo</td>
</tr>
<tr>
<td>archive/</td>
<td>-fred/Mail/*</td>
<td>-fred/Mail/*</td>
</tr>
</tbody>
</table>

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

6.3.10. 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.

The returned untagged LSUB response MAY contain different mailbox flags from a LIST untagged response. If this should happen, the flags in the untagged LIST are considered more authoritative.

A special situation occurs when using LSUB with the % wildcard. Consider what happens if "foo/bar" (with a hierarchy delimiter of "/") is subscribed but "foo" is not. A "%" wildcard to LSUB must return foo, not foo/bar, in the LSUB response, and it MUST be flagged with the \Noselect attribute.
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
C: A003 LSUB "#news." "comp.%"
S: * LSUB (\NoSelect) "." #news.comp.mail
S: A003 OK LSUB completed

6.3.11.  NAMESPACE Command

Arguments: none

Responses: REQUIRED untagged responses: NAMESPACE

Result: OK - command completed
        NO - Can’t complete the command
        BAD - arguments invalid

The NAMESPACE command causes a single untagged NAMESPACE response to be returned. The untagged NAMESPACE response contains the prefix and hierarchy delimiter to the server’s Personal Namespace(s), Other Users’ Namespace(s), and Shared Namespace(s) that the server wishes to expose. The response will contain a NIL for any namespace class that is not available. Namespace_Response_Extensions MAY be included in the response. Namespace_Response_Extensions which are not on the IETF standards track, MUST be prefixed with an "X-".

Example 1:

In this example a server supports a single personal namespace. No leading prefix is used on personal mailboxes and "/" is the hierarchy delimiter.

C: A001 NAMESPACE
S: * NAMESPACE ("" "/") NIL NIL
S: A001 OK NAMESPACE command completed

Example 2:

A user logged on anonymously to a server. No personal mailboxes are associated with the anonymous user and the user does not have access to the Other Users’ Namespace. No prefix is required to access shared mailboxes and the hierarchy delimiter is "."
Example 3:

A server that contains a Personal Namespace and a single Shared Namespace.

C: A001 NAMESPACE
S: * NAMESPACE NIL NIL ("" "/")
S: A001 OK NAMESPACE command completed

Example 4:

A server that contains a Personal Namespace, Other Users’ Namespace and multiple Shared Namespaces. Note that the hierarchy delimiter used within each namespace can be different.

C: A001 NAMESPACE
S: * NAMESPACE ("" "/") ("~" "/") ("#shared/ "/") ("#public/ "/") ("#ftp/ "/") ("#news. ".")
S: A001 OK NAMESPACE command completed

The prefix string allows a client to do things such as automatically creating personal mailboxes or LISTing all available mailboxes within a namespace.

Example 5:

A server that supports only the Personal Namespace, with a leading prefix of INBOX to personal mailboxes and a hierarchy delimiter of ".".

C: A001 NAMESPACE
S: * NAMESPACE ("INBOX." ".") NIL NIL
S: A001 OK NAMESPACE command completed

< Automatically create a mailbox to store sent items.>

C: A002 CREATE "INBOX.Sent Mail"
S: A002 OK CREATE command completed

Although typically a server will support only a single Personal Namespace, and a single Other User’s Namespace, circumstances exist where there MAY be multiples of these, and a client MUST be prepared for them. If a client is configured such that it is required to create a certain mailbox, there can be circumstances where it is
unclear which Personal Namespaces it should create the mailbox in. In these situations a client SHOULD let the user select which namespaces to create the mailbox in.

Example 6:

In this example, a server supports 2 Personal Namespaces. In addition to the regular Personal Namespace, the user has an additional personal namespace to allow access to mailboxes in an MH format mailstore.

The client is configured to save a copy of all mail sent by the user into a mailbox called ‘Sent Mail’. Furthermore, after a message is deleted from a mailbox, the client is configured to move that message to a mailbox called ‘Deleted Items’.

Note that this example demonstrates how some extension flags can be passed to further describe the #mh namespace.

C: A001 NAMESPACE
S: * NAMESPACE {("" "/");("mh/" "/" "X-PARAM" ("FLAG1" "FLAG2"))}
    NIL NIL
S: A001 OK NAMESPACE command completed

< It is desired to keep only one copy of sent mail. It is unclear which Personal Namespace the client should use to create the ‘Sent Mail’ mailbox. The user is prompted to select a namespace and only one ‘Sent Mail’ mailbox is created. >

C: A002 CREATE "Sent Mail"
S: A002 OK CREATE command completed

< The client is designed so that it keeps two ‘Deleted Items’ mailboxes, one for each namespace. >

C: A003 CREATE "Delete Items"
S: A003 OK CREATE command completed

C: A004 CREATE "#mh/Deleted Items"
S: A004 OK CREATE command completed

The next level of hierarchy following the Other Users’ Namespace prefix SHOULD consist of <username>, where <username> is a user name as per the LOGIN or AUTHENTICATE command.

A client can construct a LIST command by appending a "%" to the Other Users’ Namespace prefix to discover the Personal Namespaces of other users that are available to the currently authenticated user.
In response to such a LIST command, a server SHOULD NOT return user names that have not granted access to their personal mailboxes to the user in question.

A server MAY return a LIST response containing only the names of users that have explicitly granted access to the user in question.

Alternatively, a server MAY return NO to such a LIST command, requiring that a user name be included with the Other Users’ Namespace prefix before listing any other user’s mailboxes.

Example 7:

A server that supports providing a list of other user’s mailboxes that are accessible to the currently logged on user.

```
C: A001 NAMESPACE
S: * NAMESPACE ("" "/") ("Other Users/" "/") NIL
S: A001 OK NAMESPACE command completed

C: A002 LIST "" "Other Users/"
S: * LIST () "/" "Other Users/Mike"
S: * LIST () "/" "Other Users/Karen"
S: * LIST () "/" "Other Users/Matthew"
S: * LIST () "/" "Other Users/Tesa"
S: A002 OK LIST command completed
```

Example 8:

A server that does not support providing a list of other user’s mailboxes that are accessible to the currently logged on user. The mailboxes are listable if the client includes the name of the other user with the Other Users’ Namespace prefix.
C: A001 NAMESPACE
S: * NAMESPACE ("/") NIL
S: A001 OK NAMESPACE command completed

< In this example, the currently logged on user has access to the Personal Namespace of user Mike, but the server chose to suppress this information in the LIST response. However, by appending the user name Mike (received through user input) to the Other Users’ Namespace prefix, the client is able to get a listing of the personal mailboxes of user Mike. >

C: A002 LIST "" "/%"
S: A002 NO The requested item could not be found.

C: A003 LIST "" "/%"
S: * LIST () "/" "~/INBOX"
S: * LIST () "/" "~/Foo"
S: A003 OK LIST command completed.

A prefix string might not contain a hierarchy delimiter, because in some cases it is not needed as part of the prefix.

Example 9:

A server that allows access to the Other Users’ Namespace by prefixing the others’ mailboxes with a ‘~’ followed by <username>, where <username> is a user name as per the LOGIN or AUTHENTICATE command.

C: A001 NAMESPACE
S: * NAMESPACE ("/") ("~" "/") NIL
S: A001 OK NAMESPACE command completed

< List the mailboxes for user mark >

C: A002 LIST "" "/%"
S: * LIST () "/" "~/INBOX"
S: * LIST () "/" "~/foo"
S: A002 OK LIST command completed

6.3.12. STATUS Command

Arguments: mailbox name
status data item names

Responses: REQUIRED untagged responses: STATUS

Result: OK - status completed
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.

The STATUS command provides an alternative to opening a second IMAP4rev2 connection and doing an EXAMINE command on a mailbox to query that mailbox’s status without deselecting the current mailbox in the first IMAP4rev2 connection.

Unlike the LIST command, the STATUS command is not guaranteed to be fast in its response. Under certain circumstances, it can be quite slow. 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.

Note: The STATUS command is intended to access the status of mailboxes other than the currently selected mailbox. Because the STATUS command can cause the mailbox to be opened internally, and because this information is available by other means on the selected mailbox, the STATUS command SHOULD NOT be used on the currently selected mailbox.

The STATUS command MUST NOT be used as a "check for new messages in the selected mailbox" operation (refer to sections Section 7, Section 7.3.1 for more information about the proper method for new message checking).

Because the STATUS command is not guaranteed to be fast in its results, clients SHOULD NOT expect to be able to issue many consecutive STATUS commands and obtain reasonable performance.

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

MESSAGES The number of messages in the mailbox.

UIDNEXT The next unique identifier value of the mailbox. Refer to Section 2.3.1.1 for more information.

UIDVALIDITY The unique identifier validity value of the mailbox. Refer to Section 2.3.1.1 for more information.

UNSEEN The number of messages which do not have the \Seen flag set.
SIZE  The total size of the mailbox in octets. This is not strictly required to be an exact value, but it MUST be equal to or greater than the sum of the values of the RFC822.SIZE FETCH message data items (see Section 6.4.6) of all messages in the mailbox.

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

6.3.13.  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-5322] or [I18N-HDRS] 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-5322] 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 to 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.

On successful completion of an APPEND, the server SHOULD return an APPENDUID response code.

In the case of a mailbox that has permissions set so that the client can APPEND to the mailbox, but not SELECT or EXAMINE it, the server SHOULD NOT send an APPENDUID response code as it would disclose information about the mailbox.

In the case of a mailbox that has UIDNOTSTICKY status (see UIDNOTSTICKY response code definition), the server MAY omit the APPENDUID response code as it is not meaningful.

If the server does not return the APPENDUID response codes, the client can discover this information by selecting the destination mailbox. The location of messages placed in the destination mailbox by APPEND can be determined by using FETCH and/or SEARCH commands (e.g., for Message-ID or some unique marker placed in the message in an APPEND).

If the mailbox is currently selected, the normal new message 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}
S: + Ready for literal data
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
In this example, A003 and A004 demonstrate successful appending and copying to a mailbox that returns the UIDs assigned to the messages. A005 is an example in which no messages were copied; this is because in A003, we see that message 2 had UID 304, and message 3 had UID 319; therefore, UIDs 305 through 310 do not exist (refer to Section 2.3.1.1 for further explanation). A006 is an example of a message being copied that did not return a COPYUID; and, as expected, A007 shows that the mail store containing that mailbox does not support persistent UIDs.

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

6.3.14. IDLE Command

Arguments: none

Responses: continuation data will be requested; the client sends the continuation data "DONE" to end the command
Result:  OK - IDLE completed after client sent "DONE"
        NO - failure: the server will not allow the IDLE command
              at this time
        BAD - command unknown or arguments invalid

Without the IDLE command a client requires to poll the server for
changes to the selected mailbox (new mail, deletions, flag changes).
It’s often more desirable to have the server transmit updates to the
client in real time. This allows a user to see new mail immediately.
The IDLE command allows a client to tell the server that it’s ready
to accept such real-time updates.

The IDLE command is sent from the client to the server when the
client is ready to accept unsolicited mailbox update messages. The
server requests a response to the IDLE command using the continuation
("+") response. The IDLE command remains active until the client
responds to the continuation, and as long as an IDLE command is
active, the server is now free to send untagged EXISTS, EXPUNGE,
FETCH, and other responses at any time. If the server choose to send
unsolicited FETCH responses, they MUST include UID FETCH item.

The IDLE command is terminated by the receipt of a "DONE"
continuation from the client; such response satisfies the server’s
continuation request. At that point, the server MAY send any
remaining queued untagged responses and then MUST immediately send
the tagged response to the IDLE command and prepare to process other
commands. As in the base specification, the processing of any new
command may cause the sending of unsolicited untagged responses,
subject to the ambiguity limitations. The client MUST NOT send a
command while the server is waiting for the DONE, since the server
will not be able to distinguish a command from a continuation.

The server MAY consider a client inactive if it has an IDLE command
running, and if such a server has an inactivity timeout it MAY log
the client off implicitly at the end of its timeout period. Because
of that, clients using IDLE are advised to terminate the IDLE and re-
issue it at least every 29 minutes to avoid being logged off. This
still allows a client to receive immediate mailbox updates even
though it need only "poll" at half hour intervals.
In the 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, NAMESPACE, CREATE, DELETE, RENAME, SUBSCRIBE, UNSUBSCRIBE, LIST, LSUB, STATUS, and APPEND), the following commands are valid in the selected state: CHECK, CLOSE, UNSELECT, EXPUNGE, SEARCH, FETCH, STORE, COPY, MOVE, 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 message 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
BAD - command unknown or arguments invalid

The CLOSE command permanently removes all messages that have the \Deleted flag set from the currently selected mailbox, and returns to the authenticated state from the 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. UNSELECT Command

Arguments: none

Responses: no specific responses for this command

Result: OK - unselect completed, now in authenticated state
         BAD - no mailbox selected, or argument supplied but none permitted

The UNSELECT command frees server’s resources associated with the selected mailbox and returns the server to the authenticated state. This command performs the same actions as CLOSE, except that no messages are permanently removed from the currently selected mailbox.

Example:    C: A342 UNSELECT
             S: A342 OK Unselect completed

6.4.4. 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 all messages that have the \Deleted flag set from the currently selected mailbox. 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.

6.4.5. SEARCH Command

Arguments: OPTIONAL result specifier
OPTIONAL \[CHARSET\] specification
searching criteria (one or more)

Responses: REQUIRED untagged response: ESEARCH

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.

The SEARCH command may contain result options. Result options control what kind of information is returned about messages matching the search criteria in an untagged ESEARCH response. If no result option is specified or empty list of options is specified "()", ALL is assumed (see below). The order of individual options is arbitrary. Individual options may contain parameters enclosed in parentheses (*). If an option has parameters, they consist of atoms and/or strings and/or lists in a specific order. Any options not defined by extensions that the server supports must be rejected with a BAD response.

(*) - if an option has a mandatory parameter, which can always be represented as a number or a sequence-set, the option parameter does not need the enclosing (). See ABNF for more details.

This document specifies the following result options:

MIN

Return the lowest message number/UID that satisfies the SEARCH criteria.

If the SEARCH results in no matches, the server MUST NOT include the MIN result option in the ESEARCH response; however, it still MUST send the ESEARCH response.

MAX
Return the highest message number/UID that satisfies the SEARCH criteria.

If the SEARCH results in no matches, the server MUST NOT include the MAX result option in the ESEARCH response; however, it still MUST send the ESEARCH response.

ALL

Return all message numbers/UIDs that satisfy the SEARCH criteria using the sequence-set syntax. Note, the client MUST NOT assume that messages/UIDs will be listed in any particular order.

If the SEARCH results in no matches, the server MUST NOT include the ALL result option in the ESEARCH response; however, it still MUST send the ESEARCH response.

COUNT

Return number of the messages that satisfy the SEARCH criteria. This result option MUST always be included in the ESEARCH response.

Note: future extensions to this document can allow servers to return multiple ESEARCH responses for a single extended SEARCH command. However all options specified above MUST result in a single ESEARCH response. These extensions will have to describe how results from multiple ESEARCH responses are to be amalgamated.

Searching criteria consist of one or more search keys.

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-5322]/[MIME-IMB] headers, MUST be decoded before comparing text. US-ASCII and UTF-8 charsets MUST be supported; other [CHARSET]s MAY be supported. If "CHARSET" is not provided, an IMAP4rev2 server MUST assume UTF-8.
If the server does not support the specified [CHARSET], it MUST return a tagged NO response (not a BAD). This response SHOULD contain the BADCHARSET response code, which MAY list the [CHARSET]s supported by the server.

In all search keys that use strings, a message matches the key if the string is a substring of the associated text. The matching SHOULD be case-insensitive for characters within ASCII range. Consider using [IMAP-I18N] for language-sensitive case-insensitive searching. Note that the empty string is a substring; this is useful when doing a HEADER search in order to test for a header field presence in the message.

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

<sequence 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 (disregarding time and timezone) 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-5322]) and that contains the specified string in the text of the header (what comes after the colon). If the string to search is zero-length, this matches
all messages that have a header line with the specified field-name regardless of the contents.

KEYWORD <flag>  Messages with the specified keyword flag set.

LARGER <n>  Messages with an [RFC-5322] size larger than the specified number of octets.

NEW [[Fix this]] 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.

ON <date>  Messages whose internal date (disregarding time and timezone) is within the specified date.

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

SEEN  Messages that have the \Seen flag set.

SENTBEFORE <date>  Messages whose [RFC-5322] Date: header (disregarding time and timezone) is earlier than the specified date.

SENTON <date>  Messages whose [RFC-5322] Date: header (disregarding time and timezone) is within the specified date.

SENTSINCE <date>  Messages whose [RFC-5322] Date: header (disregarding time and timezone) is within or later than the specified date.

SINCE <date>  Messages whose internal date (disregarding time and timezone) is within or later than the specified date.

SMALLER <n>  Messages with an [RFC-5322] 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 <sequence set> Messages with unique identifiers corresponding to the specified unique identifier set. Sequence set ranges are permitted.

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 flag set.

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

Example: C: A282 SEARCH RETURN (MIN COUNT) FLAGGED SINCE 1-Feb-1994 NOT FROM "Smith"
S: * ESEARCH (TAG "A282") MIN 2 COUNT 3
S: A282 OK SEARCH completed

Example: C: A283 SEARCH RETURN () FLAGGED SINCE 1-Feb-1994 NOT FROM "Smith"
S: * ESEARCH (TAG "A283") ALL 2,10:11
S: A283 OK SEARCH completed

Example: C: A284 SEARCH TEXT "string not in mailbox"
S: * ESEARCH (TAG "A284")
S: A284 OK SEARCH completed
C: A285 SEARCH CHARSET UTF-8 TEXT {6}
S: + Ready for literal text
C: XXXXXX
S: * ESEARCH (TAG "A285") ALL 43
S: A285 OK SEARCH completed

Note: Since this document is restricted to 7-bit ASCII text, it is not possible to show actual UTF-8 data. The "XXXXXX" is a placeholder for what would be 6 octets of 8-bit data in an actual transaction.

The following example demonstrates finding the first unseen message in the mailbox:

Example: C: A284 SEARCH RETURN (MIN) UNSEEN
S: * ESEARCH (TAG "A284") MIN 4
S: A284 OK SEARCH completed
The following example demonstrates that if the ESEARCH UID indicator is present, all data in the ESEARCH response is referring to UIDs; for example, the MIN result specifier will be followed by a UID.

Example:    
C: A285 UID SEARCH RETURN (MIN MAX) 1:5000
S: * ESEARCH (TAG "A285") UID MIN 7 MAX 3800
S: A285 OK SEARCH completed

The following example demonstrates returning the number of deleted messages:

Example:    
C: A286 SEARCH RETURN (COUNT) DELETED
S: * ESEARCH (TAG "A286") COUNT 15
S: A286 OK SEARCH completed

6.4.6. FETCH Command

Arguments: sequence set
            message data item names or macro

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.

Most data items, identified in the formal syntax under the msg-att-static rule, are static and MUST NOT change for any particular message. Other data items, identified in the formal syntax under the msg-att-dynamic rule, MAY change, either as a result of a STORE command or due to external events.

For example, if a client receives an ENVELOPE for a message when it already knows the envelope, it can safely ignore the newly transmitted envelope.

There are three macros which specify commonly-used sets of data items, and can be used instead of data items. A macro must be used by itself, and not in conjunction with other macros or data items.

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

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

The currently defined data items that can be fetched are:

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 or MESSAGE/GLOBAL 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 or MESSAGE/GLOBAL. 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-5322] header of the message or of an encapsulated [MIME-IMT] MESSAGE/RFC822 or MESSAGE/GLOBAL message. HEADER.FIELDS and HEADER.FIELDS.NOT are followed by a list of field-name (as defined in [RFC-5322]) 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 ASCII range case-insensitive but otherwise exact. Subsetting does not exclude the [RFC-5322] delimiting blank line between the
header and the body; the blank line is included in all header
fetches, except in the case of a message which has no body and
no blank line.

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-5322] header.

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

HEADER     ([RFC-5322] header of the message)
TEXT       ([RFC-5322] text body of the message) MULTIPART/MIXED
1          TEXT/PLAIN
2          APPLICATION/OCTET-STREAM
3          MESSAGE/RFC822
3.HEADER   ([RFC-5322] header of the message)
3.TEXT     ([RFC-5322] text body of the message) MULTIPART/MIXED
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-5322] header of the message)
4.2.TEXT   ([RFC-5322] text body of the message) MULTIPART/MIXED
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-5322] header and [MIME-IMB] headers.

ENVELOPE The envelope structure of the message. This is computed by the server by parsing the [RFC-5322] header into the component parts, defaulting various fields as necessary.

FLAGS The flags that are set for this message.

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-5322] 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

6.4.7. STORE Command

Arguments: sequence 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 is 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 is 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 is returned as if a FETCH of those flags was done.

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

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

Arguments: sequence 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.

On successful completion of a COPY, the server SHOULD return a COPYUID response code.

In the case of a mailbox that has permissions set so that the client can COPY to the mailbox, but not SELECT or EXAMINE it, the server SHOULD NOT send an COPYUID response code as it would disclose information about the mailbox.

In the case of a mailbox that has UIDNOTSTICKY status (see the UIDNOTSTICKY response code), the server MAY omit the COPYUID response code as it is not meaningful.

If the server does not return the COPYUID response code, the client can discover this information by selecting the destination mailbox. The location of messages placed in the destination mailbox by COPY can be determined by using FETCH and/or SEARCH commands (e.g., for Message-ID).

Example:  C: A003 COPY 2:4 MEETING
          S: A003 OK COPY completed
6.4.9. MOVE Command

Arguments: sequence set
            mailbox name

Responses: no specific responses for this command

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

The MOVE command moves the specified message(s) to the end of the specified destination mailbox. The flags and internal date of the message(s) SHOULD be preserved.

This means that a new message is created in the target mailbox with a new UID, the original message is removed from the source mailbox, and it appears to the client as a single action. This has the same effect for each message as this sequence:

1. [UID] COPY

2. [UID] STORE +FLAGS.SILENT \DELETED

3. UID EXPUNGE

Although the effect of the MOVE is the same as the preceding steps, the semantics are not identical: The intermediate states produced by those steps do not occur, and the response codes are different. In particular, though the COPY and EXPUNGE response codes will be returned, response codes for a STORE MUST NOT be generated and the \Deleted flag MUST NOT be set for any message.

Because a MOVE applies to a set of messages, it might fail partway through the set. Regardless of whether the command is successful in moving the entire set, each individual message SHOULD either be moved or unaffected. The server MUST leave each message in a state where it is in at least one of the source or target mailboxes (no message can be lost or orphaned). The server SHOULD NOT leave any message in both mailboxes (it would be bad for a partial failure to result in a bunch of duplicate messages). This is true even if the server returns a tagged NO response to the command.

Because of the similarity of MOVE to COPY, extensions that affect COPY affect MOVE in the same way. Response codes such as TRYCREATE (see Section 7.1), as well as those defined by extensions, are sent as appropriate.
Servers SHOULD send COPYUID in response to a UID MOVE (see Section 6.4.10) command. For additional information see Section 7.1.

Servers are also advised to send the COPYUID response code in an untagged OK before sending EXPUNGE or moved responses. (Sending COPYUID in the tagged OK, as described in the UIDPLUS specification, means that clients first receive an EXPUNGE for a message and afterwards COPYUID for the same message. It can be unnecessarily difficult to process that sequence usefully.)

An example:

C: a UID MOVE 42:69 foo
S: * OK [COPYUID 432432 42:69 1202:1229]
S: * 22 EXPUNGE
S: (more expunges)
S: a OK Done

Note that the server may send unrelated EXPUNGE responses as well, if any happen to have been expunged at the same time; this is normal IMAP operation.

Note that moving a message to the currently selected mailbox (that is, where the source and target mailboxes are the same) is allowed when copying the message to the currently selected mailbox is allowed.

The server may send EXPUNGE responses before the tagged response, so the client cannot safely send more commands with message sequence number arguments while the server is processing MOVE.

MOVE and UID MOVE can be pipelined with other commands, but care has to be taken. Both commands modify sequence numbers and also allow unrelated EXPUNGE responses. The renumbering of other messages in the source mailbox following any EXPUNGE response can be surprising and makes it unsafe to pipeline any command that relies on message sequence numbers after a MOVE or UID MOVE. Similarly, MOVE cannot be pipelined with a command that might cause message renumbering. See Section 5.5, for more information about ambiguities as well as handling requirements for both clients and servers.

6.4.10. UID Command

Arguments: command name
command arguments

Responses: untagged responses: FETCH, ESEARCH

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

The UID command has three forms. In the first form, it takes as its arguments a COPY, MOVE, FETCH, or STORE command with arguments appropriate for the associated command. However, the numbers in the sequence set argument are unique identifiers instead of message sequence numbers. Sequence set ranges are permitted, but there is no guarantee that unique identifiers will be contiguous.

A non-existent unique identifier is ignored without any error message generated. Thus, it is possible for a UID FETCH command to return an OK without any data or a UID COPY, UID MOVE or UID STORE to return an OK without performing any operations.

In the second form, the UID command takes an EXPUNGE command with an extra parameter the specified a sequence set of UIDs to operate on. The UID EXPUNGE command permanently removes all messages that both have the \Deleted flag set and have a UID that is included in the specified sequence set from the currently selected mailbox. If a message either does not have the \Deleted flag set or has a UID that is not included in the specified sequence set, it is not affected.

UID EXPUNGE is particularly useful for disconnected use clients. By using UID EXPUNGE instead of EXPUNGE when resynchronizing with the server, the client can ensure that it does not inadvertently remove any messages that have been marked as \Deleted by other clients between the time that the client was last connected and the time the client resynchronizes.

Example:  
C: A003 UID EXPUNGE 3000:3002  
S: * 3 EXPUNGE  
S: * 3 EXPUNGE  
S: * 3 EXPUNGE  
S: A003 OK UID EXPUNGE completed

In the third 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 ESEARCH response for a UID SEARCH command are unique identifiers instead of message sequence numbers. Also, the corresponding ESEARCH response MUST include the UID indicator. For example, the command UID SEARCH 1:100 UID 443:557 returns the unique identifiers corresponding to the intersection of two sequence sets, the message sequence number range 1:100 and the UID range 443:557.

Note: in the above example, the UID range 443:557 appears. The same comment about a non-existent unique identifier being ignored
without any error message also applies here. Hence, even if neither UID 443 or 557 exist, this range is valid and would include an existing UID 495.

Also note that a UID range of 559:* always includes the UID of the last message in the mailbox, even if 559 is higher than any assigned UID value. This is because the contents of a range are independent of the order of the range endpoints. Thus, any UID range with * as one of the endpoints indicates at least one message (the message with the highest numbered UID), unless the mailbox is empty.

The number after the "*" in an untagged FETCH or EXPUNGE 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.

Note: The rule about including the UID message data item as part of a FETCH response primarily applies to the UID FETCH and UID STORE commands, including a UID FETCH command that does not include UID as a message data item. Although it is unlikely that the other UID commands will cause an untagged FETCH, this rule applies to these commands as well.

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 OK 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 IMAP4rev2 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 the selected state. In the 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 response 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 can 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.

**ALREADYEXISTS**

The operation attempts to create something that already exists, such as when the CREATE or RENAME directories attempt to create a mailbox and there is already one of that name.

C: o RENAME this that
S: o NO [ALREADYEXISTS] Mailbox "that" already exists

**APPENDUID**
Followed by the UIDVALIDITY of the destination mailbox and the UID assigned to the appended message in the destination mailbox, indicates that the message has been appended to the destination mailbox with that UID.

If the server also supports the [MULTIAPPEND] extension, and if multiple messages were appended in the APPEND command, then the second value is a UID set containing the UIDs assigned to the appended messages, in the order they were transmitted in the APPEND command. This UID set may not contain extraneous UIDs or the symbol "*".

Note: the UID set form of the APPENDUID response code MUST NOT be used if only a single message was appended. In particular, a server MUST NOT send a range such as 123:123. This is because a client that does not support [MULTIAPPEND] expects only a single UID and not a UID set.

UIDs are assigned in strictly ascending order in the mailbox (refer to Section 2.3.1.1); note that a range of 12:10 is exactly equivalent to 10:12 and refers to the sequence 10,11,12.

This response code is returned in a tagged OK response to the APPEND command.

AUTHENTICATIONFAILED

Authentication failed for some reason on which the server is unwilling to elaborate. Typically, this includes "unknown user" and "bad password".

This is the same as not sending any response code, except that when a client sees AUTHENTICATIONFAILED, it knows that the problem wasn’t, e.g., UNAVAILABLE, so there’s no point in trying the same login/password again later.

C: b LOGIN "fred" "foo"
S: b NO [AUTHENTICATIONFAILED] Authentication failed

AUTHORIZATIONFAILED Authentication succeeded in using the authentication identity, but the server cannot or will not allow the authentication identity to act as the requested authorization identity. This is only applicable when the authentication and authorization identities are different. C: c1 AUTHENTICATE PLAIN [...]
S: c1 NO [AUTHORIZATIONFAILED] No such authorization-ID
C: c2 AUTHENTICATE PLAIN
   [...] 
S: c2 NO [AUTHORIZATIONFAILED] Authenticator is not an admin

BADCHARSET Optionally followed by a parenthesized list of charsets. 
A SEARCH failed because the given charset is not supported by this 
implementation. If the optional list of charsets is given, this 
lists the charsets that are supported by this implementation.

CANNOT
The operation violates some invariant of the server and can 
ever succeed.

C: l create "/////"
S: l NO [CANNOT] Adjacent slashes are not supported

CAPABILITY Followed by a list of capabilities. This can appear in 
the initial OK or PREAUTH response to transmit an initial 
capabilities list. This makes it unnecessary for a client to send 
a separate CAPABILITY command if it recognizes this response.

CLIENTBUG
The server has detected a client bug. This can accompany all 
of OK, NO, and BAD, depending on what the client bug is.

C: k1 select "/archive/projects/experiment-iv"
   [...] 
S: k1 OK [READ-ONLY] Done
C: k2 status "/archive/projects/experiment-iv" (messages)
   [...] 
S: k2 OK [CLIENTBUG] Done

CLOSED
The CLOSED response code has no parameters. A server return 
the CLOSED response code when the currently selected mailbox is 
closed implicitly using the SELECT/EXAMINE command on another 
mailbox. The CLOSED response code serves as a boundary between 
responses for the previously opened mailbox (which was closed) 
and the newly selected mailbox; all responses before the CLOSED 
response code relate to the mailbox that was closed, and all 
subsequent responses relate to the newly opened mailbox.

There is no need to return the CLOSED response code on 
completion of the CLOSE or the UNSELECT command (or similar),
whose purpose is to close the currently selected mailbox without opening a new one.

The server can also return an unsolicited CLOSED response code when it wants to force the client to return to authenticated state. For example, the server can do that when the mailbox requires repairs or is deleted in another session.

**CONTACTADMIN**

The user should contact the system administrator or support desk.

C: e login "fred" "foo"
S: e OK [CONTACTADMIN]

**COPYUID**

Followed by the UIDVALIDITY of the destination mailbox, a UID set containing the UIDs of the message(s) in the source mailbox that were copied to the destination mailbox and containing the UIDs assigned to the copied message(s) in the destination mailbox, indicates that the message(s) have been copied to the destination mailbox with the stated UID(s).

The source UID set is in the order the message(s) were copied; the destination UID set corresponds to the source UID set and is in the same order. Neither of the UID sets may contain extraneous UIDs or the symbol "*".

UIDs are assigned in strictly ascending order in the mailbox (refer to Section 2.3.1.1); note that a range of 12:10 is exactly equivalent to 10:12 and refers to the sequence 10,11,12.

This response code is returned in a tagged OK response to the COPY command.

**CORRUPTION**

The server discovered that some relevant data (e.g., the mailbox) are corrupt. This response code does not include any information about what’s corrupt, but the server can write that to its logfiles.

C: i select "/archive/projects/experiment-iv"
S: i NO [CORRUPTION] Cannot open mailbox
Either authentication succeeded or the server no longer had the necessary data; either way, access is no longer permitted using that passphrase. The client or user should get a new passphrase.

C: d login "fred" "foo"
S: d NO [EXPIRED] That password isn’t valid any more

Someone else has issued an EXPUNGE for the same mailbox. The client may want to issue NOOP soon. [IMAP-MULTIACCESS] discusses this subject in depth.

C: h search from fred@example.com
S: * SEARCH 1 2 3 5 8 13 21 42
S: h OK [EXPUNGEISSUED] Search completed

An operation has not been carried out because it involves sawing off a branch someone else is sitting on. Someone else may be holding an exclusive lock needed for this operation, or the operation may involve deleting a resource someone else is using, typically a mailbox.

The operation may succeed if the client tries again later.

C: g delete "/archive/projects/experiment-iv"
S: g NO [INUSE] Mailbox in use

The operation ran up against an implementation limit of some kind, such as the number of flags on a single message or the number of flags used in a mailbox.

C: m STORE 42 FLAGS f1 f2 f3 f4 f5 ... f250
S: m NO [LIMIT] At most 32 flags in one mailbox supported

The operation attempts to delete something that does not exist. Similar to ALREADYEXISTS.

C: p RENAME this that
S: p NO [NONEXISTENT] No such mailbox

NOPERM

The access control system (e.g., Access Control List (ACL), see [RFC4314]) does not permit this user to carry out an operation, such as selecting or creating a mailbox.

C: f select "/archive/projects/experiment-iv"
S: f NO [NOPERM] Access denied

OVERQUOTA

The user would be over quota after the operation. (The user may or may not be over quota already.)

Note that if the server sends OVERQUOTA but doesn’t support the IMAP QUOTA extension defined by [RFC2087], then there is a quota, but the client cannot find out what the quota is.

C: n1 uid copy 1:* oldmail
S: n1 NO [OVERQUOTA] Sorry

C: n2 uid copy 1:* oldmail
S: n2 OK [OVERQUOTA] You are now over your soft quota

PARSE  The human-readable text represents an error in parsing the [RFC-5322] 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 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 ignore the change or store the state change 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.

PRIVACYREQUIRED

The operation is not permitted due to a lack of privacy. If Transport Layer Security (TLS) is not in use, the client could try STARTTLS (see Section 6.2.1) and then repeat the operation.
Internet-Draft                  IMAP4rev2                  February 2019

C: d login "fred" "foo"
S: d NO [PRIVACYREQUIRED] Connection offers no privacy

C: d select inbox
S: d NO [PRIVACYREQUIRED] Connection offers no privacy

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.

SERVERBUG

The server encountered a bug in itself or violated one of its
own invariants.

C: j select "/archive/projects/experiment-iv"
S: j NO [SERVERBUG] This should not happen

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.

UIDNEXT Followed by a decimal number, indicates the next unique
identifier value. Refer to Section 2.3.1.1 for more information.

UIDNOTSTICKY

The selected mailbox is supported by a mail store that does not
support persistent UIDs; that is, UIDVALIDITY will be different
each time the mailbox is selected. Consequently, APPEND or
COPY to this mailbox will not return an APPENDUID or COPYUID
response code.

This response code is returned in an untagged NO response to
the SELECT command.

Note: servers SHOULD NOT have any UIDNOTSTICKY mail stores.
This facility exists to support legacy mail stores in which
it is technically infeasible to support persistent UIDs.
This should be avoided when designing new mail stores.
UIDVALIDITY  Followed by a decimal number, indicates the unique identifier validity value. Refer to Section 2.3.1.1 for more information.

UNAVAILABLE

Temporary failure because a subsystem is down. For example, an IMAP server that uses a Lightweight Directory Access Protocol (LDAP) or Radius server for authentication might use this response code when the LDAP/Radius server is down.

C: a LOGIN "fred" "foo"
S: a NO [UNAVAILABLE] User’s backend down for maintenance

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 or an AUTHENTICATE command is needed.

Example: S: * OK IMAP4rev2 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; thus no LOGIN/
AUTHENTICATE command is needed.

Example:
S: * PREAUTH IMAP4rev2 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. In all cases the client SHOULD continue to read response data from the server until the connection is closed; this will ensure that any pending untagged or completion responses are read and processed.

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. The ENABLED Response

Contents: capability listing

The ENABLED response occurs as a result of an ENABLE command. The capability listing contains a space-separated listing of capability names that the server supports and that were successfully enabled.
The ENABLED response may contain no capabilities, which means that no extensions listed by the client were successfully enabled.

7.2.2. 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 "IMAP4rev2".

In addition, client and server implementations MUST implement the STARTTLS, LOGINDISABLED, and AUTH=PLAIN (described in [PLAIN]) capabilities. See the Security Considerations section for important information.

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

The LOGINDISABLED capability indicates that the LOGIN command is disabled, and that the server will respond with a tagged NO response to any attempt to use the LOGIN command even if the user name and password are valid. An IMAP client MUST NOT issue the LOGIN command if the server advertises the LOGINDISABLED capability.

Other capability names indicate that the server supports an extension, revision, or amendment to the IMAP4rev2 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 IMAP4rev2 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 "IMAP4rev2", and MUST ignore any unknown capability names.

A server MAY send capabilities automatically, by using the CAPABILITY response code in the initial PREAUTH or OK responses, and by sending an updated CAPABILITY response code in the tagged OK response as part of a successful authentication. It is unnecessary for a client to send a separate CAPABILITY command if it recognizes these automatic capabilities.

Example: S: * CAPABILITY IMAP4rev2 STARTTLS AUTH=GSSAPI XPIG-LATIN
7.2.3. 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.

The following base 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.

\HasChildren The presence of this attribute indicates that the mailbox has child mailboxes. A server SHOULD NOT set this attribute if there are child mailboxes and the user does not have permission to access any of them. In this case, \HasNoChildren SHOULD be used. In many cases, however, a server may not be able to efficiently compute whether a user has access to any child mailbox. Note that even though the \HasChildren attribute for a mailbox must be correct at the time of processing of the mailbox, a client must be prepared to deal with a situation when a mailbox is marked with the \HasChildren attribute, but no child mailbox appears in the response to the LIST command. This might happen, for example, due to children mailboxes being deleted or made inaccessible to the user (using access control) by another client before the server is able to list them.

\HasNoChildren The presence of this attribute indicates that the mailbox has NO child mailboxes that are accessible to the currently authenticated user.

\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.

It is an error for the server to return both a \HasChildren and a \HasNoChildren attribute in the same LIST response.
Note: the \HasNoChildren attribute should not be confused with the \NoInferiors attribute, which indicates that no child mailboxes exist now and none can be created in the future.

If it is not feasible for the server to determine whether or not the mailbox is "interesting", the server SHOULD NOT send either \Marked or \Unmarked. The server MUST NOT send more than one of \Marked, \Unmarked, and \Noselect for a single mailbox, and MAY send none of these.

In addition to the base name attributes defined above, an IMAP server MAY also include any or all of the following attributes that denote "role" (or "special-use") of a mailbox. These attributes are included along with base attributes defined above. A given mailbox may have none, one, or more than one of these attributes. In some cases, a special use is advice to a client about what to put in that mailbox. In other cases, it’s advice to a client about what to expect to find there.

\All This mailbox presents all messages in the user’s message store. Implementations MAY omit some messages, such as, perhaps, those in \Trash and \Junk. When this special use is supported, it is almost certain to represent a virtual mailbox.

\Archive This mailbox is used to archive messages. The meaning of an "archival" mailbox is server-dependent; typically, it will be used to get messages out of the inbox, or otherwise keep them out of the user’s way, while still making them accessible.

\Drafts This mailbox is used to hold draft messages -- typically, messages that are being composed but have not yet been sent. In some server implementations, this might be a virtual mailbox, containing messages from other mailboxes that are marked with the "\Draft" message flag. Alternatively, this might just be advice that a client put drafts here.

\Flagged This mailbox presents all messages marked in some way as "important". When this special use is supported, it is likely to represent a virtual mailbox collecting messages (from other mailboxes) that are marked with the "\Flagged" message flag.

\Junk This mailbox is where messages deemed to be junk mail are held. Some server implementations might put messages here automatically. Alternatively, this might just be advice to a client-side spam filter.

\Sent This mailbox is used to hold copies of messages that have been sent. Some server implementations might put messages here
automatically. Alternatively, this might just be advice that a client save sent messages here.

\Trash This mailbox is used to hold messages that have been deleted or marked for deletion. In some server implementations, this might be a virtual mailbox, containing messages from other mailboxes that are marked with the "\Deleted" message flag. Alternatively, this might just be advice that a client that chooses not to use the IMAP "\Deleted" model should use this as its trash location. In server implementations that strictly expect the IMAP "\Deleted" model, this special use is likely not to be supported.

All of special-use attributes are OPTIONAL, and any given server or message store may support any combination of the attributes, or none at all. In most cases, there will likely be at most one mailbox with a given attribute for a given user, but in some server or message store implementations it might be possible for multiple mailboxes to have the same special-use attribute.

Special-use attributes are likely to be user-specific. User Adam might share his \Sent mailbox with user Barb, but that mailbox is unlikely to also serve as Barb’s \Sent mailbox.

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

7.2.4. 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:  \texttt{S:* LSUB () "." #news.comp.mail.misc}

7.2.5. NAMESPACE Response

Contents: the prefix and hierarchy delimiter to the server’s
Personal Namespace(s), Other Users’ Namespace(s), and
Shared Namespace(s)

The NAMESPACE response occurs as a result of a NAMESPACE command. It
contains the prefix and hierarchy delimiter to the server’s Personal
Namespace(s), Other Users’ Namespace(s), and Shared Namespace(s) that
the server wishes to expose. The response will contain a NIL for any
namespace class that is not available. \texttt{Namespace\_Response\_Extensions}
\texttt{MAY} be included in the response. \texttt{Namespace\_Response\_Extensions} which
are not on the IETF standards track, MUST be prefixed with an \texttt{"X-"}.

Example:  \texttt{S:* NAMESPACE ("" "/") ("~" "/") NIL}

7.2.6. 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:  \texttt{S:* STATUS blurdybloop (MESSAGES 231 UIDNEXT 44292)}

7.2.7. ESEARCH Response

Contents: one or more search-return-data pairs

The ESEARCH response occurs as a result of a SEARCH or UID SEARCH
command.

The ESEARCH response starts with an optional search correlator. If
it is missing, then the response was not caused by a particular IMAP
command, whereas if it is present, it contains the tag of the command
that caused the response to be returned.

The search correlator is followed by an optional UID indicator. If
this indicator is present, all data in the ESEARCH response refers to
UIDs, otherwise all returned data refers to message numbers.

The rest of the ESEARCH response contains one or more search data
pairs. Each pair starts with unique return item name, followed by a
space and the corresponding data. Search data pairs may be returned
in any order. Unless specified otherwise by an extension, any return item name SHOULD appear only once in an ESEARCH response.

[[TBD: describe the most common search data pairs returned.]]

Example:    S: * ESEARCH UID COUNT 5 ALL 4:19,21,28
Example:    S: * ESEARCH (TAG "a567") UID COUNT 5 ALL 4:19,21,28
Example:    S: * ESEARCH COUNT 5 ALL 1:17,21

7.2.8.  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 transmitted 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 messages).

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

Example:    S: * 23 EXISTS
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).

The EXPUNGE response also decrements the number of messages in the mailbox; it is not necessary to send an EXISTS response with the new value.

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. A command is not "in progress" until the complete command has been received; in particular, a command is not "in progress" during the negotiation of command continuation.

Note: UID FETCH, UID STORE, and UID SEARCH are different commands from FETCH, STORE, and SEARCH. An EXPUNGE response MAY be sent during a UID command.

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.

Note: The origin octet facility MUST NOT be used by a server in a FETCH response unless the client specifically requested it by means of a FETCH of a BODY[<section>]<<partial>> data item.

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 or MESSAGE/GLOBAL part), MAY be in UTF-8. Note also that the [RFC-5322] delimiting blank line between the header and the body is not affected by header line subsetting; the blank line is always included as part of header data, except in the case of a message which has no body and no blank line.

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 sequence of one or more nested body structures. 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 BASE64-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 as defined in [DISPOSITION].

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

body location  A string giving the body content URI as defined in [LOCATION].

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].

body location  A string giving the body content URI as defined in [LOCATION].

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-5322] 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-5322] 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.

If the Date, Subject, In-Reply-To, and Message-ID header lines are absent in the [RFC-5322] header, the corresponding member of the envelope is NIL; if these header lines are present but empty the corresponding member of the envelope is the empty string.
Note: some servers may return a NIL envelope member in the "present but empty" case. Clients SHOULD treat NIL and empty string as identical.

Note: [RFC-5322] requires that all messages have a valid Date header. Therefore, the date member in the envelope cannot be NIL or the empty string.

Note: [RFC-5322] requires that the In-Reply-To and Message-ID headers, if present, have non-empty content. Therefore, the in-reply-to and message-id members in the envelope cannot be the empty string.

If the From, To, Cc, and Bcc header lines are absent in the [RFC-5322] header, or are present but empty, the corresponding member of the envelope is NIL.

If the Sender or Reply-To lines are absent in the [RFC-5322] header, or are present but empty, the server sets the corresponding member of the envelope to be the same value as the from member (the client is not expected to know to do this).

Note: [RFC-5322] requires that all messages have a valid From header. Therefore, the from, sender, and reply-to members in the envelope cannot be NIL.

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[HEADER]. Note that this did not result in \Seen being set, because RFC822.HEADER response data occurs as a result of a FETCH of RFC822.HEADER. BODY[HEADER] response data occurs as a result of a FETCH of BODY[HEADER] (which sets \Seen) or BODY.PEEK[HEADER] (which does not set \Seen).

RFC822.SIZE A number expressing the [RFC-5322] 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 AUTHENTICATE 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 synchronizing literal.

The client is not permitted to send the octets of the synchronizing literal unless the server indicates that it is expected. 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 IMAP4rev2 connection

The following is a transcript of an IMAP4rev2 connection. A long line in this sample is broken for editorial clarity.
S:   * OK IMAP4rev2 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:   * 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)"
"IMAP4rev2 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" "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] (342)
S:   Date: Wed, 17 Jul 1996 02:23:25 -0700 (PDT)
S:   From: Terry Gray <gray@cac.washington.edu>
S:   Subject: IMAP4rev2 WG mtg summary and minutes
S:   To: imap@cac.washington.edu
S:   cc: minutes@CNRI.Reston.VA.US, John Klensin <KLENSIN@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 IMAP4rev2 server terminating connection
S:   a006 OK LOGOUT completed

9. Formal Syntax

The following syntax specification uses the Augmented Backus-Naur Form (ABNF) notation as specified in [ABNF].
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" can be parsed as a flag-extension. Some, but not all, instances of this rule are noted below.

Note: [ABNF] rules MUST be followed strictly; in particular:

(1) 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.

(2) In all cases, SP refers to exactly one space. It is NOT permitted to substitute TAB, insert additional spaces, or otherwise treat SP as being equivalent to LWSP.

(3) The ASCII NULL character, %x00, MUST NOT be used at any time.

```plaintext
address  = "(" addr-name SP addr-adl SP addr-mailbox SP addr-host ")"

addr-adl = nstring
         ; Holds route from [RFC-5322] route-addr if non-NIL

addr-host = nstring
           ; NIL indicates [RFC-5322] group syntax.
           ; Otherwise, holds [RFC-5322] domain name

addr-mailbox = nstring
               ; NIL indicates end of [RFC-5322] group; if
               ; non-NIL and addr-host is NIL, holds
               ; [RFC-5322] group name.
               ; Otherwise, holds [RFC-5322] local-part
               ; after removing [RFC-5322] quoting

addr-name = nstring
          ; If non-NIL, holds phrase from [RFC-5322]
          ; mailbox after removing [RFC-5322] quoting

append  = "APPEND" SP mailbox [SP flag-list] [SP date-time] SP literal

append-uid = uniqueid

astring  = 1*ASTRING-CHAR / string
```
ASTRING-CHAR = ATOM-CHAR / resp-specials

atom = 1*ATOM-CHAR

ATOM-CHAR = <any CHAR except atom-specials>

atom-specials = "(" / ")" / "(" / SP / CTL / list-wildcards / quoted-specials / resp-specials

authenticate = "AUTHENTICATE" SP auth-type [SP initial-resp] *(CRLF base64)

auth-type = atom
            ; Defined by [SASL]

base64 = *(4base64-char) [base64-terminal]

base64-char = ALPHA / DIGIT / "+" / "/"
            ; Case-sensitive

base64-terminal = (2base64-char "==") / (3base64-char ")"

body = "(" (body-type-1part / body-type-mpart) ")"

body-extension = nstring / number /
            "(" body-extension *(SP 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 [SP body-fld-dsp [SP body-fld-lang [SP body-fld-loc *(SP body-extension)]]]
            ; MUST NOT be returned on non-extensible
            ; "BODY" fetch

body-ext-mpart = body-fld-param [SP body-fld-dsp [SP body-fld-lang [SP body-fld-loc *(SP body-extension)]]]
            ; MUST NOT be returned on non-extensible
            ; "BODY" fetch

body-fields = body-fld-param SP body-fld-id SP body-fld-desc SP
                body-fld-enc SP body-fld-octets

body-fld-desc = nstring
body-fld-dsp = "(" string SP body-fld-param ")" / nil

body-fld-enc = (DQUOTE ("7BIT" / "8BIT" / "BINARY" / "BASE64"/
  "QUOTED-PRINTABLE") DQUOTE) / string

body-fld-id = nstring

body-fld-lang = nstring / "(" string *(SP string) ")"

body-fld-loc = nstring

body-fld-lines = number

body-fld-md5 = nstring

body-fld-octets = number

body-fld-param = "(" string SP string *(SP string SP string) ")" / nil

body-type-1part = (body-type-basic / body-type-msg / body-type-text)
  [SP body-ext-1part]

body-type-basic = media-basic SP body-fields
  ; MESSAGE subtype MUST NOT be "RFC822" or "GLOBAL"

body-type-multipart = 1*body SP media-subtype
  [SP body-ext-multipart]
  ; MULTIPART body part

body-type-msg = media-message SP body-fields SP envelope
  SP body SP body-fld-lines

body-type-text = media-text SP body-fields SP 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" *(SP capability) SP "IMAP4rev2"
  *(SP capability)
  ; Servers MUST implement the STARTTLS, AUTH=PLAIN,
  ; and LOGINDISABLED capabilities
  ; Servers which offer RFC 1730 compatibility MUST
  ; list "IMAP4" as the first capability.

CHAR8 = %x01-ff
  ; any OCTET except NUL, %x00
charset         = atom / quoted
command         = tag SP (command-any / command-auth / command-nonauth / command-select) CRLF
                 ; Modal based on state
command-any     = "CAPABILITY" / "LOGOUT" / "NOOP" / enable / x-command
                 ; Valid in all states
command-auth    = append / create / delete / examine / list / lsub / Namespace-Command /
                 rename / select / status / subscribe / unsubscribe / idle
                 ; Valid only in Authenticated or Selected state
command-nonauth = login / authenticate / "STARTTLS"
                 ; Valid only when in Not Authenticated state
command-select  = "CHECK" / "CLOSE" / "UNSELECT" / "EXPUNGE" / copy /
                 move / fetch / store / search / uid
                 ; Valid only when in Selected state
continue-req    = "+" SP (resp-text / base64) CRLF
copy            = "COPY" SP sequence-set SP mailbox
create          = "CREATE" SP mailbox
                 ; Use of INBOX gives a NO error
date            = date-text / DQUOTE date-text DQUOTE
date-day        = 1*2DIGIT
                 ; Day of month
date-day-fixed  = (SP DIGIT) / 2DIGIT
                 ; Fixed-format version of date-day
date-text       = date-day "-" date-month "-" date-year
date-year       = 4DIGIT
date-time       = DQUOTE date-day-fixed "-" date-month "-" date-year
                 SP time SP zone DQUOTE
delete          = "DELETE" SP mailbox
digit-nz = %x31-39 ; 1-9
enable = "ENABLE" 1*(SP capability)
enable-data = "ENABLED" *(SP capability)
envelope = "(" env-date SP env-subject SP env-from SP env-sender SP env-reply-to SP env-to SP env-cc SP env-bcc SP env-in-reply-to SP env-message-id ")"
env-bcc = "(" 1*address ")" / nil
env-cc = "(" 1*address ")" / nil
e-date = nstring
e-mailbox = nstring
e-in-reply-to = nstring
e-message-id = nstring
e-reply-to = "(" 1*address ")" / nil
e-sender = "(" 1*address ")" / nil
e-subject = nstring
e-to = "(" 1*address ")" / nil
esearch-response = "ESEARCH" [search-correlator] [SP "UID"]/ *(SP search-return-data)
; ESEARCH response replaces SEARCH response
; from IMAP4rev1.
examine = "EXAMINE" SP mailbox
fetch = "FETCH" SP sequence-set SP ("ALL" / "FULL" / "FAST" /
fetch-att / "(" fetch-att *(SP fetch-att) ")")
fetch-att = "ENVELOPE" / "FLAGS" / "INTERNALDATE" /
"RFC822" [".HEADER" / ".SIZE" / ".TEXT"] /
"BODY" ["STRUCTURE"] / "UID" /
"BODY" section [partial] /
"BODY.PEEK" section [partial]
flag = "\Answered" / "\Flagged" / "\Deleted" / "\Seen" / "\Draft" / flag-keyword / flag-extension
; Does not include "\Recent"

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.
; "\Recent" was defined in RFC 3501
; and is now deprecated.

flag-fetch = flag
flag-keyword = "$MDNSent" / "$Forwarded" / atom
flag-list = "(" [flag *(SP flag)] ")"
flag-perm = flag / "\*"
greeting = "*" SP (resp-cond-auth / resp-cond-bye) CRLF

header-fld-name = astring
header-list = "(" header-fld-name *(SP header-fld-name) ")"

idle = "IDLE" CRLF "DONE"

initial-resp = (base64 / ":=
; "initial response" defined in
; Section 5.1 of [RFC4422]

list = "LIST" SP mailbox SP list-mailbox
list-mailbox = 1*list-char / string

list-char = ATOM-CHAR / list-wildcards / resp-specials

list-wildcards = "%" / "**"

literal = "{" number ["+"] "}" CRLF *CHAR8
; Number represents the number of CHAR8s.
; A non-synchronizing literal is distinguished from
; a synchronizing literal by presence of the "+"
; before the closing "}".
; Non synchronizing literals are not allowed when
Internet-Draft                  IMAP4rev2                  February 2019

; sent from server to the client.

login   = "LOGIN" SP userid SP password
lsub    = "LSUB" SP mailbox SP 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. An astring which consists of
       ; the case-insensitive sequence "I" "N" "B" "O" "X"
       ; is considered to be INBOX and not an astring.
       ; Refer to section 5.1 for further
       ; semantic details of mailbox names.
mailbox-data = "FLAGS" SP flag-list / "LIST" SP mailbox-list / "LSUB" SP mailbox-list / esearch-response / "STATUS" SP mailbox SP "(" [status-att-list] ")" / number SP "EXISTS" / Namespace-Response
mailbox-list = "(" [mbx-list-flags] ")" SP (DQUOTE QUOTED-CHAR DQUOTE / nil) SP mailbox
mbx-list-flags = *(mbx-list-oflag SP) mbx-list-sflag
                *(SP mbx-list-oflag) /
                mbx-list-oflag *(SP mbx-list-oflag)
mbx-list-oflag = "\Noinferiors" / flag-extension
                ; Other flags; multiple possible per LIST response
mbx-list-sflag = "\Noselect" / "\Marked" / "\Unmarked"
                ; Selectability flags; only one per LIST response
media-basic = ((DQUOTE ("APPLICATION" / "AUDIO" / "IMAGE" / "MESSAGE" / "VIDEO" / "FONT") DQUOTE) / string) SP media-subtype
             ; Defined in [MIME-IMT].
             ; FONT defined in RFC YYYY.
media-message = DQUOTE "MESSAGE" DQUOTE SP
                DQUOTE ("RFC822" / "GLOBAL") DQUOTE
                ; Defined in [MIME-IMT]
media-subtype = string
               ; Defined in [MIME-IMT]
media-text = DQUOTE "TEXT" DQUOTE SP media-subtype
             ; Defined in [MIME-IMT]
message-data = nz-number SP ("EXPUNGE" / ("FETCH" SP msg-att))
move = "MOVE" SP sequence-set SP mailbox
msg-att = "(" (msg-att-dynamic / msg-att-static) *(SP (msg-att-dynamic / msg-att-static))) ")"
msg-att-dynamic = "FLAGS" SP "(" [flag-fetch *(SP flag-fetch)] ")"
  ; MAY change for a message
msg-att-static = "ENVELOPE" SP envelope / "INTERNALDATE" SP date-time / "RFC822" [".HEADER" / ".TEXT"] SP nstring / "RFC822.SIZE" SP number / "BODY" ["STRUCTURE"] SP body / "BODY" section ["<" number ">"] SP nstring / "UID" SP uniqueid
  ; MUST NOT change for a message
Namespace = nil / "(" 1*Namespace-Descr ")"
Namespace-Command = "NAMESPACE"
Namespace-Descr = "(" string SP (DQUOTE QUOTED-CHAR DQUOTE / nil) *(Namespace-Response-Extension) ")"
Namespace-Response-Extension = SP string SP "(" string *(SP string) ")"
Namespace-Response = "NAMESPACE" SP Namespace
  SP Namespace SP Namespace
  ; The first Namespace is the Personal Namespace(s)
  ; The second Namespace is the Other Users’ Namespace(s)
  ; The third Namespace is the Shared Namespace(s)
nil = "NIL"
nstring = string / nil
number = 1*DIGIT
  ; Unsigned 32-bit integer
  ; (0 <= n < 4,294,967,296)
number64 = 1*DIGIT
  ; Unsigned 63-bit integer
  ; (0 <= n <= 9,223,372,036,854,775,807)
z-number = digit-nz *DIGIT
password = astring

partial-range = number ["." nz-number]
                  ; Copied from RFC 5092 (IMAP URL)

partial = "<" number "." nz-number ">"
          ; Partial FETCH request. 0-based offset of
          ; the first octet, followed by the number of octets
          ; in the fragment.

quoted          = DQUOTE *QUOTED-CHAR DQUOTE

QUOTED-CHAR     = <any TEXT-CHAR except quoted-specials> /
                  "\" quoted-specials / UTF8-2 / UTF8-3 / UTF8-4

quoted-specials = DQUOTE / "\"

rename          = "RENAME" SP mailbox SP mailbox
                  ; Use of INBOX as a destination gives a NO error

response        = *(continue-req / response-data) response-done

response-data   = "*" SP (resp-cond-state / resp-cond-bye /
                  mailbox-data / message-data / capability-data /
                  enable-data) CRLF

response-done   = response-tagged / response-fatal

response-fatal  = "*" SP resp-cond-bye CRLF
                  ; Server closes connection immediately

response-tagged = tag SP resp-cond-state CRLF

resp-code-apnd  = "APPENDUID" SP nz-number SP append-uid

resp-code-copy  = "COPYUID" SP nz-number SP uid-set SP uid-set

resp-cond-auth  = ("OK" / "PREAUTH") SP resp-text
                  ; Authentication condition

resp-cond-bye   = "BYE" SP resp-text

resp-cond-state = ("OK" / "NO" / "BAD") SP resp-text
                  ; Status condition
resp-specials = "\]

;; Can we make "text" optional? Will this have any bad side effects?
resp-text = ["[
resp-text-code "]" SP text

resp-text-code = "ALERT" /
"BADCHARSET" [SP "(" charset *(SP charset ")")"] /
capability-data / "PARSE" /
"PERMANENTFLAGS" SP "("[flag-perm *(SP flag-perm)]) ")" /
"READ-ONLY" / "READ-WRITE" / "TRYCREATE" /
"UIDNEXT" SP nz-number / "UIDVALIDITY" SP nz-number /
resp-code-apnd / resp-code-copy / "UIDNOTSTICKY" /
"UNAVAILABLE" / "AUTHENTICATIONFAILED" /
"AUTHORIZATIONFAILED" / "EXPIRED" /
"PRIVACYREQUIRED" / "CONTACTADMIN" / "NOPERM" /
"INUSE" / "EXPUNGEISSUED" / "CORRUPTION" /
"SERVERBUG" / "CLIENTBUG" / "CANNOT" /
"LIMIT" / "OVERQUOTA" / "ALREADYEXISTS" /
"NONEXISTENT" /
"CLOSED" /
atom [SP 1*<any TEXT-CHAR except "]">]

search = "SEARCH" [search-return-opts]
SP search-program

search-correlator = SP "(" "TAG" SP tag-string ")"

search-key = "ALL" / "ANSWERED" / "BCC" SP astring /
"BEFORE" SP date / "BODY" SP astring /
"CC" SP astring / "DELETED" / "FLAGGED" /
"FROM" SP astring / "KEYWORD" SP flag-keyword /
"NEW" / "OLD" / "ON" SP date / "SEEN" /
"SINCE" SP date / "SUBJECT" SP astring /
"TEXT" SP astring / "TO" SP astring /
"UNANSWERED" / "UNDELETED" / "UNFLAGGED" /
"UNKEYWORD" SP flag-keyword / "UNSEEN" /

; Above this line were in [IMAP2]
"DRAFT" / "HEADER" SP header-fld-name SP astring /
"LARGER" SP number / "NOT" SP search-key /
"OR" SP search-key SP search-key /
"SENTBEFORE" SP date / "SENTON" SP date /
"SENTSINCE" SP date / "SMALLER" SP number /
"UID" SP sequence-set / "UNDRAFT" / sequence-set /
"(" search-key *(SP search-key ")")"

search-modifier-name = tagged-ext-label
search-mod-params = tagged-ext-val
    ; This non-terminal shows recommended syntax
    ; for future extensions.

search-program = ["CHARSET" SP charset SP]
    search-key *(SP search-key)
    ; CHARSET argument to SEARCH MUST be
    ; registered with IANA.

search-ret-data-ext = search-modifier-name SP search-return-value
    ; Note that not every SEARCH return option
    ; is required to have the corresponding
    ; ESEARCH return data.

search-return-data = "MIN" SP nz-number /
    "MAX" SP nz-number /
    "ALL" SP sequence-set /
    "COUNT" SP number /
    search-ret-data-ext
    ; All return data items conform to
    ; search-ret-data-ext syntax

search-return-opts = SP "RETURN" SP "(" [search-return-opt
    *(SP search-return-opt)] ")"

search-return-opt = "MIN" / "MAX" / "ALL" / "COUNT" /
    search-ret-opt-ext
    ; conforms to generic search-ret-opt-ext
    ; syntax

search-ret-opt-ext = search-modifier-name [SP search-mod-params]

search-return-value = tagged-ext-val
    ; Data for the returned search option.
    ; A single "nz-number"/"number"/"number64" value
    ; can be returned as an atom (i.e., without
    ; quoting). A sequence-set can be returned
    ; as an atom as well.

section = "[" [section-spec] "]"

section-msgtext = "HEADER" / "HEADER.FIELDS" [".NOT"] SP header-list /
    "TEXT"
    ; top-level or MESSAGE/RFC822 or MESSAGE/GLOBAL part

section-part = nz-number "(" nz-number
    ; body part reference.
    ; Allows for accessing nested body parts.
section-spec = section-msgtext / (section-part ["." section-text])

section-text = section-msgtext / "MIME"
               ; text other than actual body part (headers, etc.)

select = "SELECT" SP mailbox

seq-number = nz-number / "*"
               ; message sequence number (COPY, FETCH, STORE
               ; commands) or unique identifier (UID COPY,
               ; UID FETCH, UID STORE commands).
               ; * represents the largest number in use. In
               ; the case of message sequence numbers, it is
               ; the number of messages in a non-empty mailbox.
               ; In the case of unique identifiers, it is the
               ; unique identifier of the last message in the
               ; mailbox or, if the mailbox is empty, the
               ; mailbox’s current UIDNEXT value.
               ; The server should respond with a tagged BAD
               ; response to a command that uses a message
               ; sequence number greater than the number of
               ; messages in the selected mailbox. This
               ; includes "*" if the selected mailbox is empty.

seq-range = seq-number ":" seq-number
               ; two seq-number values and all values between
               ; these two regardless of order.
               ; Example: 2:4 and 4:2 are equivalent and indicate
               ; values 2, 3, and 4.
               ; Example: a unique identifier sequence range of
               ; 3291:* includes the UID of the last message in
               ; the mailbox, even if that value is less than 3291.

sequence-set = (seq-number / seq-range) ["," sequence-set]
               ; set of seq-number values, regardless of order.
               ; Servers MAY coalesce overlaps and/or execute the
               ; sequence in any order.
               ; Example: a message sequence number set of
               ; 2,4:7,9,12:* for a mailbox with 15 messages is
               ; equivalent to 2,4,5,6,7,9,12,13,14,15
               ; Example: a message sequence number set of *:4,5:7
               ; for a mailbox with 10 messages is equivalent to
               ; 10,9,8,7,6,5,4,5,6,7 and MAY be reordered and
               ; overlap coalesced to be 4,5,6,7,8,9,10.

status = "STATUS" SP mailbox SP
               "(" * status-att *(SP status-att) ")"
status-att = "MESSAGES" / "UIDNEXT" / "UIDVALIDITY" / "UNSEEN" / "SIZE"

status-att-val = ("MESSAGES" SP number) / ("UIDNEXT" SP nz-number) / ("UIDVALIDITY" SP nz-number) / ("UNSEEN" SP number) / ("SIZE" SP number64)

; Extensions to the STATUS responses should extend this production.
; Extensions should use the generic syntax defined by tagged-ext.

status-att-list = status-att-val *(SP status-att-val)

store = "STORE" SP sequence-set SP store-att-flags

store-att-flags = (["+" / "-" ] "FLAGS" [.SILENT]) SP (flag-list / (flag *(SP flag)))

string = quoted / literal

subscribe = "SUBSCRIBE" SP mailbox

tag = 1*<any ASTRING-CHAR except "+">

tagged-ext-label = tagged-label-fchar *tagged-label-char

; Is a valid RFC 3501 "atom".

tagged-label-fchar = ALPHA / "-" / "." / ";"

tagged-label-char = tagged-label-fchar / DIGIT / ":" 

tagged-ext-comp = astring / tagged-ext-comp *(SP tagged-ext-comp) / "(" tagged-ext-comp ")"

; Extensions that follow this general syntax should use nstring instead of astring when appropriate in the context of the extension.
; Note that a message set or a "number" can always be represented as an "atom".
; An URL should be represented as a "quoted" string.

tagged-ext-simple = sequence-set / number / number64

tagged-ext-val = tagged-ext-simple /
"(" [tagged-ext-comp] ")"

text = 1*TEXT-CHAR

TEXT-CHAR = <any CHAR except CR and LF>

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

uid = "UID" SP
     (copy / move / fetch / search / store / uid-expunge)
     ; Unique identifiers used instead of message
     ; sequence numbers

uid-expunge = "EXPUNGE" SP sequence-set
             ; Unique identifiers used instead of message
             ; sequence numbers

uid-set = (uniqueid / uid-range) *("," uid-set)

uid-range = (uniqueid ":" uniqueid)
            ; two uniqueid values and all values
            ; between these two regards of order.
            ; Example: 2:4 and 4:2 are equivalent.

uniqueid = nz-number
           ; Strictly ascending

unsubscribe = "UNSUBSCRIBE" SP mailbox

userid = astring

UTF8-2 = <Defined in Section 4 of RFC 3629>

UTF8-3 = <Defined in Section 4 of RFC 3629>

UTF8-4 = <Defined in Section 4 of RFC 3629>

x-command = "X" atom <experimental command arguments>

zone = ("+" / "-") 4DIGIT
       ; Signed four-digit value of hhmm representing
       ; hours and minutes east 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 2060, RFC 1730, unpublished IMAP2bis.TXT document, RFC 1176, and RFC 1064.

11. Security Considerations

IMAP4rev2 protocol transactions, including electronic mail data, are sent in the clear over the network unless protection from snooping is negotiated. This can be accomplished either by the use of IMAPS service, STARTTLS command, negotiated privacy protection in the AUTHENTICATE command, or some other protection mechanism.

11.1. STARTTLS Security Considerations

IMAP client and server implementations MUST comply with relevant TLS recommendations from [RFC8314]. Additionally, when using TLS 1.2, IMAP implementations MUST implement TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 cipher suite, and SHOULD implement the TLS_RSA_WITH_AES_128_CBC_SHA [TLS] cipher suite. This is important as it assures that any two compliant implementations can be configured to interoperate. Other TLS cipher suites recommended in RFC 7525 are RECOMMENDED: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256, TLS_DHE_RSA_WITH_AES_256_GCM_SHA384 and TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384. All other cipher suites are OPTIONAL. Note that this is a change from section 2.1 of [IMAP-TLS].

During the [TLS] negotiation, the client MUST check its understanding of the server hostname against the server’s identity as presented in the server Certificate message, in order to prevent man-in-the-middle attacks. This procedure is described in [RFC7817].

Both the client and server MUST check the result of the STARTTLS command and subsequent [TLS] negotiation to see whether acceptable authentication and/or privacy was achieved.

11.2. COPYUID and APPENDUID response codes

The COPYUID and APPENDUID response codes return information about the mailbox, which may be considered sensitive if the mailbox has permissions set that permit the client to COPY or APPEND to the mailbox, but not SELECT or EXAMINE it.

Consequently, these response codes SHOULD NOT be issued if the client does not have access to SELECT or EXAMINE the mailbox.
11.3. Other Security Considerations

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 with a [SASL] mechanism that does not use plaintext passwords, by first negotiating encryption via STARTTLS or some other protection mechanism.

A server implementation MUST implement a configuration that, at the time of authentication, requires:

(1) The STARTTLS command has been negotiated.

OR

(2) Some other mechanism that protects the session from password snooping has been provided.

OR

(3) The following measures are in place:

(a) The LOGINDISABLED capability is advertised, and [SASL] mechanisms (such as PLAIN) using plaintext passwords are NOT advertised in the CAPABILITY list.

AND

(b) The LOGIN command returns an error even if the password is correct.

AND

(c) The AUTHENTICATE command returns an error with all [SASL] mechanisms that use plaintext passwords, even if the password is correct.

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

A server SHOULD have mechanisms in place to limit or delay failed AUTHENTICATE/LOGIN attempts.

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

12. IANA Considerations

IANA is requested to update "Service Names and Transport Protocol Port Numbers" registry as follows:

1. Registration for TCP "imap" port 143 should be updated to point to this document and RFC 3501.
2. Registration for TCP "imaps" port 993 should be updated to point to this document, RFC 8314 and RFC 3501.

3. Both UDP port 143 and UDP port 993 should be marked as "Reserved" in the registry.

Additional IANA actions are specified in subsection of this section.

12.1. Updates to IMAP4 Capabilities registry

IMAP4 capabilities are registered by publishing a standards track or IESG approved informational or experimental RFC. The registry is currently located at: http://www.iana.org/assignments/imap4-capabilities

As this specification revises the STARTTLS and LOGINDISABLED extensions previously defined in [IMAP-TLS], IANA is requested to update registry entries for these 2 extensions to point to this document.

12.2. GSSAPI/SASL service name

GSSAPI/Kerberos/SASL service names are registered by publishing a standards track or IESG approved experimental RFC. The registry is currently located at: http://www.iana.org/assignments/gssapi-service-names

IANA is requested to update the "imap" service name previously registered in RFC 3501, to point to this document.

13. References

13.1. Normative References


[DIGEST-MD5]

[DISPOSITION]

[PLAIN]

[KEYWORDS]

[LANGUAGE-TAGS]

[LOCATION]

[MD5]

[MIME-HDRS]

[MIME-IMB]


13.2. Informative References (related protocols)

[IMAP-DISC]

[IMAP-I18N]

[IMAP-MODEL]

[IMAP-UTF-8]
13.3. Informative References (historical aspects of IMAP and related protocols)


Appendix A. Backward compatibility with IMAP4rev1

An implementation that wants to remain compatible with IMAP4rev1 can advertise both IMAP4rev1 and IMAP4rev2 in its CAPABILITY response/response code. While some IMAP4rev1 responses were removed in IMAP4rev2, their presence will not break IMAP4rev2-only clients.

If both IMAP4rev1 and IMAP4rev2 are advertised, an IMAP client that wants to use IMAP4rev2 MUST issue an "ENABLE IMAP4rev2" command.

Servers advertising both IMAP4rev1 and IMAP4rev2 SHOULD NOT generate UTF-8 quoted strings unless the client has issued "ENABLE IMAP4rev2". Consider implementation of mechanisms described or referenced in [IMAP-UTF-8] to achieve this goal.

Servers advertising both IMAP4rev1 and IMAP4rev2, and clients intending to be compatible with IMAP4rev1 servers MUST be compatible with the international mailbox naming convention described in the following subsection.

A.1. Mailbox International Naming Convention

By convention, international mailbox names in IMAP4rev2 are specified using a modified version of the UTF-7 encoding described in [UTF-7]. Modified UTF-7 may also be usable in servers that implement an earlier version of this protocol.

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 and 0x7f-0xff) 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. Only characters inside the modified BASE64 alphabet are permitted in modified BASE64 text.

"&" is used to shift to modified BASE64 and "-" to shift back to US-ASCII. There is no implicit shift from BASE64 to US-ASCII, and null shifts ("-&" while in BASE64; note that "&-" while in US-ASCII means "&") are not permitted. However, all names start in US-ASCII, and
MUST end in US-ASCII; that is, a name that ends with a non-ASCII ISO-10646 character MUST end with a "-".

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 characters can be represented in encoded form.

Although modified UTF-7 is a convention, it establishes certain requirements on server handling of any mailbox name with an embedded "&" character. In particular, server implementations MUST preserve the exact form of the modified BASE64 portion of a modified UTF-7 name and treat that text as case-sensitive, even if names are otherwise case-insensitive or case-folded.

Server implementations SHOULD verify that any mailbox name with an embedded "&" character, used as an argument to CREATE, is: in the correctly modified UTF-7 syntax, has no superfluous shifts, and has no encoding in modified BASE64 of any printing US-ASCII character which can represent itself. However, client implementations MUST NOT depend upon the server doing this, and SHOULD NOT attempt to create a mailbox name with an embedded "&" character unless it complies with the modified UTF-7 syntax.

Server implementations which export a mail store that does not follow the modified UTF-7 convention MUST convert to modified UTF-7 any mailbox name that contains either non-ASCII characters or the "&" character.

For example, here is a mailbox name which mixes English, Chinese, and Japanese text: ~peter/mail/&U,BTFw-/&ZeVnLIqe-
For example, the string "&Jjo!" is not a valid mailbox name because it does not contain a shift to US-ASCII before the ". The correct form is ";jjo-!". The string ";U,BTFw-&ZeVnlIqe-" is not permitted because it contains a superfluous shift. The correct form is ";U,BTF2XlZyyKng-".

Appendix B. Changes from RFC 3501 / IMAP4rev1

The following is the plan for remaining changes. The plan might change over time.

1. Fold in the following extensions/RFC: RFC 5530 (IMAP Response Codes, done), UIDPLUS (done), ENABLE (done), ESEARCH (done), SPECIAL-USE (list of new mailbox attributes is done), LITERAL- (done), NAMESPACE (done), SASL-IR (done), IDLE (done), MOVE (done).

2. Add CLOSED response code (from CONDSTORE) - done

3. Add support for $MDNSent and $Forwarded IMAP keywords - done. Add more examples showing their use? Also add other keywords like $Phishing, $Junk, $NonJunk?

4. Require all unsolicited FETCH updates to include UID - done.

5. Update recommendations on TLS ciphers to match UTA WG work (as per RFC 8314, RFC 7525 and RFC 7817) - done.

6. Possibly fold in the following extensions/RFC: Base LIST-EXTENDED syntax plus deprecate LSUB (replace it with LIST \Subscribed) minus the requirement to support multiple list patterns, STATUS-in-LIST, SEARCHRES, BINARY (only the FETCH changes on leaf body part and make APPEND related ones optional. See the mailing list discussion), Unique mailstore IDs for messages (OBJECTID extension, RFC 8474) -- rough consensus to keep it as an extension.

7. Add STATUS SIZE (total mailbox size) - done Add STATUS DELETED (number of messages with \Deleted flag set)? Or DELETEDSIZE?

8. Deprecate features: What should we do with NEW search key (which implies RECENT): deprecate it or just redefine it to ignore RECENT state?

9. Drop UTF-7, all mailboxes are always in UTF-8 - done.

10. Revise IANA registration of IMAP extensions and give advice on use of "X-" convention.
11. Allow word-based searching (as per Chris Newman)?

The following changes since RFC 3501 were done so far:

1. Folded in IMAP UNSELECT (RFC 3691), UIDPLUS (RFC 4315), ESEARCH (RFC 4731), ENABLE (RFC 5161), IDLE (RFC 2177), SASL-IR (RFC 4959) and MOVE (RFC 6851) extensions. Also folded RFC 5530.

2. SEARCH command now requires to return ESEARCH response (SEARCH response is now deprecated).

3. Added CLOSED response code from RFC 7162.

4. Updated to use modern TLS-related recommendations as per RFC 8314, RFC 7817, RFC 7525.

5. For future extensibility extended ABNF for tagged-ext-simple to allow for bare number64.

6. Added SHOULD level requirement on IMAP servers to support $MDNSent and $Forwarded keywords.

7. Added STATUS SIZE.

8. Mailbox names and message headers now allow for UTF-8. Support for Modified UTF-7 in mailbox names is not required, unless compatibility with IMAP4rev1 is desired.

9. UNSEEN response code on SELECT/EXAMINE is now deprecated.

10. RECENT response on SELECT/EXAMINE, \Recent flag, RECENT STATUS item are now deprecated.

Appendix C. Acknowledgement

Earlier versions of this document were edited by Mark Crispin. Sadly, he is no longer available to help with this work. Editors of this revisions are hoping that Mark would have approved.

Chris Newman has contributed text on I18N and use of UTF-8 in messages and mailbox names.

Thank you to Tony Hansen for helping with the index generation.

This document incorporate text from RFC 4315, RFC 4466, RFC 4731, RFC 5161, RFC 6154 so work done by authors/editors of these documents is appreciated.
Index

$  $Forwarded (predefined flag)  12
   $MDNSent (predefined flag)  12

+  +FLAGS <flag list>  67
   +FLAGS.SILENT <flag list>  67

-  -FLAGS <flag list>  67
   -FLAGS.SILENT <flag list>  67

A  ALERT (response code)  74
   ALL (fetch item)  63
   ALL (search key)  60
   ALL (search result option)  59
   ALREADYEXISTS (response code)  74
   ANSWERED (search key)  60
   APPEND (command)  51
   APPENDUID (response code)  74
   AUTHENTICATE (command)  27
   AUTHENTICATIONFAILED (response code)  75
   AUTHORIZATIONFAILED (response code)  75

B  BAD (response)  82
   BADCHARSET (response code)  76
   BCC <string> (search key)  60
   BEFORE <date> (search key)  60
   BODY (fetch item)  64
   BODY (fetch result)  91
   BODY <string> (search key)  60
   BODY[<section>]<partial> (fetch item)  66
   BODYSTRUCTURE (fetch item)  66
   BODYSTRUCTURE (fetch result)  91
   BODY[<section>]<origin octet> (fetch result)  91
   BODY[<section>]<partial> (fetch item)  64
   BYE (response)  83
   Body Structure (message attribute)  13

C  CANNOT (response code)  76
   CAPABILITY (command)  24
   CAPABILITY (response code)  76
   CAPABILITY (response)  84
CC <string> (search key) 60
CHECK (command) 56
CLIENTBUG (response code) 76
CLOSE (command) 56
CLOSED (response code) 76
CONTACTADMIN (response code) 77
COPY (command) 68
COPYUID (response code) 77
CORRUPTION (response code) 77
COUNT (search result option) 59
CREATE (command) 35

DELETE (command) 36
DELETED (search key) 60
DRAFT (search key) 60

ENABLE (command) 31
ENVELOPE (fetch item) 66
ENVELOPE (fetch result) 94
ESEARCH (response) 88
EXAMINE (command) 35
EXPIRED (response code) 78
EXPUNGE (command) 57
EXPUNGE (response) 90
EXPUNGEISSUED (response code) 78
Envelope Structure (message attribute) 13

FAST (fetch item) 63
FETCH (command) 63
FETCH (response) 91
FLAGGED (search key) 60
FLAGS (fetch item) 66
FLAGS (fetch result) 95
FLAGS (response) 89
FLAGS <flag list> (store command data item) 67
FLAGS.SILENT <flag list> (store command data item) 67
FROM <string> (search key) 60
FULL (fetch item) 64
Flags (message attribute) 11

HEADER (part specifier) 64
HEADER <field-name> <string> (search key) 60
HEADER.FIELDS (part specifier) 64
HEADER.FIELDS.NOT (part specifier) 64
IDLE (command) 53
INTERNALDATE (fetch item) 66
INTERNALDATE (fetch result) 95
INUSE (response code) 78
Internal Date (message attribute) 12

KEYWORD <flag> (search key) 61
Keyword (type of flag) 12

LARGER <n> (search key) 61
LIMIT (response code) 78
LIST (command) 41
LIST (response) 85
LOGOUT (command) 25
LSUB (command) 44
LSUB (response) 87

MAX (search result option) 58
MAY (specification requirement term) 5
MESSAGES (status item) 50
MIME (part specifier) 65
MIN (search result option) 58
MOVE (command) 69
MUST (specification requirement term) 5
MUST NOT (specification requirement term) 5
Message Sequence Number (message attribute) 11

NAMESPACE (command) 45
NAMESPACE (response) 88
NEW (search key) 61
NO (response) 81
NONEXISTENT (response code) 78
NOOP (command) 25
NOPERM (response code) 79
NOT <search-key> (search key) 61

OK (response) 81
ON <date> (search key) 61
OPTIONAL (specification requirement term) 5
OR <search-key1> <search-key2> (search key) 61
OVERQUOTA (response code) 79
PARSE (response code) 79
PERMANENTFLAGS (response code) 79
PREAUTH (response) 82
PRIVACYREQUIRED (response code) 79
Permanent Flag (class of flag) 12
Predefined keywords 12

READ-ONLY (response code) 80
READ-WRITE (response code) 80
RECOMMENDED (specification requirement term) 5
RENAME (command) 38
REQUIRED (specification requirement term) 5
RFC822 (fetch item) 66
RFC822 (fetch result) 95
RFC822.HEADER (fetch item) 66
RFC822.HEADER (fetch result) 95
RFC822.SIZE (fetch item) 66
RFC822.SIZE (fetch result) 95
RFC822.TEXT (fetch item) 66
RFC822.TEXT (fetch result) 95

SEARCH (command) 58
SEEN (search key) 61
SELECT (command) 33
SENTBEFORE <date> (search key) 61
SENTON <date> (search key) 61
SENTSINCE <date> (search key) 61
SERVERBUG (response code) 80
SHOULD (specification requirement term) 5
SHOULD NOT (specification requirement term) 5
SINCE <date> (search key) 61
SIZE (status item) 51
SMALLER <n> (search key) 61
STARTTLS (command) 26
STATUS (command) 49
STATUS (response) 88
STORE (command) 66
SUBJECT <string> (search key) 61
SUBSCRIBE (command) 40
Session Flag (class of flag) 12
System Flag (type of flag) 11

TEXT (part specifier) 64
TEXT <string> (search key) 61
TO <string> (search key)  61  
TRYCREATE (response code)  80  

U

UID (command)  70  
UID (fetch item)  66  
UID (fetch result)  95  
UID <sequence set> (search key)  62  
UIDNEXT (response code)  80  
UIDNEXT (status item)  50  
UIDNOTSTICKY (response code)  80  
UIDVALIDITY (response code)  81  
UIDVALIDITY (status item)  50  
UNANSWERED (search key)  62  
UNAVAILABLE (response code)  81  
UNDELETED (search key)  62  
UNDRAFT (search key)  62  
UNFLAGGED (search key)  62  
UNKEYWORD <flag> (search key)  62  
UNSEEN (search key)  62  
UNSEEN (status item)  50  
UNSELECT (command)  57  
UNSUBSCRIBE (command)  41  
Unique Identifier (UID) (message attribute)  9  

X

X<atom> (command)  72  

[  
[RFC-5322] Size (message attribute)  13  

\  
\All (mailbox name attribute)  86  
\Answered (system flag)  11  
\Archive (mailbox name attribute)  86  
\Deleted (system flag)  11  
\Draft (system flag)  12  
\Drafts (mailbox name attribute)  86  
\Flagged (mailbox name attribute)  86  
\Flagged (system flag)  11  
\HasChildren (mailbox name attribute)  85  
\HasNoChildren (mailbox name attribute)  85  
\Junk (mailbox name attribute)  86  
\Marked (mailbox name attribute)  85  
\Noinferiors (mailbox name attribute)  85  
\Noselect (mailbox name attribute)  85  
\Recent (system flag)  12  
\Seen (system flag)  11  

Melnikov & Leiba  Expires August 7, 2019 [Page 127]
\Sent (mailbox name attribute) 86
\Trash (mailbox name attribute) 87
\Unmarked (mailbox name attribute) 85

Authors’ Addresses

Alexey Melnikov (editor)
Isode Ltd
14 Castle Mews
Hampton, Middlesex TW12 2NP
UK

Email: Alexey.Melnikov@isode.com

Barry Leiba (editor)
Huawei Technologies

Phone: +1 646 827 0648
Email: barryleiba@computer.org
URI: http://internetmessagingtechnology.org/