Status:	Proposed Standard
Obsoletes:	3501
More info:	Errata exist | Datatracker | IPR | Info page

Stream:	Internet Engineering Task Force (IETF)
RFC:	9051
Obsoletes:	3501
Category:	Standards Track
Published:	August 2021
ISSN:	2070-1721
Authors:	A. Melnikov, Ed.	B. Leiba, Ed.
---	---	---
	Isode Ltd	*Futurewei Technologies*

RFC 9051
Internet Message Access Protocol (IMAP) - Version 4rev2

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; removing messages permanently; setting and clearing flags; parsing per RFCs 5322, 2045, and 2231; 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 the one specified in RFC 6409.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc9051.

Copyright Notice

Table of Contents

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, 4, and 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 "Formal Syntax" (Section 9).

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. Note that each line includes the terminating CRLF.

The key words "**MUST**", "**MUST NOT**", "**REQUIRED**", "**SHALL**", "**SHALL NOT**", "**SHOULD**", "**SHOULD NOT**", "**RECOMMENDED**", "**NOT RECOMMENDED**", "**MAY**", and "**OPTIONAL**" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

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, UNSELECT command, or connection termination).

The term "Implicit TLS" refers to the automatic negotiation of TLS whenever a TCP connection is made on a particular TCP port that is used exclusively by that server for TLS connections. The term "Implicit TLS" is intended to contrast with the use of the STARTTLS command in IMAP that is used by the client and the server to explicitly negotiate TLS on an established cleartext TCP connection.

Characters are 8-bit UTF-8 (of which 7-bit US-ASCII is a subset), 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 a character set; refer to these documents for more detail.

There are several protocol conventions in IMAP. These refer to aspects of the specification that 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 IMAP4rev1 [RFC3501], IMAP2 [IMAP2], and unpublished IMAP2bis [IMAP2BIS] protocols. IMAP4rev2 is largely compatible with the IMAP4rev1 protocol described in RFC 3501 and the IMAP4 protocol described in [RFC1730]; the exception being in certain facilities added in [RFC1730] and [RFC3501] that proved problematic and were subsequently removed or replaced by better alternatives. In the course of the evolution of IMAP4rev2, some aspects in the earlier protocols have become obsolete. Obsolete commands, responses, and data formats that an IMAP4rev2 implementation can encounter when used with an earlier implementation are described in Appendices A and E and [IMAP-OBSOLETE]. IMAP4rev2 supports 63-bit body parts and message sizes. IMAP4rev2 compatibility with BINARY and LIST-EXTENDED IMAP extensions are described in Appendices B and C, respectively.

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 [RFC822] standard, and as a consequence, the "RFC822.SIZE" fetch item in IMAP incorporates "RFC822" in its name. "RFC822" should be interpreted as a reference to the updated [RFC5322] standard.

IMAP4rev2 does not specify a means of posting mail; this function is handled by a mail submission protocol such as the one specified in [RFC6409].

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 (cleartext port) or port 993 (Implicit TLS port).

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 reading either a line or 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. More formally: the client SHOULD generate a unique tag for every command, but a server MUST accept tag reuse.

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 Section 4.3); in the other case, the command arguments require server feedback (see the AUTHENTICATE command in Section 6.2.2). 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 sending 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 that 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 strictly enforce the syntax outlined in this specification. 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 remembered (cached), so that the client can reference its remembered copy rather than sending a command to the server to request the data. In the case of certain server data, the data MUST be remembered, as specified elsewhere in this document.

This topic is discussed in greater detail in "Server Responses" (see Section 7).

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 (UID) or the message sequence number.

2.3.1.1. Unique Identifier (UID) Message Attribute

A UID is an unsigned non-zero 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 those of all message(s) that are already in the mailbox. 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 [IMAP-MODEL]); this is discussed further in [IMAP-DISC].

Associated with every mailbox are two 32-bit unsigned non-zero values that 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. A good UIDVALIDITY value to use is a 32-bit representation of the current date/time when the value is assigned: this ensures that the value is unique and always increases. Another possible alternative is a global counter that gets incremented every time a mailbox is created.

> 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 reordered by a non-IMAP agent, the unique identifiers in the mailbox MUST be regenerated, since the former unique identifiers are no longer strictly ascending as a result of the reordering.
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. Note that this situation can be very disruptive to client message caching.
3. If the mailbox is deleted/renamed 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 assign a new UIDVALIDITY value to the new instance of the mailbox.
4. The combination of mailbox name, UIDVALIDITY, and UID must refer to a single, immutable (or expunged) message on that server forever. In particular, the internal date, RFC822.SIZE, envelope, body structure, and message texts (all BODY[...] fetch data items) MUST never change. This does not include message numbers, nor does it include attributes that can be set by a STORE command (such as FLAGS). When a message is expunged, its UID MUST NOT be reused under the same UIDVALIDITY value.

2.3.1.2. Message Sequence Number Message Attribute

A message sequence number is a relative position from 1 to the number of messages in the mailbox. This position MUST be ordered by ascending unique identifiers. 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. As another example, if message 287 in a 523-message mailbox has UID 12345, there are exactly 286 messages that have lesser UIDs and 236 messages that have greater UIDs.

2.3.2. Flags Message Attribute

A message has a list of zero or more named tokens, known as "flags", associated with it. A flag is set by its addition to this list and is cleared by its removal. There are two types of flags in IMAP4rev2: system flags and keywords. A flag of either type can be permanent or session-only.

A system flag is a flag name that is predefined in this specification and begins with "\". Certain system flags (\Deleted and \Seen) 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 use 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 [RFC3501] but were found to be useful by client implementations. These keywords SHOULD be supported (allowed in SEARCH and allowed and preserved in APPEND, COPY, and MOVE commands) by server implementations:

$Forwarded
: Message has been forwarded to another email address by being 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 [RFC8098] was generated and sent for this message. See [RFC3503] for more details on how this keyword is used and for requirements on clients and servers.

$Junk
: The user (or a delivery agent on behalf of the user) may choose to mark a message as definitely containing junk ($Junk; see also the related keyword $NotJunk). The $Junk keyword can be used to mark, group, or hide undesirable messages (and such messages might be moved or deleted later). See [IMAP-KEYWORDS-REG] for more information.

$NotJunk
: The user (or a delivery agent on behalf of the user) may choose to mark a message as definitely not containing junk ($NotJunk; see also the related keyword $Junk). The $NotJunk keyword can be used to mark, group, or show messages that the user wants to see. See [IMAP-KEYWORDS-REG] for more information.

$Phishing
: The $Phishing keyword can be used by a delivery agent to mark a message as highly likely to be a phishing email. A message that's determined to be a phishing email by the delivery agent should also be considered a junk email and have the appropriate junk filtering applied, including setting the $Junk flag and placing

the message in the \Junk special-use mailbox (see Section 7.3.1), if available.

If both the $Phishing flag and the $Junk flag are set, the user agent should display an additional warning message to the user. Additionally, the user agent might display a warning, such as something of the form, "This message may be trying to steal your personal information," when the user clicks on any hyperlinks within the message.

The requirement for both $Phishing and $Junk to be set before a user agent displays a warning is for better backwards compatibility with existing clients that understand the $Junk flag but not the $Phishing flag. This is so that when an unextended client removes the $Junk flag, an extended client will also show the correct state. See [IMAP-KEYWORDS-REG] for more information.

$Junk and $NotJunk are mutually exclusive. If more than one of these is set for a message, the client **MUST** treat it as if none are set, and it **SHOULD** unset both of them on the IMAP server.

Other registered keywords can be found in the "IMAP and JMAP Keywords" registry [IMAP-KEYWORDS-REG]. New keywords **SHOULD** be registered in this registry using the procedure specified in [RFC5788].

A flag can be permanent or session-only on a per-flag basis. Permanent flags are those that 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

An Internal Date message attribute is the internal date and time of the message on the server. This is not the date and time in the [RFC5322] header but rather a date and time that reflects when the message was received. In the case of messages delivered via [SMTP], this is the date and time of final delivery of the message as defined by [SMTP]. In the case of messages created by the IMAP4rev2 COPY or MOVE command, this **SHOULD** be the same as the Internal Date attribute of the source message. In the case of messages created 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. RFC822.SIZE Message Attribute

RFC822.SIZE is the number of octets in the message when the message is expressed in [RFC5322] format. This size SHOULD match the result of a "FETCH BODY[]" command. If the message is internally stored in some other format, the server calculates the size and often stores it for later use to avoid the need for recalculation.

2.3.5. Envelope Structure Message Attribute

An envelope structure is a parsed representation of the [RFC5322] 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 body structure is 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 [RFC5322] text of a message, IMAP4rev2 permits the fetching of portions of the full message text. Specifically, it is possible to fetch the [RFC5322] message header, the [RFC5322] 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 or UNSELECT command.

3.3. Selected State

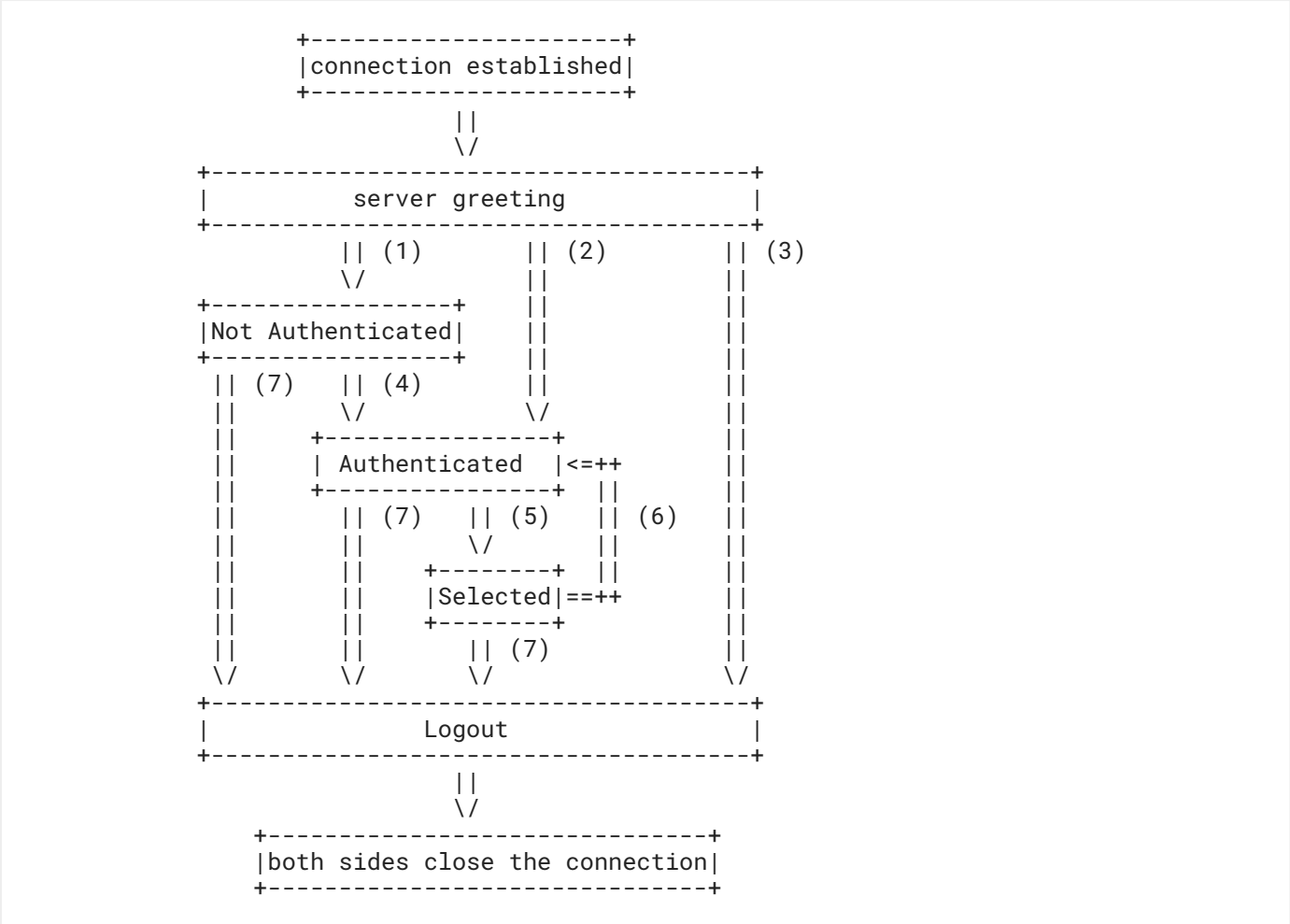
In a selected state, a mailbox has been selected to access. This state is entered when a mailbox has been successfully selected.

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 the 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 SHOULD NOT unilaterally close the connection without first sending an untagged BYE response that contains the reason for doing so. A client SHOULD NOT unilaterally close the connection; instead, it 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.



Legend for the above diagram:

(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 or UNSELECT 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. A sequence set can contain ranges of sequence numbers (such as "5:50"), an enumeration of specific sequence numbers, or a combination of the above. A sequence set can use the special symbol "*" to represent the maximum sequence number in the mailbox. A sequence set never contains unique identifiers.

A "UID set" is similar to the sequence set, but uses unique identifiers instead of message sequence numbers, and is not permitted to contain the special symbol "*".

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 a string, without limitation on the characters the string may include. The non-synchronizing literal form is also the general form, but it has a length restriction. The quoted string form is an alternative that avoids the overhead of processing a literal, but has limitations on the characters that may be used.

When the distinction between synchronizing and non-synchronizing literals is not important, this document only 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, a close brace ("}"), and a 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 alternative form of synchronizing literal and may be used from client to server anywhere a synchronizing literal is permitted. 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. Unless otherwise specified in an IMAP extension, 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 a CRLF (a synchronizing literal with an octet count of 0), or as {0+} followed by a 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 content transfer encoding is defined, unencoded binary strings are not permitted, unless returned in a <literal8> in response to a BINARY.PEEK[<section-binary>]<<partial>> or BINARY[<section-binary>]<<partial>> FETCH data item. A "binary string" is any string with NUL characters. A string with an excessive amount of CTL characters MAY also be considered to be binary. Unless returned in response to BINARY.PEEK[...]/BINARY[...] FETCH, client and server implementations MUST encode binary data into a textual form, such as base64, before transmitting the data.

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 that 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.

Examples:

The following LIST response:

```
* LIST () "/" NIL
```

is equivalent to:

```
* LIST () "/" "NIL"
```

as LIST response ABNF is using "astring" for mailbox name.

However, the following response:

```
* FETCH 1 (BODY[1] NIL)
```

is not equivalent to:

```
* FETCH 1 (BODY[1] "NIL")
```

The former indicates absence of the body part, while the latter means that it contains a string with the three characters "NIL".

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 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 denormalized UTF-8 mailbox name and convert it to Unicode Normalization Form C (NFC) (as per Net-Unicode requirements) prior to mailbox creation. Servers that choose to accept such denormalized UTF-8 mailbox names MUST accept them in all IMAP commands that have a mailbox name parameter. In particular, SELECT <name> must open the same mailbox that was successfully created with CREATE <name>, even if <name> is a denormalized UTF-8 mailbox name.

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 might not exist on some servers for some users, for example, if the user has no access to personal namespace.) 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 the case of a newly created name but otherwise are case insensitive; and yet others coerce names to a particular case. Client implementations must be able to interact with any of these.

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

1. Any character that is one of the atom-specials (see "Formal Syntax" in Section 9) 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 command 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. See Section 5.1.2.1 and Appendix A.1, respectively.

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 ASCII 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 and 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 per user 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 administrative support staff access rights to their mailbox. In the typical case, there SHOULD be only one Other Users' Namespace per user 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 that 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 that 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 and require the "#" character to be represented as %23 when within URLs. As such, server implementors MAY instead consider using namespace prefixes that do not contain the "#" character.

5.1.2.2. Common Namespace Models

The previous version of this protocol did 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 by this specification and/or extensions. 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 (Section 7.5.1) 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 remember 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 resets the autologout timer.

Note that this specification doesn't have any restrictions on an autologout timer used before successful client authentication. In particular, servers are allowed to use a shortened pre-authentication timer to protect themselves from Denial-of-Service attacks.

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. 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. One example is a FETCH that would cause \Seen flags to be set and a SEARCH UNSEEN command.

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 that 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 responses returned by the UID SEARCH.

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

> FETCH + NOOP + STORE
>
> STORE + COPY + FETCH
>
> COPY + COPY

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

> FETCH + STORE + SEARCH + NOOP
>
> 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.

Use of a SEARCH result variable (see Section 6.4.4.1) creates direct dependency between two commands. See Section 6.4.4.2 for more considerations about pipelining such dependent commands.

6. Client Commands

IMAP4rev2 commands are described in this section. Commands are organized by the state in which the command is permitted. Commands that are permitted in multiple states are listed in the minimum permitted state (for example, commands valid in authenticated and selected states 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 "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 "Responses" (Section 7) for information on these responses and in "Formal Syntax" (Section 9) 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 that 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, with the exception of 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 - arguments invalid

The CAPABILITY command requests a listing of capabilities (e.g., extensions and/or modifications of server behavior) 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 that begins with "AUTH=" indicates that the server supports that particular authentication mechanism as defined in the Simple Authentication and Security Layer (SASL) [SASL]. All such names are, by definition, part of this specification.

Other capability names refer to extensions, revisions, or amendments to this specification. See the documentation of the CAPABILITY response in Section 7.2.2 for additional information. If IMAP4rev1 capability is not advertised, no capabilities, beyond the base IMAP4rev2 set defined in this specification, are enabled without explicit client action to invoke the capability. If both IMAP4rev1 and IMAP4rev2 capabilities are advertised, no capabilities, beyond the base IMAP4rev1 set specified in [RFC3501], are enabled without explicit client action to invoke the capability.

Client and server implementations **MUST** implement the STARTTLS (Section 6.2.1) and LOGINDISABLED capabilities on cleartext ports. Client and server implementations **MUST** also implement AUTH=PLAIN (described in [PLAIN]) capability on both cleartext and Implicit TLS ports. See the Security Considerations (Section 11) for important information.

Unless otherwise specified, all registered extensions to IMAP4rev1 are also valid extensions to IMAP4rev2.

Example:

```
C: abcd CAPABILITY
S: * CAPABILITY IMAP4rev2 STARTTLS AUTH=GSSAPI
 LOGINDISABLED
S: abcd OK CAPABILITY completed
C: efgh STARTTLS
S: efgh 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 (the IDLE command; see Section 6.3.13) should be used instead of NOOP if real-time updates to mailbox state are desirable). 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 (UID 1305 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 conventional user name and plaintext password pair and has no means of establishing privacy protection or integrity checking.

The STARTTLS command is an alternative 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 11) 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
NO - TLS negotiation can't be initiated, due to server configuration error
BAD - STARTTLS received after a successful TLS negotiation or arguments invalid

Note that the STARTTLS command is available only on cleartext ports. The server MUST always respond with a tagged BAD response when the STARTTLS command is received on an Implicit TLS port.

A TLS [TLS-1.3] 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. Some past server implementations incorrectly implemented STARTTLS processing and are known to contain STARTTLS plaintext command injection vulnerability [CERT-555316]. In order to avoid this vulnerability, server implementations MUST do one of the following if any data is received in the same TCP buffer after the CRLF that starts the STARTTLS command:

1. Extra data from the TCP buffer is interpreted as the beginning of the TLS handshake. (If the data is in cleartext, this will result in the TLS handshake failing.)
2. Extra data from the TCP buffer is thrown away.

Note that the first option is friendlier to clients that pipeline the beginning of the STARTTLS command with TLS handshake data.

After successful TLS negotiation, 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 reissue the CAPABILITY command. This is necessary to protect against active attacks that 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 AUTHENTICATE PLAIN dGVzdAB0ZXN0AHRlc3Q=
S: a004 OK Success (tls protection)
```

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 canceled

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 4 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 it 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, the 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 it 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 the STARTTLS command has been negotiated, TLS has been negotiated on an Implicit TLS port, or some other mechanism that protects the session from password snooping has been provided. Server sites SHOULD NOT use any configuration that 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 as the GSSAPI mechanism described in [RFC4752], the SCRAM-SHA-256/SCRAM-SHA-256-PLUS [SCRAM-SHA-256] mechanisms, and/or the EXTERNAL [SASL] mechanism for mutual TLS authentication. (Note that the SASL framework allows for the creation of SASL mechanisms that support 2-factor authentication (2FA); however, none are fully ready to be recommended by this document.)

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 [CAPABILITY IMAP4rev2 STARTTLS AUTH=GSSAPI]
 Capabilities
C: A001 AUTHENTICATE GSSAPI
S: +
C: YIIB+wYJKoZIhvcSAQICAQBuggHqMIIB5qADAgEFoQMCAQ6iBw
   MFACAAAACjggEmYYIBIjCCAR6gAwIBBaESGxB1Lndhc2hpbmd0
   b24uZWR1oi0wK6ADAgEDoSQwIhsEaW1hcBsac2hpdmFtcy5jYW
   Mud2FzaGluZ3Rvbi5lZHWjgdMwgdCgAwIBAaEDAgEDooHDBIHA
   cS1GSa5b+fXnPZNmXB9SjL8Ollj2SKyb+3S0iXMljen/jNkpJX
   AleKTz6BQPzj8duz8EtoOuNfKgweViyn/9B9bccy1uuAE2HI0y
   C/PHXNNU9ZrBziJ8Lm0tTNc98kUpjXnHZhsMcz5Mx2GR6dGknb
   I0iaGcRerMUsWOuBmKKKRmVMMdR9T3EZdpqsBd7jZCNMWotjhi
   vd5zovQlFqQ2Wjc2+y46vKP/iXxWIuQJuDiisyXF0Y8+5GTpAL
   pHDc1/pIGmMIGjoAMCAQGigZsEgZg2on5mSuxoDHEA1w9bcW9n
   FdFxDKpdrQhVGVRDIzcCMCTzvUboqb5KjY1NJKJsfjRQiBYBdE
   NKfzK+g5DlV8nrw81uOcP8NOQCLR5XkoMHC0Dr/80ziQzbNqhx
   O6652Npft0LQwJvenwDI13YxpwOdMXzkWZN/XrEqOWp6GCgXTB
   vCyLWLlWnbaUkZdEYbKHBPjd8t/1x5Yg==
S: + YGgGCSqGSIb3EgECAgIAb1kwV6ADAgEFoQMCAQ+iSzBJoAMC
   AQGiQgRAtHTEuOP2BXb9sBYFR4SJlDZxmg39IxmRBOhXRKdDA0
   uHTCOT9Bq3OsUTXUlk0CsFLoa8j+gvGDlgHuqzWHPSQg==
C:
S: + YDMGCSqGSIb3EgECAgIBAAD/////6jcyG4GE3KkTzBeBiVHe
   ceP2CWY0SR0fAQAgAAQEBAQ=
C: YDMGCSqGSIb3EgECAgIBAAD/////3LQBHXTpFfZgrejpLlLImP
   wkhbfa2QteAQAgAG1yYwE=
S: A001 OK GSSAPI authentication successful
```

The following example demonstrates the use of an initial response.

Example:

```
S: * OK [CAPABILITY IMAP4rev2 STARTTLS AUTH=GSSAPI
 LOGINDISABLED] Server ready
C: A01 STARTTLS
S: A01 OK STARTTLS completed
<TLS negotiation, further commands are under TLS layer>
C: A02 CAPABILITY
S: * CAPABILITY IMAP4rev2 AUTH=GSSAPI AUTH=PLAIN
S: A02 OK CAPABILITY completed
C: A03 AUTHENTICATE PLAIN dGVzdAB0ZXN0AHRlc3Q=
S: A03 OK Success (tls protection)
```

Note that because the initial response is optional, the following negotiation (which does not use the initial response) is still valid and MUST be supported by the server:

```
... client connects to server and negotiates a TLS
    protection layer ...
C: C01 CAPABILITY
S: * CAPABILITY IMAP4rev2 AUTH=PLAIN
S: C01 OK Completed
C: A01 AUTHENTICATE PLAIN
S: +
C: dGVzdAB0ZXN0AHRlc3Q=
S: A01 OK Success (tls protection)
```

Note that in the above example there is a space following the "+" from the server.

The following is an example authentication using the SASL EXTERNAL mechanism (defined in [SASL]) under a TLS protection layer and an empty initial response:

```
... client connects to server and negotiates a TLS
    protection layer ...
C: C01 CAPABILITY
S: * CAPABILITY IMAP4rev2 AUTH=PLAIN AUTH=EXTERNAL
S: C01 OK Completed
C: A01 AUTHENTICATE EXTERNAL =
S: A01 OK Success (tls protection)
```

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. The LOGIN command SHOULD NOT be used except as a last resort (after attempting and failing to authenticate using the AUTHENTICATE command one or more times), and it is recommended that client implementations have a means to disable any automatic use of the LOGIN command.

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. For that reason, clients MUST NOT use LOGIN on unsecure networks.

Unless the client is accessing IMAP service on an Implicit TLS 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 that 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, SELECT and EXAMINE 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, 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 (see Section 7.1) 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 will be able to correctly parse and process the extension response data.

The ENABLE command provides an explicit indication from the client that it supports particular extensions. It is designed such that the client can send a simple constant string with the extensions it supports, and the server will enable the shared subset that both support.

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), which includes all enabled extensions as specified above. The ENABLED response is sent even if no extensions were enabled.

Clients SHOULD only include extensions that need to be enabled by the server. For example, a client can enable IMAP4rev2-specific behavior 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; that is, "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, i.e., for the above example, the ENABLED response to "ENABLE c" should not contain "a" or "b".

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; that is, 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. Note that below "X-GOOD-IDEA" is a fictitious extension capability that can be ENABLED.

```
C: t1 CAPABILITY
S: * CAPABILITY IMAP4rev2 ID LITERAL+ 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 IMAP4rev2 ID LITERAL+ X-GOOD-IDEA
S: t3 OK foo again
```

In the following example, the client enables the Conditional Store (CONDSTORE) extension [RFC7162]:

```
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, LIST
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. (The order of individual responses is not important.) Note that earlier versions of this protocol, such as the IMAP4rev1 version specified in [RFC2060], only required the FLAGS and EXISTS untagged responses and UIDVALIDITY response code. Client implementations that need to remain compatible with such older IMAP versions have to implement default behavior for missing data, as discussed with the individual items.

FLAGS
: Defined flags in the mailbox. See the description of the FLAGS response in Section 7.3.5 for more detail.

<n> EXISTS
: The number of messages in the mailbox. See the description of the EXISTS response in Section 7.4.1 for more detail.

LIST
: The server **MUST** return a LIST response with the mailbox name. The list of mailbox attributes **MUST** be accurate. If the server allows denormalized UTF-8 mailbox names (see Section 5.1) and the supplied mailbox name differs from the normalized version, the server **MUST** return LIST with the OLDNAME extended data item. See Section 6.3.9.7 for more details.

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.

OK [UIDVALIDITY <n>]
: The unique identifier validity value. Refer to Section 2.3.1.1 for more information.

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 Section 7.1).

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: * LIST () "/" INBOX
S: A142 OK [READ-WRITE] SELECT completed
```

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
[...some time later...]
C: A143 SELECT Drafts
S: * OK [CLOSED] Previous mailbox is now closed
S: * 5 EXISTS
S: * OK [UIDVALIDITY 9877410381] UIDs valid
S: * OK [UIDNEXT 102] Predicted next UID
S: * LIST () "/" Drafts
S: * FLAGS (\Answered \Flagged \Deleted \Seen \Draft)
S: * OK [PERMANENTFLAGS (\Deleted \Seen \Answered
    \Flagged \Draft \*)] System flags and keywords allowed
S: A143 OK [READ-WRITE] SELECT completed
```

Note that IMAP4rev1-compliant servers can also send the untagged RECENT response that was deprecated in IMAP4rev2, e.g., "* 0 RECENT". Pure IMAP4rev2 clients are advised to ignore the untagged RECENT response.

6.3.3. EXAMINE Command

Arguments: mailbox name

Responses: REQUIRED untagged responses: FLAGS, EXISTS, LIST
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: * LIST () "/" blurdybloop
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: OPTIONAL untagged response: LIST

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 server decides to convert (normalize) the name, it SHOULD return an untagged LIST with an OLDNAME extended data item, with the OLDNAME value being the supplied mailbox name and the name parameter being the normalized mailbox name. (See Section 6.3.9.7 for more details.)

Mailboxes created in one IMAP session MAY be announced to other IMAP sessions using an unsolicited LIST response. If the server automatically subscribes a mailbox when it is created, then the unsolicited LIST response for each affected subscribed mailbox name MUST include the \Subscribed attribute.

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 that 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 in Section 6.4.9 for more detail.

Example:

```
C: A003 CREATE owatagusiam/
S: A003 OK CREATE completed
C: A004 CREATE owatagusiam/blurdybloop
S: A004 OK CREATE completed
C: A005 CREATE NonNormalized
S: * LIST () "/" "Normalized" ("OLDNAME" ("NonNormalized"))
S: A005 OK CREATE completed
```

(In the last example, imagine that "NonNormalized" is a non-NFC normalized Unicode mailbox name and that "Normalized" is its NFC normalized version.)

> 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: OPTIONAL untagged response: LIST

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 (Section 7.3.1) 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 exist, then it SHOULD disallow the DELETE command by returning a tagged NO response. The NO response SHOULD include the HASCHILDREN response code. Alternatively, the server MAY allow the DELETE command, but it sets the \Noselect mailbox name attribute for that name.

If the server returns an OK response, 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 in Section

6.4.9 for more detail.

If the server decides to convert (normalize) the mailbox name, it SHOULD return an untagged LIST with the "\NonExistent" attribute and OLDNAME extended data item, with the OLDNAME value being the supplied mailbox name and the name parameter being the normalized mailbox name. (See Section 6.3.9.7 for more details.)

Mailboxes deleted in one IMAP session MAY be announced to other IMAP sessions using an unsolicited LIST response, containing the "\NonExistent" attribute.

Example:

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

Example:

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

6.3.6. RENAME Command

Arguments: existing mailbox name

new mailbox name

Responses: OPTIONAL untagged response: LIST

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 new mailbox 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 in the corresponding namespace 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 in Section 6.4.9 for more detail.

Renaming INBOX is permitted and does not result in a tagged BAD response, and it 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. (Note that some servers disallow renaming INBOX by returning a tagged NO response, so clients need to be able to handle the failure of such RENAME commands.)

If the server allows creation of mailboxes with names that are not valid Net-Unicode names, the server normalizes both the existing mailbox name parameter and the new mailbox name parameter. If the normalized version of any of these 2 parameters differs from the corresponding supplied version, the server SHOULD return an untagged LIST response with an OLDNAME extended data item, with the OLDNAME value being the supplied existing mailbox name and the name parameter being the normalized new mailbox name (see Section 6.3.9.7). This would allow the client to correlate the supplied name with the normalized name.

Mailboxes renamed in one IMAP session MAY be announced to other IMAP sessions using an unsolicited LIST response with an OLDNAME extended data item.

In both of the above cases, if the server automatically subscribes a mailbox when it is renamed, then the unsolicited LIST response for each affected subscribed mailbox name MUST include the \Subscribed attribute. No unsolicited LIST responses need to be sent for child mailboxes. When INBOX is successfully renamed, it is assumed that a new INBOX is created. No unsolicited LIST responses need to be sent for INBOX in this case.

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
```

Note that renaming a mailbox doesn't update subscription information on the original name. To keep subscription information in sync, the following sequence of commands can be used:

```
C: 1001 RENAME X Y
C: 1002 SUBSCRIBE Y
C: 1003 UNSUBSCRIBE X
```

Note that the above sequence of commands doesn't account for updating the subscription for any child mailboxes of mailbox X.

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 LIST (SUBSCRIBED) command. This command returns a tagged OK response if the subscription is successful or if the mailbox is already subscribed.

A server MAY validate the mailbox argument to SUBSCRIBE to verify that it exists. However, it SHOULD 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 LIST (SUBSCRIBED) command. This command returns a tagged OK response if the unsubscription is successful or if the mailbox is not subscribed.

Example:

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

6.3.9. LIST Command

Arguments (basic):
reference name
mailbox name with possible wildcards

Arguments (extended):
selection options (OPTIONAL)
reference name
mailbox patterns
return options (OPTIONAL)

Responses: untagged responses: LIST

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

The LIST command returns a subset of mailbox names from the complete set of all mailbox names available to the client. Zero or more untagged LIST responses are returned, containing the name attributes, hierarchy delimiter, name, and possible extension information; see the description of the LIST response (Section 7.3.1) 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!

The extended LIST command, originally introduced in [RFC5258], provides capabilities beyond that of the original IMAP LIST command. The extended syntax is being used if one or more of the following conditions is true:

1. the first word after the command name begins with a parenthesis ("LIST selection options");
2. the second word after the command name begins with a parenthesis; and
3. the LIST command has more than 2 parameters ("LIST return options").

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(s). A non-empty reference name argument is the name of a mailbox or a level of mailbox hierarchy, and it indicates the context in which the mailbox name is interpreted. Clients SHOULD use the empty reference argument.

In the basic syntax only, 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.

In the extended syntax, any mailbox name arguments that are empty strings are ignored. There is no special meaning for empty mailbox names when the extended syntax is used.

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, which we call a "canonical LIST pattern": the canonical pattern constructed internally by the server from the reference and mailbox name arguments.

> Note: The interpretation of the reference argument is implementation defined. It depends on 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 that exports a UNIX or NT file system, the reference argument contains the current working directory, and the mailbox name argument contains 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 interpreted as implementation dependent. 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.

Here are some examples of how references and mailbox names might be interpreted on a UNIX-based server:

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

Table 1

The first three examples above 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 (Section 7.3.1) for more details).

Any syntactically valid pattern that is not accepted by a server for any reason MUST be silently ignored, i.e., it results in no LIST responses, and the LIST command still returns a tagged OK response.

Selection options tell the server to limit the mailbox names that are selected by the LIST operation. If selection options are used, the mailboxes returned are those that match both the list of canonical LIST patterns and the selection options. Unless a particular selection option provides special rules, the selection options are cumulative: a mailbox that matches the mailbox patterns is selected only if it also matches all of the selection options. (An example of a selection option with special rules is the RECURSIVEMATCH option.)

Return options control what information is returned for each matched mailbox. Return options MUST NOT cause the server to report information about additional mailbox names other than those that match the canonical LIST patterns and selection options. If no return options are specified, the client is only expecting information about mailbox attributes. The server MAY return other information about the matched mailboxes, and clients MUST be able to handle that situation.

Initial selection options and return options are defined in the following subsections, and new ones will also be defined in extensions. Initial options defined in this document MUST be supported. Each non-initial option will be enabled by a capability string (one capability may enable multiple options), and a client MUST NOT send an option for which the server has not advertised support. A server MUST respond to options it does not

recognize with a BAD response. The client SHOULD NOT specify any option more than once; however, if the client does this, the server MUST act as if it received the option only once. The order in which options are specified by the client is not significant.

In general, each selection option except RECURSIVEMATCH will have a corresponding return option with the same name. The REMOTE selection option is an anomaly in this regard and does not have a corresponding return option. That is because it expands, rather than restricts, the set of mailboxes that are returned. Future extensions to this specification should keep this parallelism in mind and define a pair of corresponding selection and return options.

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 a failure; it is not relevant whether the user's real INBOX resides on this or some other server.

6.3.9.1. LIST Selection Options

The selection options defined in this specification are as follows:

SUBSCRIBED
: Causes the LIST command to list subscribed names rather than the existing mailboxes. This will often be a subset of the actual mailboxes. It's also possible for this list to contain the names of mailboxes that don't exist. In any case, the list MUST include exactly those mailbox names that match the canonical list pattern and are subscribed to.

  This option defines a mailbox attribute, "\Subscribed", that indicates that a mailbox name is subscribed to. The "\Subscribed" attribute MUST be supported and MUST be accurately computed when the SUBSCRIBED selection option is specified.

  Note that the SUBSCRIBED selection option implies the SUBSCRIBED return option (see below).

REMOTE
: Causes the LIST command to show remote mailboxes as well as local ones, as described in [RFC2193]. This option is intended to replace the RLIST command and, in conjunction with the SUBSCRIBED selection option, the RLSUB command. Servers that don't support the concept of remote mailboxes can ignore this option.

  This option defines a mailbox attribute, "\Remote", that indicates that a mailbox is a remote mailbox. The "\Remote" attribute MUST be accurately computed when the REMOTE option is specified.

  The REMOTE selection option has no interaction with other options. Its effect is to tell the server to apply the other options, if any, to remote mailboxes, in addition to local ones. In particular, it has no interaction with RECURSIVEMATCH (see below). A request for (REMOTE RECURSIVEMATCH) is invalid, because a request for (RECURSIVEMATCH) is also invalid. A request for (REMOTE RECURSIVEMATCH SUBSCRIBED) is asking for all subscribed mailboxes, both local and remote.

RECURSIVEMATCH
Forces the server to return information about parent mailboxes that don't match other selection options but have some submailboxes that do. Information about children is returned in the CHILDINFO extended data item, as described in Section 6.3.9.6.

Note 1: In order for a parent mailbox to be returned, it still has to match the canonical LIST pattern.

Note 2: When returning the CHILDINFO extended data item, it doesn't matter whether or not the submailbox matches the canonical LIST pattern. See also Example 9 in Section 6.3.9.8.

The RECURSIVEMATCH option MUST NOT occur as the only selection option (or only with REMOTE), as it only makes sense when other selection options are also used. The server MUST return a BAD tagged response in such case.

Note that even if the RECURSIVEMATCH option is specified, the client MUST still be able to handle cases when a CHILDINFO extended data item is returned and there are no submailboxes that meet the selection criteria of the subsequent LIST command, as they can be deleted/renamed after the LIST response was sent but before the client had a chance to access them.

6.3.9.2. LIST Return Options

The return options defined in this specification are as follows:

SUBSCRIBED
Causes the LIST command to return subscription state for all matching mailbox names. The "\Subscribed" attribute MUST be supported and MUST be accurately computed when the SUBSCRIBED return option is specified. Furthermore, all other mailbox attributes MUST be accurately computed (this differs from the behavior of the obsolete LSUB command from [RFC3501]). Note that the above requirements don't override the requirement for the LIST command to return results quickly (see Section 6.3.9), i.e., server implementations need to compute results quickly and accurately. For example, server implementors might need to create quick access indices.

CHILDREN
Requests mailbox child information as originally proposed in [RFC3348]. See Section 6.3.9.5, below, for details.

STATUS
Requests STATUS response for each matching mailbox.

This option takes STATUS data items as parameters. For each selectable mailbox matching the list pattern and selection options, the server MUST return an untagged LIST response followed by an untagged STATUS response containing the information requested in the STATUS return option, except for some cases described below.

If an attempted STATUS for a listed mailbox fails because the mailbox can't be selected (e.g., if the "l" Access Control List (ACL) right [RFC4314] is granted to the mailbox and the "r" right is not granted, or is due to a race condition between LIST and STATUS changing the mailbox to \NoSelect), the STATUS response MUST NOT be returned, and the LIST response MUST include the \NoSelect attribute. This means the server may have to buffer the LIST reply until it has successfully looked up the necessary STATUS information.

If the server runs into unexpected problems while trying to look up the STATUS information, it MAY drop the corresponding STATUS reply. In such a situation, the LIST command would still return a tagged OK reply.

See the note in the discussion of the STATUS command in Section 6.3.11 for information about obtaining status on the currently selected mailbox.

6.3.9.3. General Principles for Returning LIST Responses

This section outlines several principles that can be used by server implementations of this document to decide whether a LIST response should be returned, as well as how many responses and what kind of information they may contain.

1. At most, one LIST response should be returned for each mailbox name that matches the canonical LIST pattern. Server implementors must not assume that clients will be able to assemble mailbox attributes and other information returned in multiple LIST responses.
2. There are only two reasons for including a matching mailbox name in the responses to the LIST command (note that the server is allowed to return unsolicited responses at any time, and such responses are not governed by this rule):
   A. The mailbox name also satisfies the selection criteria.
   B. The mailbox name doesn't satisfy the selection criteria, but it has at least one descendant mailbox name that satisfies the selection criteria and that doesn't match the canonical LIST pattern.

      For more information on this case, see the CHILDINFO extended data item described in Section 6.3.9.6. Note that the CHILDINFO extended data item can only be returned when the RECURSIVEMATCH selection option is specified.

3. Attributes returned in the same LIST response are treated additively. For example, the following response

```
S: * LIST (\Subscribed \NonExistent) "/" "Fruit/Peach"
```

   means that the "Fruit/Peach" mailbox doesn't exist, but it is subscribed.

6.3.9.4. Additional LIST-Related Requirements on Clients

All clients **MUST** treat a LIST attribute with a stronger meaning as implying any attribute that can be inferred from it. (See Section 7.3.1 for the list of currently defined attributes.) For example, the client must treat the presence of the \NoInferiors attribute as if the \HasNoChildren attribute was also sent by the server.

The following table summarizes inference rules.

returned attribute	implied attribute
\NoInferiors	\HasNoChildren
\NonExistent	\NoSelect

Table 2

6.3.9.5. The CHILDREN Return Option

The CHILDREN return option is simply an indication that the client wants information about whether or not mailboxes contain child mailboxes; a server **MAY** provide it even if the option is not specified.

Many IMAP clients present the user with a hierarchical view of the mailboxes that a user has access to. Rather than initially presenting the entire mailbox hierarchy to the user, it is often preferable to show the user a collapsed outline list of the mailbox hierarchy (particularly if there is a large number of mailboxes). The user can then expand the collapsed outline hierarchy as needed. It is common to include a visual clue (such as a

''+'') within the collapsed hierarchy to indicate that there are child mailboxes under a particular mailbox. When the visual clue is clicked, the hierarchy list is expanded to show the child mailboxes. The CHILDREN return option provides a mechanism for a client to efficiently determine whether a particular mailbox has children, without issuing a LIST "" * or a LIST "" % for each mailbox name. The CHILDREN return option defines two new attributes that MUST be returned within a LIST response: \HasChildren and \HasNoChildren. Although these attributes MAY be returned in response to any LIST command, the CHILDREN return option is provided to indicate that the client particularly wants this information. If the CHILDREN return option is present, the server MUST return these attributes even if their computation is expensive.

\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 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 child 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.

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.

6.3.9.6. CHILDINFO Extended Data Item

The CHILDINFO extended data item MUST NOT be returned unless the client has specified the RECURSIVEMATCH selection option.

The CHILDINFO extended data item in a LIST response describes the selection criteria that has caused it to be returned and indicates that the mailbox has at least one descendant mailbox that matches the selection criteria.

Note: Some servers allow for mailboxes to exist without requiring their parent to exist. For example, the mailbox "Customers/ABC" can exist while the mailbox "Customers" does not. As the CHILDINFO extended data item is not allowed if the RECURSIVEMATCH selection option is not specified, such servers SHOULD use the "\NonExistent \HasChildren" attribute pair to signal to the client that there is a descendant mailbox that matches the selection criteria. See Example 11 in Section 6.3.9.8.

The returned selection criteria allows the client to distinguish a solicited response from an unsolicited one, as well as to distinguish among solicited responses caused by multiple pipelined LIST commands that specify different criteria.

Servers SHOULD only return a non-matching mailbox name along with CHILDINFO if at least one matching child is not also being returned. That is, servers SHOULD suppress redundant CHILDINFO responses.

Examples 8 and 10 in Section 6.3.9.8 demonstrate the difference between the present CHILDINFO extended data item and the "\HasChildren" attribute.

The following table summarizes interaction between the "\NonExistent" attribute and CHILDINFO (the first column indicates whether the parent mailbox exists):

Exists	Meets the selection criteria	Has a child that meets the selection criteria	Returned IMAP4rev2/LIST-EXTENDED attributes and CHILDINFO
no	no	no	no LIST response returned
yes	no	no	no LIST response returned
no	yes	no	(\NonExistent <attr>)
yes	yes	no	(<attr>)
no	no	yes	(\NonExistent) + CHILDINFO
yes	no	yes	() + CHILDINFO
no	yes	yes	(\NonExistent <attr>) + CHILDINFO
yes	yes	yes	(<attr>) + CHILDINFO

Table 3

where <attr> is one or more attributes that correspond to the selection criteria; for example, for the SUBSCRIBED option, the <attr> is \Subscribed.

6.3.9.7. OLDNAME Extended Data Item

The OLDNAME extended data item is included when a mailbox name is created (with the CREATE command), renamed (with the RENAME command), or deleted (with the DELETE command). (When a mailbox is deleted, the "\NonExistent" attribute is also included.) IMAP extensions can specify other conditions when the OLDNAME extended data item should be included.

If the server allows denormalized mailbox names (see Section 5.1) in SELECT/EXAMINE, CREATE, RENAME, or DELETE, it SHOULD return an unsolicited LIST response that includes the OLDNAME extended data item, whenever the supplied mailbox name differs from the resulting normalized mailbox name. From the client point of view, this is indistinguishable from another user renaming or deleting the mailbox, as specified in the previous paragraph.

A deleted mailbox can be announced as follows:

```
S: * LIST (\NonExistent) "." "INBOX.DeletedMailbox"
```

Example of a renamed mailbox:

```
S: * LIST () "/" "NewMailbox" ("OLDNAME" ("OldMailbox"))
```

6.3.9.8. LIST Command Examples

This example shows some uses of the basic LIST command:

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
```

Extended examples:

1: The first example shows the complete local hierarchy that will be used for the other examples.

```
C: A01 LIST "" "*"
S: * LIST (\Marked \NoInferiors) "/" "inbox"
S: * LIST () "/" "Fruit"
S: * LIST () "/" "Fruit/Apple"
S: * LIST () "/" "Fruit/Banana"
S: * LIST () "/" "Tofu"
S: * LIST () "/" "Vegetable"
S: * LIST () "/" "Vegetable/Broccoli"
S: * LIST () "/" "Vegetable/Corn"
S: A01 OK done
```

2: In the next example, we will see the subscribed mailboxes. This is similar to, but not equivalent with, the now deprecated <LSUB "" "*"> (see [RFC3501] for more details on the LSUB command). Note that the mailbox called "Fruit/Peach" is subscribed to, but it does not actually exist (perhaps it was deleted while still subscribed). The "Fruit" mailbox is not subscribed to, but it has two subscribed children. The "Vegetable" mailbox is subscribed and has two children; one of them is subscribed as well.

```
C: A02 LIST (SUBSCRIBED) "" "*"
S: * LIST (\Marked \NoInferiors \Subscribed) "/" "inbox"
S: * LIST (\Subscribed) "/" "Fruit/Banana"
S: * LIST (\Subscribed \NonExistent) "/" "Fruit/Peach"
S: * LIST (\Subscribed) "/" "Vegetable"
S: * LIST (\Subscribed) "/" "Vegetable/Broccoli"
S: A02 OK done
```

3: The next example shows the use of the CHILDREN option. The client, without having to list the second level of hierarchy, now knows which of the top-level mailboxes have submailboxes (children) and which do not. Note that it's not necessary for the server to return the \HasNoChildren attribute for the inbox,

because the \NoInferiors attribute already implies that and has a stronger meaning.

```
C: A03 LIST () "" "%" RETURN (CHILDREN)
S: * LIST (\Marked \NoInferiors) "/" "inbox"
S: * LIST (\HasChildren) "/" "Fruit"
S: * LIST (\HasNoChildren) "/" "Tofu"
S: * LIST (\HasChildren) "/" "Vegetable"
S: A03 OK done
```

4: In this example, we see more mailboxes that reside on another server. This is similar to the command <RLIST "" "%">.

```
C: A04 LIST (REMOTE) "" "%" RETURN (CHILDREN)
S: * LIST (\Marked \NoInferiors) "/" "inbox"
S: * LIST (\HasChildren) "/" "Fruit"
S: * LIST (\HasNoChildren) "/" "Tofu"
S: * LIST (\HasChildren) "/" "Vegetable"
S: * LIST (\Remote \HasNoChildren) "/" "Bread"
S: * LIST (\HasChildren \Remote) "/" "Meat"
S: A04 OK done
```

5: The following example also requests the server to include mailboxes that reside on another server. The server returns information about all mailboxes that are subscribed. This is similar to the command <RLSUB "" "*"> (see [RFC2193] for more details on RLSUB). We also see the use of two selection options.

```
C: A05 LIST (REMOTE SUBSCRIBED) "" "*"
S: * LIST (\Marked \NoInferiors \Subscribed) "/" "inbox"
S: * LIST (\Subscribed) "/" "Fruit/Banana"
S: * LIST (\Subscribed \NonExistent) "/" "Fruit/Peach"
S: * LIST (\Subscribed) "/" "Vegetable"
S: * LIST (\Subscribed) "/" "Vegetable/Broccoli"
S: * LIST (\Remote \Subscribed) "/" "Bread"
S: A05 OK done
```

6: The following example requests the server to include mailboxes that reside on another server. The server is asked to return subscription information for all returned mailboxes. This is different from the example above.

Note that the output of this command is not a superset of the output in the previous example, as it doesn't include a LIST response for the non-existent "Fruit/Peach".

```
C: A06 LIST (REMOTE) "" "*" RETURN (SUBSCRIBED)
S: * LIST (\Marked \NoInferiors \Subscribed) "/" "inbox"
S: * LIST () "/" "Fruit"
S: * LIST () "/" "Fruit/Apple"
S: * LIST (\Subscribed) "/" "Fruit/Banana"
S: * LIST () "/" "Tofu"
S: * LIST (\Subscribed) "/" "Vegetable"
S: * LIST (\Subscribed) "/" "Vegetable/Broccoli"
S: * LIST () "/" "Vegetable/Corn"
S: * LIST (\Remote \Subscribed) "/" "Bread"
S: * LIST (\Remote) "/" "Meat"
S: A06 OK done
```

7: The following example demonstrates the difference between the \HasChildren attribute and the CHILDINFO extended data item.

Let's assume there is the following hierarchy:

```
C: C01 LIST "" "*"
S: * LIST (\Marked \NoInferiors) "/" "inbox"
S: * LIST () "/" "Foo"
S: * LIST () "/" "Foo/Bar"
S: * LIST () "/" "Foo/Baz"
S: * LIST () "/" "Moo"
S: C01 OK done
```

If the client asks RETURN (CHILDREN), it will get this:

```
C: CA3 LIST "" "%" RETURN (CHILDREN)
S: * LIST (\Marked \NoInferiors) "/" "inbox"
S: * LIST (\HasChildren) "/" "Foo"
S: * LIST (\HasNoChildren) "/" "Moo"
S: CA3 OK done
```

A) Let's also assume that the mailbox "Foo/Baz" is the only subscribed mailbox. Then we get this result:

```
C: C02 LIST (SUBSCRIBED) "" "*"
S: * LIST (\Subscribed) "/" "Foo/Baz"
S: C02 OK done
```

Now, if the client issues <LIST (SUBSCRIBED) "" "%">, the server will return no mailboxes (as the mailboxes "Moo", "Foo", and "Inbox" are NOT subscribed). However, if the client issues this:

```
C: C04 LIST (SUBSCRIBED RECURSIVEMATCH) "" "%"
S: * LIST () "/" "Foo" ("CHILDINFO" ("SUBSCRIBED"))
S: C04 OK done
```

(that is, the mailbox "Foo" is not subscribed, but it has a child that is), then A1 or A2 occurs.

A1) If the mailbox "Foo" had also been subscribed, the last command would return this:

```
C: C04 LIST (SUBSCRIBED RECURSIVEMATCH) "" "%"
S: * LIST (\Subscribed) "/" "Foo" ("CHILDINFO"
    ("SUBSCRIBED"))
S: C04 OK done
```

or even this:

```
C: C04 LIST (SUBSCRIBED RECURSIVEMATCH) "" "%"
S: * LIST (\Subscribed \HasChildren) "/" "Foo"
    ("CHILDINFO" ("SUBSCRIBED"))
S: C04 OK done
```

A2) If we assume instead that the mailbox "Foo" is not part of the original hierarchy and is not subscribed, the last command will give this result:

```
C: C04 LIST (SUBSCRIBED RECURSIVEMATCH) "" "%"
S: * LIST (\NonExistent) "/" "Foo" ("CHILDINFO"
    ("SUBSCRIBED"))
S: C04 OK done
```

B) Now, let's assume that no mailbox is subscribed. In this case, the command <LIST (SUBSCRIBED RECURSIVEMATCH) "" "%"> will return no responses, as there are no subscribed children (even though "Foo" has children).

C) And finally, suppose that only the mailboxes "Foo" and "Moo" are subscribed. In that case, we see this result:

```
C: C04 LIST (SUBSCRIBED RECURSIVEMATCH) "" "%" RETURN
    (CHILDREN)
S: * LIST (\HasChildren \Subscribed) "/" "Foo"
S: * LIST (\HasNoChildren \Subscribed) "/" "Moo"
S: C04 OK done
```

(which means that the mailbox "Foo" has children, but none of them is subscribed).

8: The following example demonstrates that the CHILDINFO extended data item is returned whether or not child mailboxes match the canonical LIST pattern.

Let's assume there is the following hierarchy:

```
C: D01 LIST "" "*"
S: * LIST (\Marked \NoInferiors) "/" "inbox"
S: * LIST () "/" "foo2"
S: * LIST () "/" "foo2/bar1"
S: * LIST () "/" "foo2/bar2"
S: * LIST () "/" "baz2"
S: * LIST () "/" "baz2/bar2"
S: * LIST () "/" "baz2/bar22"
S: * LIST () "/" "baz2/bar222"
S: * LIST () "/" "eps2"
S: * LIST () "/" "eps2/mamba"
S: * LIST () "/" "qux2/bar2"
S: D01 OK done
```

And that the following mailboxes are subscribed:

```
C: D02 LIST (SUBSCRIBED) "" "*"
S: * LIST (\Subscribed) "/" "foo2/bar1"
S: * LIST (\Subscribed) "/" "foo2/bar2"
S: * LIST (\Subscribed) "/" "baz2/bar2"
S: * LIST (\Subscribed) "/" "baz2/bar22"
S: * LIST (\Subscribed) "/" "baz2/bar222"
S: * LIST (\Subscribed) "/" "eps2"
S: * LIST (\Subscribed) "/" "eps2/mamba"
S: * LIST (\Subscribed) "/" "qux2/bar2"
S: D02 OK done
```

The client issues the following command first:

```
C: D03 LIST (RECURSIVEMATCH SUBSCRIBED) "" "*2"
S: * LIST () "/" "foo2" ("CHILDINFO" ("SUBSCRIBED"))
S: * LIST (\Subscribed) "/" "foo2/bar2"
S: * LIST (\Subscribed) "/" "baz2/bar2"
S: * LIST (\Subscribed) "/" "baz2/bar22"
S: * LIST (\Subscribed) "/" "baz2/bar222"
S: * LIST (\Subscribed) "/" "eps2" ("CHILDINFO" ("SUBSCRIBED"))
S: * LIST (\Subscribed) "/" "qux2/bar2"
S: D03 OK done
```

and the server may also include the following (but this would violate a restriction in Section 6.3.9.6, because CHILDINFO is redundant):

```
S: * LIST () "/" "baz2" ("CHILDINFO" ("SUBSCRIBED"))
S: * LIST (\NonExistent) "/" "qux2" ("CHILDINFO" ("SUBSCRIBED"))
```

The CHILDINFO extended data item is returned for mailboxes "foo2", "baz2", and "eps2" because all of them have subscribed children, even though for the mailbox "foo2", only one of the two subscribed children matches the pattern; for the mailbox "baz2", all of the subscribed children match the pattern; and for the mailbox "eps2", none of the subscribed children match the pattern.

Note that if the client issues the following:

```
C: D03 LIST (RECURSIVEMATCH SUBSCRIBED) "" "*"
S: * LIST () "/" "foo2" ("CHILDINFO" ("SUBSCRIBED"))
S: * LIST (\Subscribed) "/" "foo2/bar1"
S: * LIST (\Subscribed) "/" "foo2/bar2"
S: * LIST () "/" "baz2" ("CHILDINFO" ("SUBSCRIBED"))
S: * LIST (\Subscribed) "/" "baz2/bar2"
S: * LIST (\Subscribed) "/" "baz2/bar22"
S: * LIST (\Subscribed) "/" "baz2/bar222"
S: * LIST (\Subscribed) "/" "eps2" ("CHILDINFO" ("SUBSCRIBED"))
S: * LIST (\Subscribed) "/" "eps2/mamba"
S: * LIST (\Subscribed) "/" "qux2/bar2"
S: D03 OK done
```

the LIST responses for mailboxes "foo2", "baz2", and "eps2" still have the CHILDINFO extended data item, even though this information is redundant and the client can determine it by itself.

9: The following example shows usage of an extended syntax for the mailbox pattern. It also demonstrates that the presence of the CHILDINFO extended data item doesn't necessarily imply \HasChildren.

```
C: a1 LIST "" ("foo")
S: * LIST () "/" foo
S: a1 OK done

C: a2 LIST (SUBSCRIBED) "" "foo/*"
S: * LIST (\Subscribed \NonExistent) "/" foo/bar
S: a2 OK done

C: a3 LIST (SUBSCRIBED RECURSIVEMATCH) "" foo RETURN (CHILDREN)
S: * LIST (\HasNoChildren) "/" foo ("CHILDINFO" ("SUBSCRIBED"))
S: a3 OK done
```

10: The following example shows how a server that supports missing mailbox hierarchy elements can signal to a client that didn't specify the RECURSIVEMATCH selection option that there is a child mailbox that matches the selection criteria.

```
C: a1 LIST (REMOTE) "" *
S: * LIST () "/" music/rock
S: * LIST (\Remote) "/" also/jazz
S: a1 OK done

C: a2 LIST () "" %
S: * LIST (\NonExistent \HasChildren) "/" music
S: a2 OK done

C: a3 LIST (REMOTE) "" %
S: * LIST (\NonExistent \HasChildren) "/" music
S: * LIST (\NonExistent \HasChildren) "/" also
S: a3 OK done

C: a3.1 LIST "" (% music/rock)
S: * LIST () "/" music/rock
S: a3.1 OK done
```

Because "music/rock" is the only mailbox under "music", there's no need for the server to also return "music". However, clients must handle both cases.

11: The following examples show use of the STATUS return option.

```
C: A01 LIST "" % RETURN (STATUS (MESSAGES UNSEEN))
S: * LIST () "."  "INBOX"
S: * STATUS "INBOX" (MESSAGES 17 UNSEEN 16)
S: * LIST () "." "foo"
S: * STATUS "foo" (MESSAGES 30 UNSEEN 29)
S: * LIST (\NoSelect) "." "bar"
S: A01 OK List completed.
```

The "bar" mailbox isn't selectable, so it has no STATUS reply.

```
C: A02 LIST (SUBSCRIBED RECURSIVEMATCH) "" % RETURN (STATUS
     (MESSAGES))
S: * LIST (\Subscribed) "."  "INBOX"
S: * STATUS "INBOX" (MESSAGES 17)
S: * LIST () "." "foo" (CHILDINFO ("SUBSCRIBED"))
S: A02 OK List completed.
```

The LIST reply for "foo" is returned because it has matching children, but no STATUS reply is returned because "foo" itself doesn't match the selection criteria.

6.3.10. 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. The namespace-response-extensions ABNF non-terminal is defined for extensibility and MAY be included in the NAMESPACE response.

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 "."

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

Example 3:

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

```
C: A001 NAMESPACE
S: * NAMESPACE (("" "/")) NIL (("Public Folders/" "/"))
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 create personal mailboxes or LIST 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 a server will typically 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, or just use the first Personal Namespace.

Example 6:

In this example, a server supports two Personal Namespaces. In addition to the regular Personal Namespace, the user has an additional Personal Namespace that allows 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 with the \Sent attribute (see Section 7.3.1). Furthermore, after a message is deleted from a mailbox, the client is configured to move that message to a mailbox with the \Trash attribute. The server signals with the \NonExistent mailbox attribute

that the corresponding mailboxes don't exist yet and that it is possible to create them. Once created, they could be used for \Sent or \Trash purposes, and the server will no longer include the \NonExistent mailbox attribute for them.

Note that this example demonstrates how some extension parameters can be passed to further describe the #mh namespace. See the fictitious "X-PARAM" extension parameter.

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

C: A002 LIST (SPECIAL-USE) "" "*"
S: * LIST (\NonExistent \Archive) "/" Archives
S: * LIST (\NonExistent \Drafts) "/" Drafts
S: * LIST (\NonExistent \Junk) "/" Junk
S: * LIST (\NonExistent \Sent) "/" "Sent Mail"
S: * LIST (\NonExistent \Trash) "/" "Deleted Items"
S: A002 OK LIST Completed

C: A003 LIST (SPECIAL-USE) "#mh/" "*"
S: * LIST (\NonExistent \Archive) "/" "#mh/Archives"
S: * LIST (\NonExistent \Drafts) "/" "#mh/Drafts"
S: * LIST (\NonExistent \Junk) "/" "#mh/Junk"
S: * LIST (\NonExistent \Sent) "/" "#mh/Sent Mail"
S: * LIST (\NonExistent \Trash) "/" "#mh/Deleted Items"
S: A003 OK LIST 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: A004 CREATE "Sent Mail"
S: A004 OK CREATE command completed
```

The client is designed so that it keeps two 'Deleted Items' mailboxes, one for each namespace.

```
C: A005 CREATE "Delete Items"
S: A005 OK CREATE command completed

C: A006 CREATE "#mh/Deleted Items"
S: A006 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 (("" "/")) (("#Users/" "/")) 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 "" "#Users/%"
S: A002 NO The requested item could not be found.

C: A003 LIST "" "#Users/Mike/%"
S: * LIST () "/" "#Users/Mike/INBOX"
S: * LIST () "/" "#Users/Mike/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 "" "~mark/%"
S: * LIST () "/" "~mark/INBOX"
S: * LIST () "/" "~mark/foo"
S: A002 OK LIST command completed
```

6.3.11. STATUS Command

Arguments: mailbox name

status data item names

Responses: REQUIRED untagged responses: STATUS

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

The STATUS command requests the status of the indicated mailbox. It does not change the currently selected mailbox, nor does it affect the state of any messages in the queried mailbox.

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 as "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. However, servers MUST be able to execute the STATUS command on the selected mailbox. (This might also implicitly happen when the STATUS return option is used in a LIST command.)
>
> The STATUS command MUST NOT be used as a "check for new messages in the selected mailbox" operation (refer to Sections 7 and 7.4.1 for more information about the proper method for new message checking).
>
> STATUS SIZE (see below) can take a significant amount of time, depending upon server implementation. Clients should use STATUS SIZE cautiously.

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 that do not have the \Seen flag set.

DELETED
: The number of messages that have the \Deleted 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 6.4.5) 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
```

6.3.12. APPEND Command

Arguments: mailbox name

OPTIONAL flag parenthesized list

OPTIONAL date/time string

message literal

Responses: OPTIONAL untagged response: LIST

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 [RFC5322] 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 bits using a [MIME-IMB] content transfer encoding.

> Note: There may be exceptions, such as draft messages, in which required [RFC5322] header fields 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 (other than possibly keeping the changed mailbox's UIDNEXT value); 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 cannot 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 returns an APPENDUID response code (see Section 7.1), unless otherwise specified below.

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 MUST 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 Section 7.1), the server MAY omit the APPENDUID response code as it is not meaningful.

If the mailbox is currently selected, 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 after one or more APPEND commands.

If the server decides to convert (normalize) the mailbox name, it SHOULD return an untagged LIST with an OLDNAME extended data item, with the OLDNAME value being the supplied mailbox name and the name parameter being the normalized mailbox name. (See Section 6.3.9.7 for more details.)

Example:

```
C: A003 APPEND saved-messages (\Seen) {326}
S: + Ready for literal data
C: Date: Mon, 7 Feb 1994 21:52:25 -0800 (PST)
C: From: Fred Foobar <foobar@Blurdybloop.example>
C: Subject: afternoon meeting
C: To: mooch@owatagu.siam.edu.example
C: Message-Id: <B27397-0100000@Blurdybloop.example>
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
```

Example:

```
C: A003 APPEND saved-messages (\Seen) {297+}
C: Date: Mon, 7 Feb 1994 21:52:25 -0800 (PST)
C: From: Fred Foobar <foobar@example.com>
C: Subject: afternoon meeting
C: To: mooch@example.com
C: Message-Id: <B27397-0100000@example.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 [APPENDUID 38505 3955] APPEND completed
C: A004 COPY 2:4 meeting
S: A004 OK [COPYUID 38505 304,319:320 3956:3958] Done
C: A005 UID COPY 305:310 meeting
S: A005 OK No matching messages, so nothing copied
C: A006 COPY 2 funny
S: A006 OK Done
C: A007 SELECT funny
S: * 1 EXISTS
S: * OK [UIDVALIDITY 3857529045] Validity session-only
S: * OK [UIDNEXT 2] Predicted next UID
S: * NO [UIDNOTSTICKY] Non-persistent UIDs
S: * FLAGS (\Answered \Flagged \Deleted \Seen \Draft)
S: * OK [PERMANENTFLAGS (\Deleted \Seen)] Limited
S: * LIST () "." funny
S: A007 OK [READ-WRITE] SELECT 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.

6.3.13. 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 would need to poll the server for changes to the selected mailbox (new mail, deletions, and 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 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 chooses to send unsolicited FETCH responses, they MUST include a 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 for other commands, 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 IDLE and reissue 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.

Example:

```
C: A001 SELECT INBOX
S: * FLAGS (\Deleted \Seen \Flagged)
S: * OK [PERMANENTFLAGS (\Deleted \Seen \Flagged)] Limited
S: * 3 EXISTS
S: * OK [UIDVALIDITY 1]
S: * OK [UIDNEXT 1]
S: * LIST () "/" INBOX
S: A001 OK [READ-WRITE] SELECT completed
C: A002 IDLE
S: + idling
...time passes; new mail arrives...
S: * 4 EXISTS
C: DONE
S: A002 OK IDLE terminated
...another client expunges message 2 now...
C: A003 FETCH 4 ALL
S: * 4 FETCH (...)
S: A003 OK FETCH completed
C: A004 IDLE
S: * 2 EXPUNGE
S: * 3 EXISTS
S: + idling
...time passes; another client expunges message 3...
S: * 3 EXPUNGE
S: * 2 EXISTS
...time passes; new mail arrives...
S: * 3 EXISTS
C: DONE
S: A004 OK IDLE terminated
C: A005 FETCH 3 ALL
S: * 3 FETCH (...)
S: A005 OK FETCH completed
C: A006 IDLE
```

6.4. Client Commands - Selected State

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

6.4.1. 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 it 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 as 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
```

6.4.2. 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 a session'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
```

6.4.3. EXPUNGE Command

Arguments: none

Responses: untagged responses: EXPUNGE

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

The EXPUNGE command permanently removes 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 (Section 7.5.1) for further explanation.

6.4.4. SEARCH Command

Arguments: OPTIONAL result specifier

OPTIONAL [CHARSET] specification

searching criteria (one or more)

Responses: OPTIONAL 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 as "()", ALL is assumed (see below). The order of individual options is arbitrary. Individual options may contain parameters enclosed in parentheses. (However, 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 parentheses. See "Formal Syntax" (Section 9) for more details.) 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.

Note that IMAP4rev1 used SEARCH responses [RFC3501] instead of ESEARCH responses. Clients that support only IMAP4rev2 MUST ignore SEARCH responses.

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

SAVE
: This option tells the server to remember the result of the SEARCH or UID SEARCH command (as well as any command based on SEARCH, e.g., SORT and THREAD [RFC5256]) and store it in an internal variable that we will reference as the "search result variable". The client can use the "$" marker to reference the content of this internal variable. The "$" marker can be used instead of message sequence or UID sequence in order to indicate that the server should substitute it with the list of messages from the search result variable. Thus, the client can use the result of the latest remembered SEARCH command as a parameter to another command. See Section 6.4.4.1 for details on how the value of the search result variable is determined, how it is affected by other commands executed, and how the SAVE return option interacts with other return options.

  In absence of any other SEARCH result option, the SAVE result option also suppresses any ESEARCH response that would have been otherwise returned by the SEARCH command.

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 if used by themselves or in combination. This guarantee simplifies processing in IMAP4rev2 clients. Future SEARCH extensions that relax this restriction will have to describe how results from multiple ESEARCH responses are to be combined.

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 with INTERNALDATE greater than 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 the name of a character set from the registry [CHARSET-REG]. It indicates the [CHARSET] of the strings that appear in the search criteria. [MIME-IMB] content transfer encodings and [MIME-HDRS] strings in [RFC5322]/[MIME-IMB] headers MUST be decoded before comparing text. Servers MUST support US-ASCII and UTF-8 charsets; other CHARSETs MAY be supported. Clients SHOULD use UTF-8. Note that if CHARSET is not provided, IMAP4rev2 servers MUST assume UTF-8, so selecting CHARSET UTF-8 is redundant. It is permitted for improved compatibility with existing IMAP4rev1 clients.

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 CHARSETs supported by the server.

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

The defined search keys are as follows. Refer to "Formal Syntax" (Section 9) 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 Blind Carbon Copy (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. Unlike TEXT (see below), this doesn't match any header fields. Servers are allowed to implement flexible matching for this search key, for example, by matching "swim" to both "swam" and "swum" in English language text or only performing full word matching (where "swim" will not match "swimming").

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 field with the specified field-name (as defined in [RFC5322]) and that contain the specified string in the text of the header field (what comes after the colon). If the string to search is zero-length, this matches all messages that have a header field with the specified field-name regardless of the contents. Servers should use a substring search for this SEARCH item, as clients can use it for automatic processing not initiated by end users. For example, this can be used when searching for Message-ID or Content-Type header field values that need to be exact or for searches in header fields that the IMAP server might not know anything about.

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

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

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 [RFC5322] Date: header field (disregarding time and timezone) is earlier than the specified date.

SENTON <date>
: Messages whose [RFC5322] Date: header field (disregarding time and timezone) is within the specified date.

SENTSINCE <date>
: Messages whose [RFC5322] Date: header field (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 RFC822.SIZE smaller than the specified number of octets.

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

TEXT <string>
Messages that contain the specified string in the header (including MIME header fields) or body of the message. Servers are allowed to implement flexible matching for this search key, for example, matching "swim" to both "swam" and "swum" in English language text or only performing full-word matching (where "swim" will not match "swimming").

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 {12}
S: + Ready for literal text
C: отпуск
S: * ESEARCH (TAG "A285") ALL 43
S: A285 OK SEARCH completed
```

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
```

6.4.4.1. SAVE Result Option and SEARCH Result Variable

Upon successful completion of a SELECT or an EXAMINE command (after the tagged OK response), the current search result variable is reset to the empty sequence.

A successful SEARCH command with the SAVE result option sets the value of the search result variable to the list of messages found in the SEARCH command. For example, if no messages were found, the search result variable will contain the empty sequence.

Any of the following SEARCH commands MUST NOT change the search result variable:

- a SEARCH command that caused the server to return the BAD tagged response,
- a SEARCH command with no SAVE result option that caused the server to return NO tagged response, and
- a successful SEARCH command with no SAVE result option.

A SEARCH command with the SAVE result option that caused the server to return the NO tagged response sets the value of the search result variable to the empty sequence.

When a message listed in the search result variable is EXPUNGEd, it is automatically removed from the list. Implementors are reminded that if the server stores the list as a list of message numbers, it MUST automatically adjust them when notifying the client about expunged messages, as described in Section 7.5.1.

If the server decides to send a new UIDVALIDITY value while the mailbox is opened, it causes the resetting of the search variable to the empty sequence.

Note that even if the "$" marker contains the empty sequence of messages, it must be treated by all commands accepting message sets as parameters as a valid, but non-matching, list of messages. For example, the "FETCH $" command would return a tagged OK response and no FETCH responses. See also Example 5 in Section 6.4.4.4.

The SAVE result option doesn't change whether the server would return items corresponding to MIN, MAX, ALL, or COUNT result options.

When the SAVE result option is combined with the MIN or MAX result option, and both ALL and COUNT result options are absent, the corresponding MIN/MAX is returned (if the search result is not empty), but the "$" marker would contain a single message as returned in the MIN/MAX return item.

If the SAVE result option is combined with both MIN and MAX result options, and both ALL and COUNT result options are absent, the "$" marker would contain zero messages, one message, or two messages as returned in the MIN/MAX return items.

If the SAVE result option is combined with the ALL and/or COUNT result option(s), the "$" marker would always contain all messages found by the SEARCH or UID SEARCH command.

The following table summarizes the additional requirement on ESEARCH server implementations described in this section.

Combination of Result Option	"$" Marker Value
SAVE MIN	MIN
SAVE MAX	MAX
SAVE MIN MAX	MIN & MAX
SAVE * [m]	all found messages

Table 4

where '*' means "ALL" and/or "COUNT", and '[m]' means optional "MIN" and/or "MAX"

Implementation note: server implementors should note that "$" can reference IMAP message sequences or UID sequences, depending on the context where it is used. For example, the "$" marker can be set as a result of a SEARCH (SAVE) command and used as a parameter to a UID FETCH command (which accepts a UID sequence, not a message sequence), or the "$" marker can be set as a result of a UID SEARCH (SAVE) command and used as a parameter to a FETCH command (which accepts a message sequence, not a UID sequence). Server implementations need to automatically map the "$" marker value to message numbers or UIDs, depending on the context where the "$" marker is used.

6.4.4.2. Multiple Commands in Progress

Use of a SEARCH RETURN (SAVE) command followed by a command using the "$" marker creates direct dependency between the two commands. As directed by Section 5.5, a server MUST execute the two commands in the order they were received.

A client MAY pipeline a SEARCH RETURN (SAVE) command with one or more commands using the "$" marker, as long as this doesn't create an ambiguity, as described in Section 5.5. Examples 7-9 in Section 6.4.4.4 explain this in more details.

6.4.4.3. Refusing to Save Search Results

In some cases, the server MAY refuse to save a SEARCH (SAVE) result, for example, if an internal limit on the number of saved results is reached. In this case, the server MUST return a tagged NO response containing the NOTSAVED response code and set the search result variable to the empty sequence, as described in Section 6.4.4.1.

6.4.4.4. Examples Showing Use of the SAVE Result Option

Only in this section: explanatory comments in examples that start with // are not part of the protocol.

1. The following example demonstrates how the client can use the result of a SEARCH command to FETCH headers of interesting messages:

   Example 1:

```
C: A282 SEARCH RETURN (SAVE) FLAGGED SINCE 1-Feb-1994
    NOT FROM "Smith"
S: A282 OK SEARCH completed, result saved
C: A283 FETCH $ (UID INTERNALDATE FLAGS BODY.PEEK[HEADER])
S: * 2 FETCH (UID 14 ...
S: * 84 FETCH (UID 100 ...
S: * 882 FETCH (UID 1115 ...
S: A283 OK completed
```

   The client can also pipeline the two commands:

   Example 2:

```
C: A282 SEARCH RETURN (SAVE) FLAGGED SINCE 1-Feb-1994
    NOT FROM "Smith"
C: A283 FETCH $ (UID INTERNALDATE FLAGS BODY.PEEK[HEADER])
S: A282 OK SEARCH completed
S: * 2 FETCH (UID 14 ...
S: * 84 FETCH (UID 100 ...
S: * 882 FETCH (UID 1115 ...
S: A283 OK completed
```

2. The following example demonstrates that the result of one SEARCH command can be used as input to another SEARCH command:

Example 3:

```
C: A300 SEARCH RETURN (SAVE) SINCE 1-Jan-2004
    NOT FROM "Smith"
S: A300 OK SEARCH completed
C: A301 UID SEARCH UID $ SMALLER 4096
S: * ESEARCH (TAG "A301") UID ALL 17,900,901
S: A301 OK completed
```

Note that the second command in Example 3 can be replaced with:

```
C: A301 UID SEARCH $ SMALLER 4096
```

and the result of the command would be the same.

3. The following example shows that the "$" marker can be combined with other message numbers using the OR SEARCH criterion.

   Example 4:

```
C: P282 SEARCH RETURN (SAVE) SINCE 1-Feb-1994
    NOT FROM "Smith"
S: P282 OK SEARCH completed
C: P283 SEARCH CHARSET UTF-8 (OR $ 1,3000:3021) TEXT {8+}
C: мать
S: * ESEARCH (TAG "P283") ALL 882,1102,3003,3005:3006
S: P283 OK completed
```

4. The following example demonstrates that a failed SEARCH sets the search result variable to the empty list. The server doesn't implement the KOI8-R charset.

   Example 5:

```
C: B282 SEARCH RETURN (SAVE) SINCE 1-Feb-1994
    NOT FROM "Smith"
S: B282 OK SEARCH completed
C: B283 SEARCH RETURN (SAVE) CHARSET KOI8-R
    (OR $ 1,3000:3021) TEXT {4}
C: XXXX
S: B283 NO [BADCHARSET UTF-8] KOI8-R is not supported
//After this command, the saved result variable contains
//no messages.  A client that wants to reissue the B283
//SEARCH command with another CHARSET would have to reissue
//the B282 command as well.  One possible workaround for
//this is to include the desired CHARSET parameter
//in the earliest SEARCH RETURN (SAVE) command in a
//sequence of related SEARCH commands, to cause
//the earliest SEARCH in the sequence to fail.
//A better approach might be to always use CHARSET UTF-8
//instead.
```

   Note: Since this document format is restricted to 7-bit ASCII text, it is not possible to show actual KOI8-R data. The "XXXX" is a placeholder for what would be 4 octets of 8-bit data in an actual transaction.

5. The following example demonstrates that it is not an error to use the "$" marker when it contains no messages.

Example 6:

```
  C: E282 SEARCH RETURN (SAVE) SINCE 28-Oct-2006
      NOT FROM "Eric"
  C: E283 COPY $ "Other Messages"
 //The "$" contains no messages
  S: E282 OK SEARCH completed
  S: E283 OK COPY completed, nothing copied
```

Example 7:

```
  C: F282 SEARCH RETURN (SAVE) KEYWORD $Junk
  C: F283 COPY $ "Junk"
  C: F284 STORE $ +FLAGS.Silent (\Deleted)
  S: F282 OK SEARCH completed
  S: F283 OK COPY completed
  S: F284 OK STORE completed
```

Example 8:

```
  C: G282 SEARCH RETURN (SAVE) KEYWORD $Junk
  C: G283 SEARCH RETURN (ALL) SINCE 28-Oct-2006
      FROM "Eric"
 // The server can execute the two SEARCH commands
 // in any order, as they don't have any dependency.
 // For example, it may return:
  S: * ESEARCH (TAG "G283") ALL 3:15,27,29:103
  S: G283 OK SEARCH completed
  S: G282 OK SEARCH completed
```

The following example demonstrates that the result of the second SEARCH RETURN (SAVE) always overrides the result of the first.

Example 9:

```
  C: H282 SEARCH RETURN (SAVE) KEYWORD $Junk
  C: H283 SEARCH RETURN (SAVE) SINCE 28-Oct-2006
      FROM "Eric"
  S: H282 OK SEARCH completed
  S: H283 OK SEARCH completed
 // At this point "$" would contain results of H283
```

The following example demonstrates behavioral difference for different combinations of ESEARCH result options.

Example 10:

```
  C: C282 SEARCH RETURN (ALL) SINCE 12-Feb-2006
      NOT FROM "Smith"
  S: * ESEARCH (TAG "C283") ALL 2,10:15,21
//$ value hasn't changed
  S: C282 OK SEARCH completed

  C: C283 SEARCH RETURN (ALL SAVE) SINCE 12-Feb-2006
      NOT FROM "Smith"
  S: * ESEARCH (TAG "C283") ALL 2,10:15,21
//$ value is 2,10:15,21
  S: C283 OK SEARCH completed

  C: C284 SEARCH RETURN (SAVE MIN) SINCE 12-Feb-2006
      NOT FROM "Smith"
  S: * ESEARCH (TAG "C284") MIN 2
//$ value is 2
  S: C284 OK SEARCH completed

  C: C285 SEARCH RETURN (MAX SAVE MIN) SINCE
      12-Feb-2006 NOT FROM "Smith"
  S: * ESEARCH (TAG "C285") MIN 2 MAX 21
//$ value is 2,21
  S: C285 OK SEARCH completed

  C: C286 SEARCH RETURN (MAX SAVE MIN COUNT)
      SINCE 12-Feb-2006 NOT FROM "Smith"
  S: * ESEARCH (TAG "C286") MIN 2 MAX 21 COUNT 8
//$ value is 2,10:15,21
  S: C286 OK SEARCH completed

  C: C286 SEARCH RETURN (ALL SAVE MIN) SINCE
      12-Feb-2006 NOT FROM "Smith"
  S: * ESEARCH (TAG "C286") MIN 2 ALL 2,10:15,21
//$ value is 2,10:15,21
  S: C286 OK SEARCH completed
```

6.4.5. 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 (Section 9) 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 that 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)

Several data items reference "section" or "section-binary". See Section 6.4.5.1 for their detailed definition.

The currently defined data items that can be fetched are:

BINARY[<section-binary>]<<partial>>
: Requests that the specified section be transmitted after performing decoding of the section's Content-Transfer-Encoding.

  The <partial> argument, if present, requests that a subset of the data be returned. The semantics of a partial FETCH BINARY command are the same as for a partial FETCH BODY command, with the exception that the <partial> arguments refer to the DECODED section data.

  Note that this data item can only be requested for leaf body parts: those that have media types other than multipart/*, message/rfc822, or message/global.

BINARY.PEEK[<section-binary>]<<partial>>
: An alternate form of BINARY[<section-binary>] that does not implicitly set the \Seen flag.

BINARY.SIZE[<section-binary>]
: Requests the decoded size of the section (i.e., the size to expect in response to the corresponding FETCH BINARY request).

  Note: client authors are cautioned that this might be an expensive operation for some server implementations. Needlessly issuing this request could result in degraded performance due to servers having to calculate the value every time the request is issued.

  Note that this data item can only be requested for leaf body parts: those that have media types other than multipart/*, message/rfc822, or message/global.

BODY
: Non-extensible form of BODYSTRUCTURE.

BODY[<section>]<<partial>>
: The text of a particular body section. If BODY[] is specified (the section specification is omitted), the FETCH is requesting the [RFC5322] expression of the entire message.

  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 [RFC5322] header and [MIME-IMB] headers. See Section 7.5.2 for more details.

ENVELOPE
The envelope structure of the message. This is computed by the server by parsing the [RFC5322] header into the component parts, defaulting various fields as necessary. See Section 7.5.2 for more details.

FLAGS
The flags that are set for this message.

INTERNALDATE
The internal date of the message.

RFC822.SIZE
The size of the message, as defined in Section 2.3.4.

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
```

6.4.5.1. FETCH Section Specification

Several FETCH data items reference "section" or "section-binary". 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. (Non-numeric part specifiers have to be the last specifier in a section specification.) An empty section specification refers to the entire message, including the header.

Every message has at least one part number. Messages that do not use MIME, and MIME messages that are not multipart and have no encapsulated message within them, 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 [RFC5322] 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-names (as defined in [RFC5322]) 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 is otherwise exact. Subsetting does not exclude the [RFC5322] 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 that 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 [RFC5322] header.

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

```
HEADER     ([RFC5322] header of the message)
TEXT       ([RFC5322] text body of the message) MULTIPART/MIXED
1          TEXT/PLAIN
2          APPLICATION/OCTET-STREAM
3          MESSAGE/RFC822
3.HEADER   ([RFC5322] header of the message)
3.TEXT     ([RFC5322] 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 ([RFC5322] header of the message)
4.2.TEXT   ([RFC5322] 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
```

6.4.6. 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
```

6.4.7. 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 MUST return an error. It 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 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 (other than possibly incrementing UIDNEXT), i.e., partial copy MUST NOT be done.

On successful completion of a COPY, the server returns a COPYUID response code (see Section 7.1). Two exceptions to this requirement are listed below.

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 MUST NOT send a COPYUID response code as it would disclose information about the mailbox.

In the case of a mailbox that has UIDNOTSTICKY status (see Section 7.1), the server MAY omit the COPYUID response code as it is not meaningful.

Example:

```
C: A003 COPY 2:4 MEETING
S: A003 OK [COPYUID 38505 304,319:320 3956:3958] COPY completed
```

6.4.8. 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.

Unlike the COPY command, MOVE of a set of messages might fail partway through the set. Regardless of whether the command is successful in moving the entire set, each individual message MUST be either 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.

If the destination mailbox does not exist, a server MUST return an error. It MUST NOT automatically create the mailbox. Unless it is certain that the destination mailbox cannot 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 MOVE if the CREATE is successful.

Because of the similarity of MOVE to COPY, extensions that affect COPY affect MOVE in the same way. Response codes listed in Section 7.1, as well as those defined by extensions, are sent as indicated for COPY.

Servers send COPYUID in response to a MOVE or a UID MOVE (see Section 6.4.9) command. For additional information about COPYUID, see Section 7.1. Note that there are several exceptions listed in Section 6.4.7 that allow servers not to return COPYUID.

Servers are also REQUIRED to send the COPYUID response code in an untagged OK before sending EXPUNGE or similar responses. (Sending COPYUID in the tagged OK, as described in Section 6.4.7, 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
...More EXPUNGE responses from the server...
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.

6.4.9. UID Command

Arguments: command name

command arguments

Responses: untagged responses: FETCH, ESEARCH, EXPUNGE

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 that specifies a sequence set of UIDs to operate on. The UID EXPUNGE command permanently removes all messages that have both the \Deleted flag set and 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 an 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

Each command that is not part of this specification MUST have at least one capability name (see Section 6.1.1) associated with it. (Multiple commands can be associated with the same capability name.)

Server implementations MUST NOT send any added untagged responses (not specified in this specification), unless the client requested it by issuing the associated experimental command (specified in an extension document) or the ENABLE command (Section 6.3.1).

The following example demonstrates how a client can check for the presence of a fictitious XPIG-LATIN capability that adds the XPIG-LATIN command and the XPIG-LATIN untagged response. (Note that for an extension, the command name and the capability name don't have to be the same.)

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 requests. 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 "Formal Syntax" (Section 9).

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 remembered by the client when it is received; this is noted in the description of that data. Such data conveys critical information that affects the interpretation of all subsequent commands and responses (e.g., updates reflecting the creation or destruction of messages).

Other server data SHOULD be remembered for later reference; if the client does not need to remember the data, or if remembering the data has no obvious purpose (e.g., a SEARCH response when no SEARCH command is in progress), the data can 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 an 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 - Generic 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 is presented to the user in a fashion that calls the user's attention to the message. Content of ALERT response codes received on a connection without TLS or SASL security-layer confidentiality SHOULD be ignored by clients. If displayed, such alerts MUST be clearly marked as potentially suspicious. (Note that some existing clients are known to hyperlink returned text, which make them very dangerous.) Alerts received after successful establishment of a TLS/SASL confidentiality layer MUST be presented to the user.

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

```
C: o356 RENAME this that
S: o356 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, it 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
: This operation violates some invariant of the server and can never 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. It can also appear in tagged responses to LOGIN or AUTHENTICATE commands. This makes it unnecessary for a client to send a separate CAPABILITY command if it recognizes this response code and there was no change to the TLS and/or authentication state since it was received.

CLIENTBUG
: The server has detected a client bug. This can accompany any 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 returns the CLOSED response code when the currently selected mailbox is closed implicitly using the SELECT or 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.

CONTACTADMIN
The user should contact the system administrator or support desk.

```
C: e login "fred" "foo"
S: e NO [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, followed by another UID set containing the UIDs assigned to the copied message(s) in the destination mailbox, indicates that the message(s) has been copied to the destination mailbox with the stated UID(s).

The source UID set is in the order the message(s) was 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 or UID COPY command or in the untagged OK response to the MOVE or UID MOVE 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
```

EXPIRED
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
```

EXPUNGEISSUED
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 maria@example.com
S: * ESEARCH (TAG "h") ALL 1:3,5,8,13,21,42
S: h OK [EXPUNGEISSUED] Search completed
```

HASCHILDREN
The mailbox delete operation failed because the mailbox has one or more children, and the server doesn't

allow deletion of mailboxes with children.

```
C: m356 DELETE Notes
S: o356 NO [HASCHILDREN] Mailbox "Notes" has children
that need to be deleted first
```

INUSE
: 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
```

LIMIT
: 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
```

NONEXISTENT
: 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., 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 [RFC5322] header or [MIME-IMB] headers of a message in the mailbox.

PERMANENTFLAGS
: Followed by a parenthesized list of flags and indicates which of the known flags the client can change permanently. Any flags that are in the FLAGS untagged response, but not in the PERMANENTFLAGS list, cannot be set permanently. The PERMANENTFLAGS list can also include the special flag *, which indicates that it is possible to create new keywords by attempting to store those keywords in the mailbox. 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.

  There is no need for a server that included the special flag * to return a new PERMANENTFLAGS response code when a new keyword was successfully set on a message upon client request. However, if the server has a limit on the number of different keywords that can be stored in a mailbox and that limit is reached, the server MUST send a new PERMANENTFLAGS response code without the special flag *.

PRIVACYREQUIRED
: The operation is not permitted due to a lack of data confidentiality. If TLS is not in use, the client could try STARTTLS (see Section 6.2.1) or alternatively reconnect on an Implicit TLS port, and then repeat the operation.

```
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 as read-only, or its access while selected has changed from read-write to read-only.

READ-WRITE
: The mailbox is selected as 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, COPY, or MOVE 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 and 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 and 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
```

UNKNOWN-CTE
: The server does not know how to decode the section's Content-Transfer-Encoding.

Client implementations MUST 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.

Because PREAUTH moves the connection directly to the authenticated state, it effectively prevents the client from using the STARTTLS command (Section 6.2.1). For this reason, the PREAUTH response SHOULD only be returned by servers on connections that are protected by TLS (such as on an Implicit TLS port [RFC8314]) or protected through other means such as IPsec. Clients that require mandatory TLS MUST close the connection after receiving the PREAUTH response on a non-protected port.

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 Status

These responses are always untagged. This is how server status data are transmitted from the server to the client.

7.2.1. 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.

Example:

```
S: * ENABLED CONDSTORE QRESYNC
```

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", but note that it doesn't have to be the first capability listed. The order of capability names has no significance.

Client and server implementations MUST implement the capabilities "AUTH=PLAIN" (described in [PLAIN]), and MUST implement "STARTTLS" and "LOGINDISABLED" on the cleartext port. See the Security Considerations (Section 11) for important information related to these capabilities.

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

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 (their validity will not be checked). 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. If IMAP4rev1 capability is not advertised, server responses MUST conform to this document until the client issues a command that uses an additional capability. If both IMAP4rev1 and IMAP4rev2 capabilities are advertised, server responses MUST conform to [RFC3501] until the client issues a command that uses an additional capability. (For example, the client can issue ENABLE IMAP4rev2 to enable IMAP4rev2-specific behavior.)

Capability names SHOULD be registered with IANA using the RFC Required policy [RFC8126]. A server SHOULD NOT offer unregistered capability names.

Client implementations SHOULD NOT require any capability name other than "IMAP4rev2", and possibly "STARTTLS" and "LOGINDISABLED" (on a cleartext port). Client implementations 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 and there was no change to the TLS and/or authentication state since they were received.

The list of capabilities returned by a server MAY change during the connection. In particular, it is quite common for the server to change the list of capabilities after successful TLS negotiation (STARTTLS command) and/or after successful authentication (AUTHENTICATE or LOGIN commands).

Example:

```
S: * CAPABILITY STARTTLS AUTH=GSSAPI IMAP4rev2 LOGINDISABLED
 XPIG-LATIN
```

Note that in the above example, XPIG-LATIN is a fictitious capability name.

7.3. Server Responses - Mailbox Status

These responses are always untagged. This is how mailbox status data are transmitted from the server to the client. Many of these responses typically result from a command with the same name.

7.3.1. LIST Response

Contents:

- name attributes
- hierarchy delimiter
- name
- OPTIONAL extension data

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 mailbox name attributes are defined:

\NonExistent
: The "\NonExistent" attribute indicates that a mailbox name does not refer to an existing mailbox. Note that this attribute is not meaningful by itself, as mailbox names that match the canonical LIST pattern but don't exist must not be returned unless one of the two conditions listed below is also satisfied:

  1. The mailbox name also satisfies the selection criteria (for example, it is subscribed and the "SUBSCRIBED" selection option has been specified).
  2. "RECURSIVEMATCH" has been specified, and the mailbox name has at least one descendant mailbox name that does not match the LIST pattern and does match the selection criteria.

In practice, this means that the "\NonExistent" attribute is usually returned with one or more of "\Subscribed", "\Remote", "\HasChildren", or the CHILDINFO extended data item.

The "\NonExistent" attribute implies "\NoSelect".

\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 mailboxes. Note that even though the \HasChildren attribute for a mailbox must be correct at the time of processing 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 child 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.

\Subscribed
: The mailbox name was subscribed to using the SUBSCRIBE command.

\Remote
: The mailbox is a remote mailbox.

It is an error for the server to return both a \HasChildren and a \HasNoChildren attribute in the same LIST response. A client that encounters a LIST response with both \HasChildren and \HasNoChildren attributes present should act as if both are absent in the 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 it MAY send none of these.

In addition to the base mailbox 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 as its trash location. In server implementations that strictly expect the IMAP "\Deleted" model, this special use is likely not to be supported.

All 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.

Other mailbox name attributes can be found in the "IMAP Mailbox Name Attributes" registry [IMAP-MAILBOX-NAME-ATTRS-REG].

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 command. Unless \Noselect or \NonExistent is indicated, the name MUST also be valid as an argument for commands, such as SELECT, that accept mailbox names.

The name might be followed by an OPTIONAL series of extended fields, a parenthesized list of tagged data (also referred to as an "extended data item"). The first element of an extended field is a string, which identifies the type of data. [RFC5258] specifies requirements on string registration (which are called "tags"; such tags are not to be confused with IMAP command tags); in particular, it states that "Tags MUST be registered with IANA". This document doesn't change that. See Section 9.5 of [RFC5258] for the registration template. The server MAY return data in the extended fields that was not directly solicited by the client in the corresponding LIST command. For example, the client can enable extra extended fields by using another IMAP extension that makes use of the extended LIST responses. The client MUST ignore all extended fields it doesn't recognize.

Example:

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

Example:

```
S: * LIST (\Marked) ":" Tables (tablecloth (("edge" "lacy")
    ("color" "red")) Sample "text")
S: * LIST () ":" Tables:new (tablecloth ("edge" "lacy")
    Sample ("text" "more text"))
```

7.3.2. 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. The Namespace-Response-Extensions ABNF non-terminal is defined for extensibility and MAY be included in the response.

Example:

```
S: * NAMESPACE (("" "/")) (("~" "/")) NIL
```

7.3.3. STATUS Response

Contents:

name

status parenthesized list

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

Example:

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

7.3.4. 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 a unique return item name, followed by a space and the corresponding data. Search data pairs may be returned in any order. Unless otherwise specified by an extension, any return item name SHOULD appear only once in an ESEARCH response.

This document specifies the following return item names:

MIN
: Returns the lowest message number/UID that satisfies the SEARCH criteria.

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

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

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

ALL
: Returns all message numbers/UIDs that satisfy the SEARCH criteria using the sequence-set syntax. Each set MUST be complete; in particular, a UID set is returned in an ESEARCH response only when each number in the range corresponds to an existing (matching) message. 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 return item in the ESEARCH response; however, it still MUST send the ESEARCH response.

COUNT
: Returns the number of messages that satisfy the SEARCH criteria. This return item MUST always be included in the ESEARCH response.

Example:

```
S: * ESEARCH UID COUNT 17 ALL 4:18,21,28
```

Example:

```
S: * ESEARCH (TAG "a567") UID COUNT 17 ALL 4:18,21,28
```

Example:

```
S: * ESEARCH COUNT 18 ALL 1:17,21
```

7.3.5. 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 remembered by the client.

Example:

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

7.4. 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.

7.4.1. 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 remembered by the client.

Example:

```
S: * 23 EXISTS
```

7.5. 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.

7.5.1. 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 remembered by the client.

Example:

```
S: * 44 EXPUNGE
```

7.5.2. 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 are in parentheses. This response occurs as the result of a FETCH or STORE command, as well as by a unilateral server decision (e.g., flag updates).

The current data items are:

BINARY[<section-binary>]<<number>>
: An <nstring> or <literal8> expressing the content of the specified section after removing any encoding specified in the corresponding Content-Transfer-Encoding header field. If <number> is present, it refers to the offset within the DECODED section data.

  If the domain of the decoded data is "8bit" and the data does not contain the NUL octet, the server SHOULD return the data in a <string> instead of a <literal8>; this allows the client to determine if the "8bit" data contains the NUL octet without having to explicitly scan the data stream for NULs.

Messaging clients and servers have been notoriously lax in their adherence to the Internet CRLF convention for terminating lines of textual data (text/* media types) in Internet protocols. When sending data in a BINARY[...] FETCH data item, servers MUST ensure that textual line-oriented sections are always transmitted using the IMAP CRLF line termination syntax, regardless of the underlying storage representation of the data on the server.

If the server does not know how to decode the section's Content-Transfer-Encoding, it MUST fail the request and issue a "NO" response that contains the "UNKNOWN-CTE" response code.

BINARY.SIZE[<section-binary>]
The size of the section after removing any encoding specified in the corresponding Content-Transfer-Encoding header field. The value returned MUST match the size of the <nstring> or <literal8> that will be returned by the corresponding FETCH BINARY request.

If the server does not know how to decode the section's Content-Transfer-Encoding, it MUST fail the request and issue a "NO" response that contains the "UNKNOWN-CTE" response code.

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 [RFC5322] 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 the header data, except in the case of a message that 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]. Servers SHOULD decode parameter-value continuations and parameter-value character sets as described in [RFC2231], for example, if the message contains parameters "baz*0", "baz*1", and "baz*2", the server should decode them per [RFC2231], concatenate, and return the resulting value as a parameter "baz". Similarly, if the message contains parameters "foo*0*" and "foo*1*", the server should decode them per [RFC2231], convert to UTF-8, concatenate, and return the resulting value as a parameter "foo*".

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]. Servers SHOULD decode parameter-value continuations as described in [RFC2231].

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 header field value as defined in Section 7 of [MIME-IMB].

body description
: A string giving the Content-Description header field value as defined in Section 8 of [MIME-IMB].

body encoding
: A string giving the content transfer encoding as defined in Section 6 of [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 [RFC5322] 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: display name, [SMTP] at-domain-list (source route and obs-route ABNF production from [RFC5322]), mailbox name (local-part ABNF production from [RFC5322]), and hostname.

[RFC5322] group syntax is indicated by a special form of address structure in which the hostname field is NIL. If the mailbox name field is also NIL, this is an end-of-group marker (semicolon in RFC 822 syntax). If the mailbox name field is non-NIL, this is the start of a group marker, and the mailbox name field holds the group name phrase.

If the Date, Subject, In-Reply-To, and Message-ID header fields are absent in the [RFC5322] header, the corresponding member of the envelope is NIL; if these header fields 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 the empty string as identical.

Note: [RFC5322] requires that all messages have a valid Date header field. Therefore, for a well-formed message, the date member in the envelope cannot be NIL or the empty string. However, it can be NIL for a malformed or draft message.

Note: [RFC5322] requires that the In-Reply-To and Message-ID header fields, if present, have non-empty content. Therefore, for a well-formed message, the in-reply-to and message-id members in the envelope cannot be the empty string. However, they can still be the empty string for a malformed message.

If the From, To, Cc, and Bcc header fields are absent in the [RFC5322] header, or are present but empty, the corresponding member of the envelope is NIL.

If the Sender or Reply-To header fields are absent in the [RFC5322] 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 how to do this).

Note: [RFC5322] requires that all messages have a valid From header field. Therefore, for a well-formed message, the from, sender, and reply-to members in the envelope cannot be NIL. However, they can be NIL for a malformed or draft message.

FLAGS
: A parenthesized list of flags that are set for this message.

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

RFC822.SIZE
: A number expressing the size of a message, as described in Section 2.3.4.

UID
: A number expressing the unique identifier of the message.

If the server chooses to send unsolicited FETCH responses, they MUST include UID FETCH item. Note that this is a new requirement when compared to [RFC3501].

Example:

```
S: * 23 FETCH (FLAGS (\Seen) RFC822.SIZE 44827 UID 447)
```

7.6. 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 on a non-TLS port. A long line in this sample is

broken for editorial clarity.

```
S:   * OK [CAPABILITY STARTTLS AUTH=SCRAM-SHA-256 LOGINDISABLED
      IMAP4rev2] IMAP4rev2 Service Ready
C:   a000 starttls
S:   a000 OK Proceed with TLS negotiation
 <TLS negotiation>
C:   A001 AUTHENTICATE SCRAM-SHA-256
      biwsbj11c2VyLHI9ck9wck5HZndFYmVSV2diTkVrcU8=
S:   + cj1yT3ByTkdmd0ViZVJXZ2JORWtxTyVodllEcFdVYTJSYVRDQWZ1eEZJbGopaE
     5sRiRrMCxzPVcyMlphSjBTTlk3c29Fc1VFamI2Z1E9PSxpPTQwOTY=
C:   Yz1iaXdzLHI9ck9wck5HZndFYmVSV2diTkVrcU8laHZZRHBXVWEyUmFUQ0FmdXhG
     SWxqKWhObEYkazAscD1kSHpiWmFwV0lrNGpVaE4rVXRlOXl0YWc5empmTUhnc3Ft
     bWl6N0FuZFZRPQ==
S:   + dj02cnJpVFJCaTIzV3BSUi93dHVwK21NaFVaVW4vZEI1bkxUSlJzamw5NUc0
     PQ==
C:
S:   A001 OK SCRAM-SHA-256 authentication successful
C:   babc ENABLE IMAP4rev2
S:   * ENABLED IMAP4rev2
S:   babc OK Some capabilities enabled
C:   a002 select inbox
S:   * 18 EXISTS
S:   * FLAGS (\Answered \Flagged \Deleted \Seen \Draft)
S:   * OK [UIDVALIDITY 3857529045] UIDs valid
S:   * LIST () "/" INBOX ("OLDNAME" ("inbox"))
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.ed>")
      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 that 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. Unless otherwise noted, all alphabetic characters are case insensitive. The use of uppercase or lowercase 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 linear whitespace (LWSP).

3. The ASCII NUL character, %x00, MUST NOT be used anywhere, with the exception of the OCTET production.

```
SP              = <Defined in RFC 5234>
CTL             = <Defined in RFC 5234>
CRLF            = <Defined in RFC 5234>
ALPHA           = <Defined in RFC 5234>
DIGIT           = <Defined in RFC 5234>
DQUOTE          = <Defined in RFC 5234>
OCTET           = <Defined in RFC 5234>

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

addr-adl        = nstring
                    ; Holds route from [RFC5322] obs-route if
                    ; non-NIL

addr-host       = nstring
                    ; NIL indicates [RFC5322] group syntax.
                    ; Otherwise, holds [RFC5322] domain name

addr-mailbox    = nstring
                    ; NIL indicates end of [RFC5322] group; if
                    ; non-NIL and addr-host is NIL, holds
                    ; [RFC5322] group name.
                    ; Otherwise, holds [RFC5322] local-part
                    ; after removing [RFC5322] quoting

addr-name       = nstring
                    ; If non-NIL, holds phrase from [RFC5322]
                    ; mailbox after removing [RFC5322] 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
                    ; Authentication mechanism name, as defined by
                    ; [SASL], Section 7.1

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 / number64 /
                   "(" 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
                  ; Content-Transfer-Encoding header field value.
                  ; Defaults to "7BIT" (as per RFC 2045)
                  ; if not present in the body part.

body-fld-id     = nstring

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

body-fld-loc    = nstring

body-fld-lines  = number64

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-mpart = 1*body SP media-subtype
                  [SP body-ext-mpart]
                    ; 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 SHOULD be
                    ; registered with IANA using the
                    ; RFC Required policy, i.e., in
                    ; a Standards Track, an Experimental,
                    ; or an Informational RFC.

capability-data = "CAPABILITY" *(SP capability) SP "IMAP4rev2"
                  *(SP capability)
                    ; See Section 6.1.1 for information about
                    ; required security-related capabilities.
                    ; Servers that offer RFC 1730 compatibility MUST
                    ; list "IMAP4" as the first capability.
                    ; Servers that offer RFC 3501 compatibility MUST
                    ; list "IMAP4rev1" as one of the capabilities.

CHAR            = <defined in [ABNF]>

CHAR8           = %x01-ff
                    ; any OCTET except NUL, %x00

charset         = atom / quoted

childinfo-extended-item =  "CHILDINFO" SP "("
            list-select-base-opt-quoted
            *(SP list-select-base-opt-quoted) ")"
            ; Extended data item (mbox-list-extended-item)
            ; returned when the RECURSIVEMATCH
            ; selection option is specified.
            ; Note 1: the CHILDINFO extended data item tag can be
            ; returned with or without surrounding quotes, as per
            ; mbox-list-extended-item-tag production.
            ; Note 2: The selection options are always returned
            ; quoted, unlike their specification in
            ; the extended LIST command.

child-mbox-flag =  "\HasChildren" / "\HasNoChildren"
            ; attributes for the CHILDREN return option, at most
            ; one possible per LIST response

command         = tag SP (command-any / command-auth /
                  command-nonauth / command-select) CRLF
                    ; Modal based on state

command-any     = "CAPABILITY" / "LOGOUT" / "NOOP"
                    ; Valid in all states

command-auth    = append / create / delete / enable / examine /
                  list / 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  = "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-month      = "Jan" / "Feb" / "Mar" / "Apr" / "May" / "Jun" /
                  "Jul" / "Aug" / "Sep" / "Oct" / "Nov" / "Dec"

date-text       = date-day "-" date-month "-" date-year

date-year       = 4DIGIT

date-time       = DQUOTE date-day-fixed "-" date-month "-" date-year
                  SP time SP zone DQUOTE

delete          = "DELETE" SP mailbox
                    ; Use of INBOX gives a NO error

digit-nz        = %x31-39
                    ; 1-9

eitem-standard-tag =  atom
            ; a tag for LIST extended data item defined in a Standard
            ; Track or Experimental RFC.

eitem-vendor-tag =  vendor-token "-" atom
            ; a vendor-specific tag for LIST extended data item

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

env-date        = nstring

env-from        = "(" 1*address ")" / nil

env-in-reply-to = nstring

env-message-id  = nstring

env-reply-to    = "(" 1*address ")" / nil

env-sender      = "(" 1*address ")" / nil

env-subject     = nstring
```

```
env-to          = "(" 1*address ")" / nil

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.SIZE" /
                  "BODY" ["STRUCTURE"] / "UID" /
                  "BODY" section [partial] /
                  "BODY.PEEK" section [partial] /
                  "BINARY" [".PEEK"] section-binary [partial] /
                  "BINARY.SIZE" section-binary

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
                    ; a future Standard or Standards Track
                    ; revisions of this specification.
                    ; "\Recent" was defined in RFC 3501
                    ; and is now deprecated.

flag-fetch      = flag / obsolete-flag-recent

flag-keyword    = "$MDNSent" / "$Forwarded" / "$Junk" /
                  "$NotJunk" / "$Phishing" / 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 4 of [SASL]

list            = "LIST" [SP list-select-opts] SP
                  mailbox SP mbox-or-pat
                  [SP list-return-opts]


list-mailbox    = 1*list-char / string
```

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

list-return-opt   =  return-option
                     ; Note that return-option is the ABNF
                     ; non-terminal used by RFC 5258

list-return-opts =  "RETURN" SP
            "(" [list-return-opt *(SP list-return-opt)] ")"
            ; list return options, e.g., CHILDREN

list-select-base-opt =  "SUBSCRIBED" / option-extension
            ; options that can be used by themselves

list-select-base-opt-quoted =  DQUOTE list-select-base-opt DQUOTE

list-select-independent-opt =  "REMOTE" / option-extension
            ; options that do not syntactically interact with
            ; other options

list-select-mod-opt =  "RECURSIVEMATCH" / option-extension
            ; options that require a list-select-base-opt
            ; to also be present

list-select-opt =  list-select-base-opt / list-select-independent-opt
                   / list-select-mod-opt

list-select-opts =  "(" [
                   (*(list-select-opt SP) list-select-base-opt
                   *(SP list-select-opt))
                  / (list-select-independent-opt
                   *(SP list-select-independent-opt))
                     ] ")"
            ; Any number of options may be in any order.
            ; If a list-select-mod-opt appears, then a
            ; list-select-base-opt must also appear.
            ; This allows these:
            ; ()
            ; (REMOTE)
            ; (SUBSCRIBED)
            ; (SUBSCRIBED REMOTE)
            ; (SUBSCRIBED RECURSIVEMATCH)
            ; (SUBSCRIBED REMOTE RECURSIVEMATCH)
            ; But does NOT allow these:
            ; (RECURSIVEMATCH)
            ; (REMOTE RECURSIVEMATCH)

list-wildcards  = "%" / "*"

literal         = "{" number64 ["+"] "}" CRLF *CHAR8
                    ; <number64> represents the number of CHAR8s.
                    ; A non-synchronizing literal is distinguished
                    ; from a synchronizing literal by the presence of
                    ; "+" before the closing "}".
                    ; Non-synchronizing literals are not allowed when
                    ; sent from server to the client.

literal8        =  "~{" number64 "}" CRLF *OCTET
                    ; <number64> represents the number of OCTETs
                    ; in the response string.

login           = "LOGIN" SP userid SP password
```

```
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 that
                    ; consists of the case-insensitive sequence
                    ; "I" "N" "B" "O" "X" is considered
                    ; to be an 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 /
                   esearch-response /
                   "STATUS" SP mailbox SP "(" [status-att-list] ")" /
                   number SP "EXISTS" / namespace-response /
                   obsolete-search-response /
                   obsolete-recent-response
                    ; obsolete-search-response and
                    ; obsolete-recent-response can only be returned
                    ; by servers that support both IMAPrev1
                    ; and IMAPrev2.

mailbox-list    = "(" [mbx-list-flags] ")" SP
                   (DQUOTE QUOTED-CHAR DQUOTE / nil) SP mailbox
                   [SP mbox-list-extended]
            ; This is the list information pointed to by the ABNF
            ; item "mailbox-data", which is defined above

mbox-list-extended =  "(" [mbox-list-extended-item
                      *(SP mbox-list-extended-item)] ")"

mbox-list-extended-item = mbox-list-extended-item-tag SP
                          tagged-ext-val

mbox-list-extended-item-tag = astring
               ; The content MUST conform to either
               ; "eitem-vendor-tag" or "eitem-standard-tag"
               ; ABNF productions.

mbox-or-pat =  list-mailbox / patterns

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" / child-mbox-flag /
                  "\Subscribed" / "\Remote" / flag-extension
               ; Other flags; multiple from this list are
               ; possible per LIST response, but each flag
               ; can only appear once per LIST response

mbx-list-sflag  = "\NonExistent" / "\Noselect" / "\Marked" /
                  "\Unmarked"
               ; Selectability flags; only one per LIST response

media-basic     = ((DQUOTE ("APPLICATION" / "AUDIO" / "IMAGE" /
                  "FONT" / "MESSAGE" / "MODEL" / "VIDEO" ) DQUOTE)
                  / string)
                  SP media-subtype
                    ; FONT defined in [RFC8081].
                    ; MODEL defined in [RFC2077].
                    ; Other top-level media types
                    ; are defined in [MIME-IMT].
```

```
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.SIZE" SP number64 /
                  "BODY" ["STRUCTURE"] SP body /
                  "BODY" section ["<" number ">"] SP nstring /
                  "BINARY" section-binary SP (nstring / literal8) /
                  "BINARY.SIZE" section-binary SP number /
                  "UID" SP uniqueid
                    ; MUST NOT change for a message

name-component  = 1*UTF8-CHAR
                    ; MUST NOT contain ".", "/", "%", or "*"

namespace         = nil / "(" 1*namespace-descr ")"

namespace-command = "NAMESPACE"

namespace-descr   = "(" string SP
                       (DQUOTE QUOTED-CHAR DQUOTE / nil)
                        [namespace-response-extensions] ")"

namespace-response-extensions = *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)

nz-number       = digit-nz *DIGIT
                    ; Non-zero unsigned 32-bit integer
                    ; (0 < n < 4,294,967,296)

nz-number64     = digit-nz *DIGIT
                    ; Unsigned 63-bit integer
                    ; (0 < n <= 9,223,372,036,854,775,807)

obsolete-flag-recent = "\Recent"

obsolete-recent-response = number SP "RECENT"

obsolete-search-response = "SEARCH" *(SP nz-number)

oldname-extended-item =  "OLDNAME" SP "(" mailbox ")"
                    ; Extended data item (mbox-list-extended-item)
                    ; returned in a LIST response when a mailbox is
                    ; renamed or deleted. Also returned when
                    ; the server canonicalized the provided mailbox
                    ; name.
                    ; Note 1: the OLDNAME tag can be returned
                    ; with or without surrounding quotes, as per
                    ; mbox-list-extended-item-tag production.

option-extension = (option-standard-tag / option-vendor-tag)
                   [SP option-value]

option-standard-tag =  atom
               ; an option defined in a Standards Track or
               ; Experimental RFC

option-val-comp =  astring /
                   option-val-comp *(SP option-val-comp) /
                   "(" option-val-comp ")"

option-value =  "(" option-val-comp ")"

option-vendor-tag =  vendor-token "-" atom
               ; a vendor-specific option, non-standard

partial-range    = number64 ["." nz-number64]
                    ; Copied from RFC 5092 (IMAP URL)
                    ; and updated to support 64-bit sizes.

partial         = "<" number64 "." nz-number64 ">"
                    ; Partial FETCH request. 0-based offset of
                    ; the first octet, followed by the number of
                    ; octets in the fragment.

password        = astring

patterns        = "(" list-mailbox ")"
                  ; [RFC5258] supports multiple patterns,
                  ; but this document only requires one
                  ; to be supported.
                  ; If the server is also implementing
                  ; [RFC5258], the "patterns" syntax from
                  ; that document must be followed.

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   = "]"

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" / "NOTSAVED" / "HASCHILDREN" /
                  "CLOSED" /
                  "UNKNOWN-CTE" /
                  atom [SP 1*<any TEXT-CHAR except "]">]

return-option   = "SUBSCRIBED" / "CHILDREN" / status-option /
                   option-extension

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 /
                  "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 number64 / "NOT" SP search-key /
                  "OR" SP search-key SP search-key /
                  "SENTBEFORE" SP date / "SENTON" SP date /
                  "SENTSINCE" SP date / "SMALLER" SP number64 /
                  "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.
                     ; Note that "$" marker is not allowed
                     ; after the ALL return data item.

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

search-return-opt  = "MIN" / "MAX" / "ALL" / "COUNT" /
                      "SAVE" /
                      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-binary  = "[" [section-part] "]"

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.
```

```
sequence-set    =/ seq-last-command
                   ; Allow for "result of the last command"
                   ; indicator.

seq-last-command   = "$"

status          = "STATUS" SP mailbox SP
                  "(" status-att *(SP status-att) ")"

status-att      = "MESSAGES" / "UIDNEXT" / "UIDVALIDITY" /
                  "UNSEEN" / "DELETED" / "SIZE"

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

status-option = "STATUS" SP "(" status-att *(SP status-att) ")"
                   ; This ABNF production complies with
                   ; <option-extension> syntax.

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 "+">

tag-string      = astring
                  ; <tag> represented as <astring>

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".
                      ; A 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 / UTF8-2 / UTF8-3 / UTF8-4)
                    ; Non-ASCII text can only be returned
                    ; after ENABLE IMAP4rev2 command

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 regardless of order.
                    ; Example: 2:4 and 4:2 are equivalent.

uniqueid        = nz-number
                    ; Strictly ascending

unsubscribe     = "UNSUBSCRIBE" SP mailbox

userid          = astring

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

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>

vendor-token    = "vendor." name-component
                    ; Definition copied from RFC 2244.
                    ; MUST be registered with IANA

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: [RFC3501], [RFC2060], [RFC1730], unpublished IMAP2bis.TXT document, [IMAP2], and [RFC1064].

11. Security Considerations

IMAP4rev2 protocol transactions, including electronic mail data, are sent in the clear over the network, exposing them to possible eavesdropping and manipulation unless protection is negotiated. This can be accomplished by use of the Implicit TLS port, the STARTTLS command, negotiated confidentiality protection in the AUTHENTICATE command, or some other protection mechanism.

11.1. TLS-Related Security Considerations

This section applies to use of both the STARTTLS command and the Implicit TLS port.

IMAP client and server implementations MUST comply with relevant TLS recommendations from [RFC8314]. If recommendations/requirements in this document conflict with recommendations from [RFC8314], for example in regards to TLS ciphersuites, recommendations from this document take precedence.

Clients and servers MUST implement TLS 1.2 [TLS-1.2] or newer. Use of TLS 1.3 [TLS-1.3] is RECOMMENDED. TLS 1.2 may be used only in cases where the other party has not yet implemented TLS 1.3. Additionally, when using TLS 1.2, IMAP implementations MUST implement the TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 cipher suite. This is important as it ensures that any two compliant implementations can be configured to interoperate. Other TLS cipher suites recommended in RFC 7525 [RFC7525] are RECOMMENDED: TLS_DHE_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].

The list of mandatory-to-implement TLS 1.3 cipher suites is described in Section 9.1 of [TLS-1.3].

During the TLS negotiation [TLS-1.3] [TLS-1.2], 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 on-path attackers attempting to masquerade as the server. This procedure is described in [RFC7817].

Both the client and server MUST check the result of the STARTTLS command and subsequent TLS [TLS-1.3] [TLS-1.2] negotiation to see whether acceptable authentication and/or privacy was achieved.

11.2. STARTTLS Command versus Use of Implicit TLS Port

For maximum backward compatibility, the client MUST implement both TLS negotiation on an Implicit TLS port and TLS negotiation using the STARTTLS command on a cleartext port.

The server MUST implement TLS negotiation on an Implicit TLS port. The server SHOULD also implement IMAP on a cleartext port. If the server listens on a cleartext port, it MUST allow the STARTTLS command on it.

Some site/firewall maintainers insist on TLS site-wide and prefer not to rely on a configuration option in each higher-level protocol. For this reason, IMAP4rev2 clients SHOULD try both ports 993 and 143 (and both IPv4 and IPv6) concurrently by default, unless overridden by either user configuration or DNS SRV records [RFC6186]. A good algorithm for implementing such concurrent connect is described in [RFC8305].

11.3. Client Handling of Unsolicited Responses Not Suitable for the Current Connection State

Cleartext mail transmission (whether caused by firewall configuration errors that result in TLS stripping or weak security policies in email clients that choose not to negotiate TLS in the first place) can enable injection of responses that can confuse or even cause crashes in email clients. The following measures are recommended to minimize damage from them.

- See Section 7.1.4 for special security considerations related to the PREAUTH response.
- Many server responses and response codes are only meaningful in authenticated or even selected state. However, nothing prevents a server (or an on-path attacker) from sending such invalid responses in cleartext before STARTTLS/AUTHENTICATE commands are issued. Before authentication, clients SHOULD ignore any responses other than CAPABILITY and server status responses (Section 7.1), as well as any response codes other than CAPABILITY. (In particular, some email clients are known to incorrectly process LIST responses received before authentication, or FETCH responses when no mailbox is selected.) Clients SHOULD ignore the ALERT response code until after TLS (whether using STARTTLS or TLS negotiation on an Implicit TLS port) or a SASL security layer with confidentiality protection has been successfully negotiated. Unless explicitly allowed by an IMAP extension, when not in selected state, clients MUST ignore responses / response codes related to message and mailbox status such as FLAGS, EXIST, EXPUNGE, and FETCH.

11.4. 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.

11.5. LIST Command and Other Users' Namespace

In response to a LIST command containing an argument of the Other Users' Namespace prefix, a server MUST NOT list users that have not granted list access to their personal mailboxes to the currently authenticated user. Providing such a list could compromise security by potentially disclosing confidential information of who is located on the server or providing a starting point for a list of user accounts to attack.

11.6. Use of MD5

The BODYSTRUCTURE FETCH data item can contain the MD5 digest of the message body in the "body MD5" field (body-fld-md5 ABNF production). While MD5 is no longer considered a secure cryptographic hash [RFC6151], this field is used solely to expose the value of the Content-MD5 header field (if present in the original message), which is just a message integrity check and is not used for cryptographic purposes. Also note that other mechanisms that provide message integrity checks were defined since RFC 1864 [MD5] was published and are now more commonly used than Content-MD5. Two such mechanisms are the DKIM-Signature header field [RFC6376] and S/MIME signing [RFC8550] [RFC8551].

11.7. Other Security Considerations

A server error message for an AUTHENTICATE command that 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 TLS negotiated on an Implicit TLS port
   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.

A server SHOULD report any authentication failure and analyze such authentication failure attempts with regard to a password brute-force attack as well as a password spraying attack [NCSC]. Accounts with passwords that match well-known passwords from spraying attacks MUST be blocked, and users associated with such accounts must be requested to change their passwords. Only a password with significant strength SHOULD be accepted.

Additional security considerations are discussed in the sections that define the AUTHENTICATE and LOGIN commands (see Sections 6.2.2 and 6.2.3, respectively).

12. IANA Considerations

IANA has updated the "Service Names and Transport Protocol Port Numbers" registry as follows:

1. Registration for TCP port 143 and the corresponding "imap" service name have been updated to point to this document and [RFC3501].
2. Registration for TCP port 993 and the corresponding "imaps" service name have been updated to point to this document, [RFC8314], and [RFC3501].
3. UDP ports 143 and 993 have both been marked as "Reserved" in the registry.

Additional IANA actions are specified in the subsections that follow.

12.1. Updates to IMAP Capabilities Registry

IMAP4 capabilities are registered by publishing a Standards Track or IESG-approved Informational or Experimental RFC. The registry is currently located at: <https://www.iana.org/assignments/imap4-capabilities>

As this specification revises the AUTH= prefix, STARTTLS, and LOGINDISABLED extensions, IANA has updated registry entries for these 3 extensions to point to this document and [RFC3501].

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: <https://www.iana.org/assignments/gssapi-service-names>

IANA has updated the "imap" service name previously registered in [RFC3501] to point to both this document and [RFC3501].

12.3. LIST Selection Options, LIST Return Options, and LIST Extended Data Items

[RFC5258] specifies IANA registration procedures for LIST selection options, LIST return options, and LIST extended data items. This document doesn't change these registration procedures. In particular, LIST selection options (Section 6.3.9.1) and LIST return options (Section 6.3.9.2) are registered using the procedure specified in Section 9 of [RFC5258] (and using the registration template from Section 9.3 of [RFC5258]). LIST extended data items are registered using the registration template from Section 9.6 of [RFC5258]).

IANA has added a reference to RFC 9051 for the "OLDNAME" LIST-EXTENDED extended data item entry. This is in addition to the existing reference to [RFC5465].

12.4. IMAP Mailbox Name Attributes and IMAP Response Codes

IANA has updated the "IMAP Mailbox Name Attributes" registry to point to this document in addition to [RFC3501].

IANA has updated the "IMAP Response Codes" registry to point to this document in addition to [RFC3501].

13. References

13.1. Normative References

[ABNF] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <https://www.rfc-editor.org/info/rfc5234>.

[BCP178] Saint-Andre, P., Crocker, D., and M. Nottingham, "Deprecating the "X-" Prefix and Similar Constructs in Application Protocols", BCP 178, RFC 6648, June 2012.

<https://www.rfc-editor.org/info/bcp178>

[CHARSET] Freed, N. and J. Postel, "IANA Charset Registration Procedures", BCP 19, RFC 2978, DOI 10.17487/RFC2978, October 2000, <https://www.rfc-editor.org/info/rfc2978>.

[DISPOSITION] Troost, R., Dorner, S., and K. Moore, Ed., "Communicating Presentation Information in Internet Messages: The Content-Disposition Header Field", RFC 2183, DOI 10.17487/RFC2183, August 1997, <https://www.rfc-editor.org/info/rfc2183>.

[I18N-HDRS] Yang, A., Steele, S., and N. Freed, "Internationalized Email Headers", RFC 6532, DOI 10.17487/RFC6532, February 2012, <https://www.rfc-editor.org/info/rfc6532>.

[IMAP-IMPLEMENTATION] Leiba, B., "IMAP4 Implementation Recommendations", RFC 2683, DOI 10.17487/RFC2683, September 1999, <https://www.rfc-editor.org/info/rfc2683>.

[IMAP-MULTIACCESS] Gahrns, M., "IMAP4 Multi-Accessed Mailbox Practice", RFC 2180, DOI 10.17487/RFC2180, July 1997, <https://www.rfc-editor.org/info/rfc2180>.

[LANGUAGE-TAGS] Alvestrand, H., "Content Language Headers", RFC 3282, DOI 10.17487/RFC3282, May 2002, <https://www.rfc-editor.org/info/rfc3282>.

[LOCATION] Palme, J., Hopmann, A., and N. Shelness, "MIME Encapsulation of Aggregate Documents, such as HTML (MHTML)", RFC 2557, DOI 10.17487/RFC2557, March 1999, <https://www.rfc-editor.org/info/rfc2557>.

[MD5] Myers, J. and M. Rose, "The Content-MD5 Header Field", RFC 1864, DOI 10.17487/RFC1864, October 1995, <https://www.rfc-editor.org/info/rfc1864>.

[MIME-HDRS] Moore, K., "MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text", RFC 2047, DOI 10.17487/RFC2047, November 1996, <https://www.rfc-editor.org/info/rfc2047>.

[MIME-IMB] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, DOI 10.17487/RFC2045, November 1996, <https://www.rfc-editor.org/info/rfc2045>.

[MIME-IMT] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, DOI 10.17487/RFC2046, November 1996, <https://www.rfc-editor.org/info/rfc2046>.

[MULTIAPPEND] Crispin, M., "Internet Message Access Protocol (IMAP) - MULTIAPPEND Extension", RFC 3502, DOI 10.17487/RFC3502, March 2003, <https://www.rfc-editor.org/info/rfc3502>.

[NET-UNICODE] Klensin, J. and M. Padlipsky, "Unicode Format for Network Interchange", RFC 5198, DOI 10.17487/RFC5198, March 2008, <https://www.rfc-editor.org/info/rfc5198>.

[PLAIN] Zeilenga, K., Ed., "The PLAIN Simple Authentication and Security Layer (SASL) Mechanism", RFC 4616, DOI 10.17487/RFC4616, August 2006, <https://www.rfc-editor.org/info/rfc4616>.

[RFC2077] Nelson, S., Parks, C., and Mitra., "The Model Primary Content Type for Multipurpose Internet Mail Extensions", RFC 2077, DOI 10.17487/RFC2077, January 1997, <https://www.rfc-editor.org/info/rfc2077>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC2231] Freed, N. and K. Moore, "MIME Parameter Value and Encoded Word Extensions: Character Sets, Languages, and Continuations", RFC 2231, DOI 10.17487/RFC2231, November 1997, <https://www.rfc-editor.org/info/rfc2231>.

[RFC3503] Melnikov, A., "Message Disposition Notification (MDN) profile for Internet Message Access Protocol (IMAP)", RFC 3503, DOI 10.17487/RFC3503, March 2003, <https://www.rfc-editor.org/info/rfc3503>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/info/rfc4648>.

[RFC4752] Melnikov, A., Ed., "The Kerberos V5 ("GSSAPI") Simple Authentication and Security Layer (SASL) Mechanism", RFC 4752, DOI 10.17487/RFC4752, November 2006, <https://www.rfc-editor.org/info/rfc4752>.

[RFC5258] Leiba, B. and A. Melnikov, "Internet Message Access Protocol version 4 - LIST Command Extensions", RFC 5258, DOI 10.17487/RFC5258, June 2008, <https://www.rfc-editor.org/info/rfc5258>.

[RFC5322] Resnick, P., Ed., "Internet Message Format", RFC 5322, DOI 10.17487/RFC5322, October 2008, <https://www.rfc-editor.org/info/rfc5322>.

[RFC5788] Melnikov, A. and D. Cridland, "IMAP4 Keyword Registry", RFC 5788, DOI 10.17487/RFC5788, March 2010, <https://www.rfc-editor.org/info/rfc5788>.

[RFC7525] Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 7525, DOI 10.17487/RFC7525, May 2015, <https://www.rfc-editor.org/info/rfc7525>.

[RFC7817] Melnikov, A., "Updated Transport Layer Security (TLS) Server Identity Check Procedure for Email-Related Protocols", RFC 7817, DOI 10.17487/RFC7817, March 2016, <https://www.rfc-editor.org/info/rfc7817>.

[RFC8081] Lilley, C., "The "font" Top-Level Media Type", RFC 8081, DOI 10.17487/RFC8081, February 2017, <https://www.rfc-editor.org/info/rfc8081>.

[RFC8098] Hansen, T., Ed. and A. Melnikov, Ed., "Message Disposition Notification", STD 85, RFC 8098, DOI 10.17487/RFC8098, February 2017, <https://www.rfc-editor.org/info/rfc8098>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8314] Moore, K. and C. Newman, "Cleartext Considered Obsolete: Use of Transport Layer Security (TLS) for Email Submission and Access", RFC 8314, DOI 10.17487/RFC8314, January 2018, <https://www.rfc-editor.org/info/rfc8314>.

[SASL] Melnikov, A., Ed. and K. Zeilenga, Ed., "Simple Authentication and Security Layer (SASL)", RFC 4422, DOI 10.17487/RFC4422, June 2006, <https://www.rfc-editor.org/info/rfc4422>.

[SCRAM-SHA-256] Hansen, T., "SCRAM-SHA-256 and SCRAM-SHA-256-PLUS Simple Authentication and Security Layer (SASL) Mechanisms", RFC 7677, DOI 10.17487/RFC7677, November 2015, <https://www.rfc-editor.org/info/rfc7677>.

[TLS-1.2] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <https://www.rfc-editor.org/info/rfc5246>.

[TLS-1.3] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[UTF-7] Goldsmith, D. and M. Davis, "UTF-7 A Mail-Safe Transformation Format of Unicode", RFC 2152, DOI 10.17487/RFC2152, May 1997, <https://www.rfc-editor.org/info/rfc2152>.

[UTF-8] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <https://www.rfc-editor.org/info/rfc3629>.

13.2. Informative References

13.2.1. Related Protocols

[ANONYMOUS] Zeilenga, K., "Anonymous Simple Authentication and Security Layer (SASL) Mechanism", RFC 4505, DOI 10.17487/RFC4505, June 2006, <https://www.rfc-editor.org/info/rfc4505>.

[CERT-555316] Carnegie Mellon University, "STARTTLS plaintext command injection vulnerability", Software Engineering Institute, CERT Coordination Center, Vulnerability Note VU#555316, September 2011, <https://www.kb.cert.org/vuls/id/555316>.

[CHARSET-REG] IANA, "Character Set Registrations", <https://www.iana.org/assignments/charset-reg/>.

[IMAP-DISC] Melnikov, A., Ed., "Synchronization Operations for Disconnected IMAP4 Clients", RFC 4549, DOI 10.17487/RFC4549, June 2006, <https://www.rfc-editor.org/info/rfc4549>.

[IMAP-I18N] Newman, C., Gulbrandsen, A., and A. Melnikov, "Internet Message Access Protocol Internationalization", RFC 5255, DOI 10.17487/RFC5255, June 2008, <https://www.rfc-editor.org/info/rfc5255>.

[IMAP-KEYWORDS-REG] IANA, "IMAP and JMAP Keywords", <https://www.iana.org/assignments/imap-jmap-keywords/>.

[IMAP-MAILBOX-NAME-ATTRS-REG] IANA, "IMAP Mailbox Name Attributes", <https://www.iana.org/assignments/imap-mailbox-name-attributes/>.

[IMAP-MODEL] Crispin, M., "Distributed Electronic Mail Models in IMAP4", RFC 1733, DOI 10.17487/RFC1733, December 1994, <https://www.rfc-editor.org/info/rfc1733>.

[IMAP-URL] Melnikov, A., Ed. and C. Newman, "IMAP URL Scheme", RFC 5092, DOI 10.17487/RFC5092, November 2007, <https://www.rfc-editor.org/info/rfc5092>.

[IMAP-UTF-8] Resnick, P., Ed., Newman, C., Ed., and S. Shen, Ed., "IMAP Support for UTF-8", RFC 6855, DOI 10.17487/RFC6855, March 2013, <https://www.rfc-editor.org/info/rfc6855>.

[NCSC] NCSC, "Spray you, spray me: defending against password spraying attacks", May 2018, <https://www.ncsc.gov.uk/blog-post/spray-you-spray-me-defending-against-password-spraying-attacks>.

[RFC2087] Myers, J., "IMAP4 QUOTA extension", RFC 2087, DOI 10.17487/RFC2087, January 1997, <https://www.rfc-editor.org/info/rfc2087>.

[RFC2177] Leiba, B., "IMAP4 IDLE command", RFC 2177, DOI 10.17487/RFC2177, June 1997, <https://www.rfc-editor.org/info/rfc2177>.

[RFC2193] Gahrns, M., "IMAP4 Mailbox Referrals", RFC 2193, DOI 10.17487/RFC2193, September 1997, <https://www.rfc-editor.org/info/rfc2193>.

[RFC2342] Gahrns, M. and C. Newman, "IMAP4 Namespace", RFC 2342, DOI 10.17487/RFC2342, May 1998, <https://www.rfc-editor.org/info/rfc2342>.

[RFC3348] Gahrns, M. and R. Cheng, "The Internet Message Action Protocol (IMAP4) Child Mailbox Extension", RFC 3348, DOI 10.17487/RFC3348, July 2002, <https://www.rfc-editor.org/info/rfc3348>.

[RFC3516] Nerenberg, L., "IMAP4 Binary Content Extension", RFC 3516, DOI 10.17487/RFC3516, April 2003, <https://www.rfc-editor.org/info/rfc3516>.

[RFC3691] Melnikov, A., "Internet Message Access Protocol (IMAP) UNSELECT command", RFC 3691, DOI 10.17487/RFC3691, February 2004, <https://www.rfc-editor.org/info/rfc3691>.

[RFC4314] Melnikov, A., "IMAP4 Access Control List (ACL) Extension", RFC 4314, DOI 10.17487/RFC4314, December 2005, <https://www.rfc-editor.org/info/rfc4314>.

[RFC4315] Crispin, M., "Internet Message Access Protocol (IMAP) - UIDPLUS extension", RFC 4315, DOI 10.17487/RFC4315, December 2005, <https://www.rfc-editor.org/info/rfc4315>.

[RFC4466] Melnikov, A. and C. Daboo, "Collected Extensions to IMAP4 ABNF", RFC 4466, DOI 10.17487/RFC4466, April 2006, <https://www.rfc-editor.org/info/rfc4466>.

[RFC4731] Melnikov, A. and D. Cridland, "IMAP4 Extension to SEARCH Command for Controlling What Kind of Information Is Returned", RFC 4731, DOI 10.17487/RFC4731, November 2006, <https://www.rfc-editor.org/info/rfc4731>.

[RFC4959] Siemborski, R. and A. Gulbrandsen, "IMAP Extension for Simple Authentication and Security Layer (SASL) Initial Client Response", RFC 4959, DOI 10.17487/RFC4959, September 2007, <https://www.rfc-editor.org/info/rfc4959>.

[RFC5161] Gulbrandsen, A., Ed. and A. Melnikov, Ed., "The IMAP ENABLE Extension", RFC 5161, DOI 10.17487/RFC5161, March 2008, <https://www.rfc-editor.org/info/rfc5161>.

[RFC5182] Melnikov, A., "IMAP Extension for Referencing the Last SEARCH Result", RFC 5182, DOI 10.17487/RFC5182, March 2008, <https://www.rfc-editor.org/info/rfc5182>.

[RFC5256] Crispin, M. and K. Murchison, "Internet Message Access Protocol - SORT and THREAD Extensions", RFC 5256, DOI 10.17487/RFC5256, June 2008, <https://www.rfc-editor.org/info/rfc5256>.

[RFC5465] Gulbrandsen, A., King, C., and A. Melnikov, "The IMAP NOTIFY Extension", RFC 5465, DOI 10.17487/RFC5465, February 2009, <https://www.rfc-editor.org/info/rfc5465>.

[RFC5530] Gulbrandsen, A., "IMAP Response Codes", RFC 5530, DOI 10.17487/RFC5530, May 2009, <https://www.rfc-editor.org/info/rfc5530>.

[RFC5819] Melnikov, A. and T. Sirainen, "IMAP4 Extension for Returning STATUS Information in Extended LIST", RFC 5819, DOI 10.17487/RFC5819, March 2010, <https://www.rfc-editor.org/info/rfc5819>.

[RFC6151] Turner, S. and L. Chen, "Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms", RFC 6151, DOI 10.17487/RFC6151, March 2011, <https://www.rfc-editor.org/info/rfc6151>.

[RFC6154] Leiba, B. and J. Nicolson, "IMAP LIST Extension for Special-Use Mailboxes", RFC 6154, DOI 10.17487/RFC6154, March 2011, <https://www.rfc-editor.org/info/rfc6154>.

[RFC6186] Daboo, C., "Use of SRV Records for Locating Email Submission/Access Services", RFC 6186, DOI 10.17487/RFC6186, March 2011, <https://www.rfc-editor.org/info/rfc6186>.

[RFC6376] Crocker, D., Ed., Hansen, T., Ed., and M. Kucherawy, Ed., "DomainKeys Identified Mail (DKIM) Signatures", STD 76, RFC 6376, DOI 10.17487/RFC6376, September 2011, <https://www.rfc-editor.org/info/rfc6376>.

[RFC6409] Gellens, R. and J. Klensin, "Message Submission for Mail", STD 72, RFC 6409, DOI 10.17487/RFC6409, November 2011, <https://www.rfc-editor.org/info/rfc6409>.

[RFC6851] Gulbrandsen, A. and N. Freed, Ed., "Internet Message Access Protocol (IMAP) - MOVE Extension", RFC 6851, DOI 10.17487/RFC6851, January 2013, <https://www.rfc-editor.org/info/rfc6851>.

[RFC7162] Melnikov, A. and D. Cridland, "IMAP Extensions: Quick Flag Changes Resynchronization (CONDSTORE) and Quick Mailbox Resynchronization (QRESYNC)", RFC 7162, DOI 10.17487/RFC7162, May 2014, <https://www.rfc-editor.org/info/rfc7162>.

[RFC7888] Melnikov, A., Ed., "IMAP4 Non-synchronizing Literals", RFC 7888, DOI 10.17487/RFC7888, May 2016, <https://www.rfc-editor.org/info/rfc7888>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[RFC8305] Schinazi, D. and T. Pauly, "Happy Eyeballs Version 2: Better Connectivity Using Concurrency", RFC 8305, DOI 10.17487/RFC8305, December 2017, <https://www.rfc-editor.org/info/rfc8305>.

[RFC8438] Bosch, S., "IMAP Extension for STATUS=SIZE", RFC 8438, DOI 10.17487/RFC8438, August 2018, <https://www.rfc-editor.org/info/rfc8438>.

[RFC8474] Gondwana, B., Ed., "IMAP Extension for Object Identifiers", RFC 8474, DOI 10.17487/RFC8474, September 2018, <https://www.rfc-editor.org/info/rfc8474>.

[RFC8550] Schaad, J., Ramsdell, B., and S. Turner, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 4.0 Certificate Handling", RFC 8550, DOI 10.17487/RFC8550, April 2019, <https://www.rfc-editor.org/info/rfc8550>.

[RFC8551] Schaad, J., Ramsdell, B., and S. Turner, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 4.0 Message Specification", RFC 8551, DOI 10.17487/RFC8551, April 2019, <https://www.rfc-editor.org/info/rfc8551>.

[SMTP] Klensin, J., "Simple Mail Transfer Protocol", RFC 5321, DOI 10.17487/RFC5321, October 2008, <https://www.rfc-editor.org/info/rfc5321>.

13.2.2. Historical Aspects of IMAP and Related Protocols

[IMAP-COMPAT] Crispin, M., "IMAP4 Compatibility with IMAP2bis", RFC 2061, DOI 10.17487/RFC2061, December 1996, <https://www.rfc-editor.org/info/rfc2061>.

[IMAP-HISTORICAL] Crispin, M., "IMAP4 Compatibility with IMAP2 and IMAP2bis", RFC 1732, DOI 10.17487/RFC1732, December 1994, <https://www.rfc-editor.org/info/rfc1732>.

[IMAP-OBSOLETE] Crispin, M., "Internet Message Access Protocol - Obsolete Syntax", RFC 2062, DOI 10.17487/RFC2062, December 1996, <https://www.rfc-editor.org/info/rfc2062>.

[IMAP-TLS] Newman, C., "Using TLS with IMAP, POP3 and ACAP", RFC 2595, DOI 10.17487/RFC2595, June 1999, <https://www.rfc-editor.org/info/rfc2595>.

[IMAP2] Crispin, M., "Interactive Mail Access Protocol: Version 2", RFC 1176, DOI 10.17487/RFC1176, August 1990, <https://www.rfc-editor.org/info/rfc1176>.

[IMAP2BIS] Crispin, M., "INTERACTIVE MAIL ACCESS PROTOCOL - VERSION 2bis", Work in Progress, Internet-Draft, draft-ietf-imap-imap2bis-02, 29 October 1993, <https://datatracker.ietf.org/doc/html/draft-ietf-imap-imap2bis-02>.

[RFC1064] Crispin, M., "Interactive Mail Access Protocol: Version 2", RFC 1064, DOI 10.17487/RFC1064, July 1988, <https://www.rfc-editor.org/info/rfc1064>.

[RFC1730] Crispin, M., "Internet Message Access Protocol - Version 4", RFC 1730, DOI 10.17487/RFC1730, December 1994, <https://www.rfc-editor.org/info/rfc1730>.

[RFC2060] Crispin, M., "Internet Message Access Protocol - Version 4rev1", RFC 2060, DOI 10.17487/RFC2060, December 1996, <https://www.rfc-editor.org/info/rfc2060>.

[RFC3501] Crispin, M., "INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1", RFC 3501, DOI 10.17487/RFC3501, March 2003, <https://www.rfc-editor.org/info/rfc3501>.

[RFC822] Crocker, D., "STANDARD FOR THE FORMAT OF ARPA INTERNET TEXT MESSAGES", STD 11, RFC 822, DOI 10.17487/RFC0822, August 1982, <https://www.rfc-editor.org/info/rfc822>.

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. (Such server implementation is likely to also want to advertise other IMAP4rev1 extensions that were folded into IMAP4rev2; see Appendix E.) 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.

When compared to IMAP4rev1, some request data items, corresponding response data items, and responses were removed in IMAP4rev2. See Appendix E for more details. With the exception of obsolete SEARCH and RECENT responses, servers advertising both IMAP4rev1 and IMAP4rev2 would never return such removed response data items/responses unless explicitly requested by an IMAPrev1 client.

Servers advertising both IMAP4rev1 and IMAP4rev2 MUST 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 Mailbox International Naming Convention described in Appendix A.1.

Also see Appendix D for special considerations for servers that support 63-bit body part / message sizes and want to advertise support for both IMAP4rev1 and IMAP4rev2.

A.1. Mailbox International Naming Convention for Compatibility with IMAP4rev1

Support for the Mailbox International Naming Convention described in this section is not required for IMAP4rev2-only clients and servers. It is only used for backward compatibility with IMAP4rev1 implementations.

By convention, international mailbox names in IMAP4rev1 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 2-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 of a US-ASCII character that 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 the 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 that 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 that export a mail store that does not follow the modified UTF-7 convention MUST convert any mailbox name that contains either non-ASCII characters or the "&" character to modified UTF-7.

> For example, here is a mailbox name that 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. Backward Compatibility with BINARY Extension

IMAP4rev2 incorporates a subset of functionality provided by the BINARY extension [RFC3516]; in particular, it includes additional FETCH items (BINARY, BINARY.PEEK, and BINARY.SIZE) but not extensions to the APPEND command. IMAP4rev2 implementations that support full [RFC3516] functionality need to also advertise the BINARY capability in the CAPABILITY response / response code.

Appendix C. Backward Compatibility with LIST-EXTENDED Extension

IMAP4rev2 incorporates most of the functionality provided by the LIST-EXTENDED extension [RFC5258]. In particular, the syntax for multiple mailbox patterns is not supported in IMAP4rev2, unless LIST-EXTENDED capability is also advertised in the CAPABILITY response / response code.

Appendix D. 63-Bit Body Part and Message Sizes

IMAP4rev2 increases allowed body part and message sizes that servers can support from 32 to 63 bits. Server implementations don't have to support 63-bit-long body parts/message sizes; however, client implementations have to expect them.

As IMAP4rev1 didn't support 63-bit-long body part / message sizes, there is an interoperability issue exposed by 63-bit-capable servers/mailboxes that are accessible by both IMAP4rev1 and IMAP4rev2 email clients. As IMAP4rev1 would be unable to retrieve the full content of messages bigger than 4 Gb, such servers either need to replace messages bigger that 4 Gb with messages under 4 Gb or hide them from IMAP4rev1 clients. This document doesn't prescribe any implementation strategy to address this issue.

Appendix E. Changes from RFC 3501 / IMAP4rev1

Below is the summary of changes since RFC 3501:

1. Support for 64-bit message and body part sizes.
2. Folded in IMAP NAMESPACE [RFC2342], UNSELECT [RFC3691], UIDPLUS [RFC4315], ESEARCH [RFC4731], SEARCHRES [RFC5182], ENABLE [RFC5161], IDLE [RFC2177], SASL-IR [RFC4959], LIST-EXTENDED [RFC5258], LIST-STATUS [RFC5819], MOVE [RFC6851], and LITERAL- extensions [RFC7888]. Also folded in IMAP ABNF extensions [RFC4466], response codes [RFC5530], the FETCH side of the BINARY extension [RFC3516], and the list of new mailbox attributes from SPECIAL-USE [RFC6154].
3. Added STATUS SIZE [RFC8438] and STATUS DELETED.
4. SEARCH command now requires to return the ESEARCH response (SEARCH response is now deprecated).
5. Clarified which SEARCH keys have to use substring match and which don't.
6. Clarified that the server should decode parameter value continuations as described in [RFC2231]. This requirement was hidden in [RFC2231] itself.
7. Clarified that the COPYUID response code is returned for both MOVE and UID MOVE.
8. Tightened requirements about COPY/MOVE commands not creating a target mailbox. Also required them to return the TRYCREATE response code, if the target mailbox doesn't exist and can be created.
9. Added the CLOSED response code from [RFC7162]. SELECT/EXAMINE when a mailbox is already selected now requires a CLOSED response code to be returned.
10. SELECT/EXAMINE are now required to return an untagged LIST response.
11. UNSEEN response code on SELECT/EXAMINE is now deprecated.
12. RECENT response on SELECT/EXAMINE, \Recent flag, RECENT STATUS, and SEARCH NEW items are now deprecated.
13. Clarified that the server doesn't need to send a new PERMANENTFLAGS response code when a new keyword was successfully added and the server advertised * earlier for the same mailbox.
14. For future extensibility, extended ABNF for tagged-ext-simple to allow for bare number64.

15. Added SHOULD level requirement on IMAP servers to support $MDNSent, $Forwarded, $Junk, $NonJunk, and $Phishing keywords.
16. 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.
17. Removed the CHECK command. Clients should use NOOP instead.
18. RFC822, RFC822.HEADER, and RFC822.TEXT FETCH data items were deprecated. Clients should use the corresponding BODY[] variants instead.
19. LSUB command was deprecated. Clients should use LIST (SUBSCRIBED) instead.
20. IDLE command can now return updates not related to the currently selected mailbox state.
21. All unsolicited FETCH updates are required to include UID.
22. Clarified that client implementations MUST ignore response codes that they do not recognize. (Changed from a SHOULD to a MUST.)
23. resp-text ABNF non-terminal was updated to allow for empty text.
24. After ENABLE, IMAP4rev2 human-readable response text can include non-ASCII encoded in UTF-8.
25. Updated to use modern TLS-related recommendations as per [RFC7525], [RFC7817], and [RFC8314].
26. Added warnings about use of ALERT response codes and PREAUTH response.
27. Replaced DIGEST-MD5 SASL mechanism with SCRAM-SHA-256. DIGEST-MD5 was deprecated.
28. Clarified that any command received from the client resets server autologout timer.
29. Revised IANA registration procedure for IMAP extensions and removed "X" convention in accordance with [BCP178].
30. Loosened requirements on servers when closing connections to be more aligned with existing practices.

Appendix F. Other Recommended IMAP Extensions

Support for the following extensions is recommended for all IMAP clients and servers. While they significantly reduce bandwidth and/or number of round trips used by IMAP in certain situations, the EXTRA WG decided that requiring them as a part of IMAP4rev2 would push the bar to implement too high for new implementations. Also note that the absence of any IMAP extension from this list doesn't make it somehow deficient or not recommended for use with IMAP4rev2.

1. Quick Mailbox Resynchronization (QRESYNC) and CONDSTORE extensions [RFC7162]. They make discovering changes to IMAP mailboxes more efficient, at the expense of storing a bit more state.
2. OBJECTID extension [RFC8474] helps with preserving the IMAP client cache when messages are moved/copied or mailboxes are renamed.

Acknowledgements

Earlier draft versions of this document were edited by Mark Crispin. Sadly, he is no longer available to help with this work. Editors of this revision 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. Thank you to Murray Kucherawy, Timo Sirainen, Bron Gondwana, Stephan Bosch, Robert Sparks, Arnt Gulbrandsen, Benjamin Kaduk, Daniel Migaul, Roman Danyliw, and Éric Vyncke for extensive feedback.

This document incorporates text from [RFC2342] (by Mike Gahrns and Chris Newman), [RFC3516] (by Lyndon Nerenberg), [RFC4315] (by Mark Crispin), [RFC4466] (by Cyrus Daboo), [RFC4731] (by Dave Cridland), [RFC4959] (by Rob Siemborski and Arnt Gulbrandsen), [RFC5161] (by Arnt Gulbrandsen), [RFC5465] (by Arnt Gulbrandsen and Curtis King), [RFC5530] (by Arnt Gulbrandsen), [RFC5819] (by Timo Sirainen), [RFC6154] (by Jamie Nicolson), [RFC6851] (by Arnt Gulbrandsen and Ned Freed), and [RFC8438] (by Stephan Bosch), so work done by authors/editors of these documents is appreciated. Note that editors of this document were redacted from the above list.

The CHILDREN return option was originally proposed by Mike Gahrns and Raymond Cheng in [RFC3348]. Most of the information in Section 6.3.9.5 is taken directly from their original specification [RFC3348].

Thank you to Damian Poddebniak, Fabian Ising, Hanno Boeck, and Sebastian Schinzel for pointing out that the ENABLE command should be a member of "command-auth" and not "command-any" ABNF production, as well as pointing out security issues associated with ALERT, PREAUTH, and other responses received before authentication.

Index

Authors' Addresses

Alexey Melnikov (EDITOR)
Isode Ltd
14 Castle Mews
Hampton, Middlesex
TW12 2NP
United Kingdom
Email: Alexey.Melnikov@isode.com

Barry Leiba (EDITOR)
Futurewei Technologies
Email: barryleiba@computer.org
URI: http://internetmessagingtechnology.org/