Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 20, 2016

A. Melnikov, Ed.
Isode Ltd

October 18, 2015

DMAP MESSAGE ACCESS PROTOCOL
draft-melnikov-dmap-00.txt

Abstract

The DMAP Message Access Protocol, Version 1 allows a client to access and manipulate electronic mail messages on a server, without revealing too much information about messages being accessed to the server. DMAP permits manipulation of mailboxes (remote message folders) in a way that is functionally equivalent to local folders. DMAP also provides the capability for an offline client to resynchronize with the server and for message submission. DMAP supports discovery of keys (signets) belonging to other users the client can communicate to. Syncronization and publication of keys (private key, might include certificates) and signets (public part, certificate).

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

Note: This document is a very early draft and omission of specific syntax is intentional. It is intended to stimulate discussions about specific protocol syntax and general design.

This Internet-Draft will expire on April 20, 2016.

Copyright Notice

Table of Contents

1. How to Read This Document

1.1. Organization of This Document

1.2. Conventions Used in This Document

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

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

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

The word "can" (not "may") is used to refer to a possible circumstance or situation, as opposed to an optional facility of the protocol.

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

"Connection" refers to the entire sequence of client/server interaction from the initial establishment of the network connection until its termination.

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

Characters are 8-bit UTF-8 unless otherwise specified.

1.3. Special Notes to Implementors/To Do

[[CREF1: This section needs to be rewritten or removed before publication.]]

This specification is experimental. While early implementations are encouraged, there are lots of open issues and possibility for drastical change to the protocol. Implementors are enouraged to contact authors of this specification before starting implementing this specification.

The following changes are planned (this is not an exhaustive list):

- Include LITERAL+ syntax.
- Incorporate IDLE
- Merge LIST and STATUS into a single command
- Fix the mailbox (folder) hierarchy separator character to be "."
- Reorganize sections to group command by purpose.

2. Design Goals

This protocols strives to satisfy the following goals (note that some of the goals are in conflict, so certain compromises were made):

Any DMAP connection is always protected by TLS. [[CREF2: Add text about server TLS identity verification.]]

Most of the message content and associated metadata is encrypted with a key only known to DMAP clients, so DMAP servers get very limited access to user data.

Open Issue: should the list of mailbox names be accessible to the server (unencrypted)? What about their attributes (e.g. mailbox roles, such as Sent or Drafts)? It might still be possible for a server (or MITM attacker) to figure out mailbox roles based on usage pattern.

Open Issue: should it be possible for the server to search for messages which contain a particular message flag (in that case such flags should be stored unencrypted)?

Open Issue: is it useful to support searching for all messages from or to a particular domain? (Compare this with searching for a particular sender/recipient, which is useful)

The protocol allows for efficient bandwidth usage for mobile clients. For example, it should be possible to download a message body structure, which is much smaller than the message itself and allows the client to decide which body parts is worth downloading. Also, it should be possible to download binary body parts (without any Content Transfer Encoding).

Submission of new messages through DMAP is supported in order to make client configuration easier.

The best bits of the IMAP protocol are reused, making implementations slightly easier.

3. Protocol Overview

3.1. Link Level

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

3.2. Commands and Responses

An DMAP connection consists of the establishment of a client/server network connection, mandatory TLS authentication exchange . Once TLS exchange completes successfully the connection proceeds with 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.

[[CREF3: Might need revising if this changes.]]> 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 DMAP client or server is either reading a line, or is reading a sequence of octets with a known count followed by a line.

3.2.1. Client Protocol Sender and Server Protocol Receiver

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

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

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

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

The protocol receiver of an DMAP 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.

3.2.2. Server Protocol Sender and Client Protocol Receiver

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

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

The server completion result response indicates the success or failure of the operation. It is tagged with the same tag as the client command which began the operation. Thus, if more than one command is in progress, the tag in a server completion response identifies the command to which the response applies. There are three possible server completion responses: OK (indicating success), NO (indicating failure), or BAD (indicating a protocol error such as unrecognized command or command syntax error).

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

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

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

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

3.3. 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.

3.3.1. Message Numbers

TBD: decide if message sequence numbers are needed

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

3.3.1.1. Unique Identifier (UID) Message Attribute

An unsigned 32-bit value assigned to each message, which when used with the unique identifier validity value (see below) forms a 64-bit value that MUST NOT refer to any other message in the mailbox or any subsequent mailbox with the same name forever. Unique identifiers are assigned in a strictly ascending fashion in the mailbox; as each message is added to the mailbox it is assigned a higher UID than the message(s) which were added previously. Unlike message sequence numbers, unique identifiers are not necessarily contiguous.

The unique identifier of a message MUST NOT change during the session, and SHOULD NOT change between sessions. Any change of unique identifiers between sessions MUST be detectable using the UIDVALIDITY mechanism discussed below. Persistent unique identifiers are required for a client to resynchronize its state from a previous session with the server (e.g., disconnected or offline access clients).

Associated with every mailbox are two 32-bit unsigned values which aid in unique identifier handling: the next unique identifier value (UIDNEXT) and the unique identifier validity value (UIDVALIDITY).

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

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

The unique identifier validity value is sent in a UIDVALIDITY response code in an OK untagged response at mailbox selection time.

Unique identifiers MUST persist at all times. The following considerations about unique identifiers apply:

1. Unique identifiers MUST be strictly ascending in the mailbox at all times. If the physical message store is re-ordered (or messages are modified) by a non-DMAP agent, this requires that the unique identifiers in the mailbox be regenerated, since the former unique identifiers are no longer strictly ascending as a result of the re-ordering.

2. If the mailbox is deleted and a new mailbox with the same name is created at a later date (or another mailbox is renamed to have the name of a previously deleted or renamed mailbox), the server must either keep track of unique identifiers from the previous instance of the mailbox, or it must assign a new UIDVALIDITY value to the new instance of the mailbox. A good UIDVALIDITY value to use in this case is a 32-bit representation of the creation date/time of the mailbox. It is alright to use a constant such as 1, but only if it guaranteed that unique identifiers will never be reused, even in the case of a mailbox being deleted (or renamed) and a new mailbox by the same name created at some future time.

3. The combination of mailbox name, UIDVALIDITY, and UID must refer to a single immutable message on that server forever. In particular, the internal date, message size, 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 (e.g., FLAGS).

3.3.1.2. Message Sequence Number Message Attribute

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

Message sequence numbers can be reassigned during the session. For example, when a message is permanently removed (expunged) from the mailbox, the message sequence number for all subsequent messages is decremented. The number of messages in the mailbox is also decremented. Similarly, a new message can be assigned a message

sequence number that was once held by some other message prior to an expunge.

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

3.3.2. Flags Message Attribute

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

A system flag is a flag name that is pre-defined in this specification. All system flags begin with "\". Certain system flags (\Deleted and \Seen) have special semantics described elsewhere. The currently-defined system flags are: [[CREF4: Alexey: some of these should be moved to the encrypted per message metadata block.]]

\Seen Message has been read.

\Answered Message has been answered.

\Forwarded Message has been forwarded.

\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).

\Submitted and \SubmitPending The \SubmitPending flag designates the message as awaiting to be submitted. This keyword allows storing messages waiting to be submitted in the same mailbox where messages that were already submitted and/or are being edited are stored. A mail client sets this flag when it decides that the message needs to be sent out. When a client (it might be a different client from the one that decided that the message is pending submission) starts sending the message, it atomically adds the \Submitted flag. Once submission is successful, the \SubmitPending flag is atomically cleared. The two flags allow

messages being actively submitted (messages that have both \Submitted and \SubmitPending flags set) to be distinguished from messages awaiting to be submitted, or from messages already submitted. They also allow all messages that were supposed to be submitted to be found, if the client submitting them crashes or quits before submitting them. [[CREF5: Update SUBMIT to also talk about these flags.]]

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). Keywords registered in documents that extend this specification SHOULD start with "$".

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

3.3.3. Internal Date Message Attribute

The internal date and time of the message on the server. This is not the date and time in the [RFC-5322] header, but rather a date and time which reflects when the message was received. In the case of messages delivered via DMTP , this SHOULD be the date and time of final delivery of the message. In the case of messages delivered by the DMAP COPY command, this SHOULD be the internal date and time of the source message. In the case of messages delivered by the DMAP APPEND command, this SHOULD be the date and time as specified in the APPEND command description. All other cases are implementation defined.

3.3.4. Size Message Attribute

The number of octets in the message.

3.3.5. Body Structure Message Attribute

A parsed representation of the body structure information of the message.

3.3.6. Modification Sequence (MODSEQ) Message Attribute

A 63 bits positive integer that gets incremented every time there is a change to one of mutable attributes of a message. (Currently such mutable attributes only include message flags).

3.4. Message Texts

In addition to being able to fetch the full text of a message, DMAP permits the fetching of portions of the full message. Specifically, it is possible to fetch any message chunk.

4. State and Flow Diagram

Once the connection between client and server is established, an DMAP 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.

4.1. 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.

4.2. Authenticated State

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

4.3. Selected State

TBD: Decide if Selected state can be eliminated entirely

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

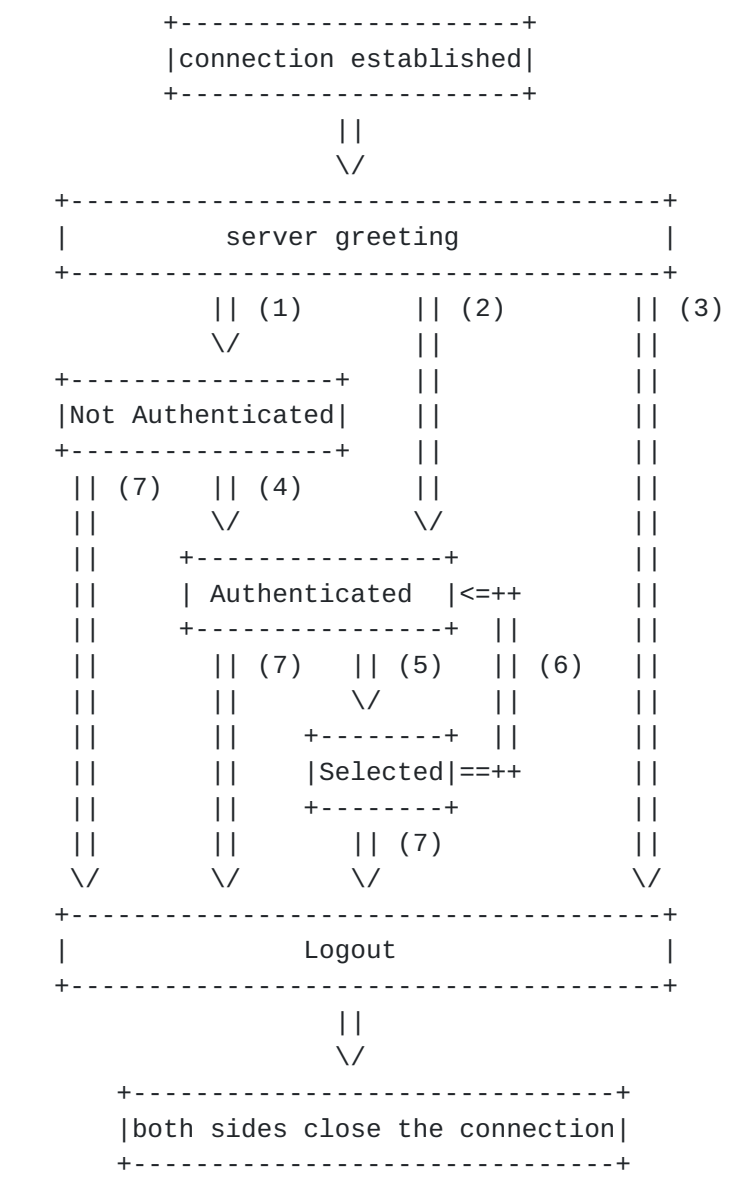
4.4. Logout State

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

If the client requests the logout state, the server MUST send an untagged BYE response and a tagged OK response to the LOGOUT command

before the server closes the connection; and the client MUST read the tagged OK response to the LOGOUT command before the client closes the connection.

A server MUST NOT unilaterally close the connection without sending an untagged BYE response that contains the reason for having done so. A client SHOULD NOT unilaterally close the connection, and instead SHOULD issue a LOGOUT command. If the server detects that the client has unilaterally closed the connection, the server MAY omit the untagged BYE response and simply close its connection.



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

5. Data Formats

DMAP uses textual commands and responses. Data in DMAP 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.

5.1. Atom

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

5.2. Number

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

5.3. String

A string is in one of two forms: either literal or quoted string. The literal form is the general form of string. The quoted string form is an alternative that avoids the overhead of processing a literal at the cost of limitations of characters which may be used.

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

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

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

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

5.3.1. 8-bit and Binary Strings

...Include direct support for BINARY-like literals.

5.4. 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.

5.5. NIL

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

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

6. Operational Considerations

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

6.1. Mailbox Naming

Mailbox names are encoded in UTF-8.

The case-insensitive mailbox name INBOX is a special name reserved to mean "the primary mailbox for this user on this server". 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; others preserve case of a newly-created name but otherwise are case-insensitive; and yet others coerce names to a particular case. Client implementations MUST interact with any of these.

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

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

2. CTL and other non-graphic characters are difficult to represent in a user interface and are thus disallowed.

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. The "/" character is reserved to delimit levels of hierarchy.

6.1.1. Mailbox Hierarchy Naming

If it is desired to export hierarchical mailbox names, mailbox names MUST be left-to-right hierarchical using a single "/" character to separate levels of hierarchy.

6.2. Mailbox Size and Message Status Updates

At any time, a server can send data that the client did not request. Sometimes, such behavior is REQUIRED. For example, agents other than the server MAY add messages to the mailbox (e.g., new message delivery), change the flags of the messages in the mailbox (e.g., simultaneous access to the same mailbox by multiple agents), or even remove messages from the mailbox. A server MUST send mailbox size updates automatically if a mailbox size change is observed during the processing of a command. A server SHOULD send message flag updates automatically, without requiring the client to request such updates explicitly.

Special rules exist for server notification of a client about the removal of messages to prevent synchronization errors; see the description of the EXPUNGE response for more detail. In particular, it is NOT permitted to send an EXISTS response that would reduce the number of messages in the mailbox; only the EXPUNGE response can do this.

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

6.3. 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.

6.4. Autologout Timer

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

6.5. Multiple Commands in Progress (Command Pipelining)

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

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

7. Client Commands

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

Command arguments, identified by "Arguments:" in the command descriptions below, are described by function, not by syntax. The precise syntax of command arguments is described in the Formal Syntax (Section 10).

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

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

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

7.1. Client Commands - Any State

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

7.1.1. CAPABILITY Command

Arguments: none

Responses: REQUIRED untagged response: CAPABILITY

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

The CAPABILITY command requests a listing of capabilities that the server supports. The server MUST send a single untagged CAPABILITY response with "DMAP=..." (see below) as one of the listed capabilities before the (tagged) OK response.

The DMAP= capability describes in which mode DMAP operates. It MUST be followed by one of "TRUSTFUL", "CAUTIOUS" or "PARANOID". [[CREF6: Add more details about different modes and how they change the behaviour]]

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

Other capability names refer to extensions, revisions, or amendments to this specification. See the documentation of the CAPABILITY response for additional information. No capabilities, beyond the base DMAP set defined in this specification, are enabled without explicit client action to invoke the capability.

See the section entitled "Client Commands - Experimental/Expansion" for information about the form of site or implementation-specific capabilities.

7.1.2. NOOP Command

Arguments: none

Responses: no specific responses for this command (but see below)

Result: OK - noop completed
BAD - arguments invalid

The NOOP command always succeeds. It does nothing.

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

7.1.3. 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.

7.2. Client Commands - Not Authenticated State

In the not authenticated state, the AUTHENTICATE 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.

[[CREF7: Is this still a useful feature in DMAP context?]] 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]. 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: AUTHENTICATE. See the Security Considerations section for important information about these commands.

7.2.1. AUTHENTICATE Command

Arguments: authentication mechanism name

Responses: continuation data can be requested

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

The AUTHENTICATE command indicates 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 of [SASL].

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

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 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.

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 STACIE mechanism described in [[Add ref]]. 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 such as PLAIN. Server sites SHOULD NOT use any configuration which permits a plaintext password mechanism. Client and server implementations SHOULD implement additional [SASL] mechanisms that do not use plaintext passwords, such as STACIE, SCRAM [[CREF8: Add references]], and/or the GSSAPI mechanism described in [SASL].

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. In other words, the client MAY request authentication types in decreasing order of preference.

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.

7.3. Client Commands - Authenticated State

In the authenticated state, commands that manipulate mailboxes as atomic entities are permitted. Of these commands, the SELECT and EXAMINE commands will select a mailbox for access and enter the selected state. [[CREF9: Should we also add "one shot resync" commands a la QRESYNC/JMAP?]]

In addition to the universal commands (CAPABILITY, NOOP, and LOGOUT), the following commands are valid in the authenticated state: SELECT, EXAMINE, CREATE, DELETE, RENAME, SUBSCRIBE, UNSUBSCRIBE, LIST, STATUS, and APPEND, as well as key ring and signet ring management commands described in subsequent sections.

7.3.1. SELECT Command

Arguments: mailbox name

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

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

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

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

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

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

OK [UIDNEXT <n>] The next unique identifier value. Refer to Section 3.3.1.1 for more information.

OK [UIDVALIDITY <n>] The unique identifier validity value. Refer to Section 3.3.1.1 for more information. If this is missing, the server does not support unique identifiers.

Only one mailbox can be selected at a time in a connection; simultaneous access to multiple mailboxes requires multiple connections. The SELECT command automatically deselects any currently selected mailbox before attempting the new selection. [[CREF10: Add CLOSED response to delimit old and new mailbox state.]] Consequently, if a mailbox is selected and a SELECT command that fails is attempted, no mailbox is selected.

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

If the client is not permitted to modify the mailbox but is permitted read access, the mailbox is selected as read-only, and the server MUST prefix the text of the tagged OK response to SELECT with the "[READ-ONLY]" response code. Read-only access through SELECT differs from the EXAMINE command in that certain read-only mailboxes MAY permit the change of permanent state on a per-user (as opposed to global) basis.

7.3.2. EXAMINE Command

Arguments: mailbox name

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

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

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

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

7.3.3. CREATE Command

Arguments: mailbox name

Responses: no specific responses for this command

Result: OK - create completed

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

The CREATE command creates a mailbox with the given name. [[CREF11: Encrypted mailbox name?]] An OK response is returned only if a new mailbox with that name has been created. It is an error to attempt to create INBOX or a mailbox with a name that refers to an extant mailbox. Any error in creation will return a tagged NO response.

If the mailbox name is suffixed with the server's hierarchy separator character (as returned from the server by a LIST command), this is a declaration that the client intends to create mailbox names under this name in the hierarchy. Server implementations that do not require this declaration MUST ignore the declaration. In any case, the name created is without the trailing hierarchy delimiter.

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

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

7.3.4. DELETE Command

Arguments: mailbox name

Responses: no specific responses for this command

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

The DELETE command permanently removes the mailbox with the given name. A tagged OK response is returned only if the mailbox has been deleted. It is an error to attempt to delete INBOX or a mailbox name that does not exist.

The DELETE command MUST NOT remove inferior hierarchical names. For example, if a mailbox "foo" has an inferior "foo.bar" (assuming "." is the hierarchy delimiter character), removing "foo" MUST NOT remove "foo.bar". It is an error to attempt to delete a name that has

inferior hierarchical names and also has the \Noselect mailbox name attribute (see the description of the LIST response for more details).

It is permitted to delete a name that has inferior hierarchical names and does not have the \Noselect mailbox name attribute. If the server implementation does not permit deleting the name while inferior hierarchical names exists the \Noselect mailbox name attribute is set for that name. In any case, all messages in that mailbox are removed by the DELETE command.

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

7.3.5. RENAME Command

Arguments: existing mailbox name
new mailbox name

Responses: no specific responses for this command

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

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

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

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

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

[[CREF12: If we always support returning roles for mailboxes, there is no need for this special behaviour.]] Renaming INBOX is permitted, and has special behavior. It moves all messages in INBOX to a new mailbox with the given name, leaving INBOX empty. If the server implementation supports inferior hierarchical names of INBOX, these are unaffected by a rename of INBOX.

7.3.6. SUBSCRIBE Command

Arguments: mailbox

Responses: no specific responses for this command

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

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

A server MAY validate the mailbox argument to SUBSCRIBE to verify that it exists. However, it MUST NOT unilaterally remove an existing mailbox name from the subscription list even if a mailbox by that name no longer exists. [[CREF13: Do we need this restriction?]]

> 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.

7.3.7. UNSUBSCRIBE Command

Arguments: mailbox name

Responses: no specific responses for this command

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

The UNSUBSCRIBE command removes the specified mailbox name from the server's set of "active" or "subscribed" mailboxes as returned by the LIST (SUBSCRIBED) command. This command returns a tagged OK response only if the unsubscription is successful. [[CREF14: We can allow UNSUBSCRIBE to succeed for a mailbox which is not subscribed.]]

7.3.8. LIST Command

Arguments: OPTIONAL selection options
mailbox name with possible wildcards
OPTIONAL return options

Responses: untagged responses: LIST

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

[[CREF15: Update to include options, like "SUBSCRIBED".]] The LIST command returns a subset of names from the complete set of all names available to the client. Zero or more untagged LIST replies are returned, containing the name attributes, hierarchy delimiter, name, and optional mailbox status information; see the description of the LIST reply for more detail.

The LIST command SHOULD return its data quickly, without undue delay. If each name requires 1 second of processing, then a list of 1200 names would take 20 minutes!

The returned mailbox names MUST match the supplied mailbox name pattern.

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

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

[[CREF16: Is this needed with roles?]] 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 mailbox name arguments with wildcards as described above. The criteria for omitting INBOX is whether SELECT INBOX will return failure; it is not relevant whether the user's real INBOX resides on this or some other server.

7.3.9. 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 DMAP connection and doing an EXAMINE command on a mailbox to query that mailbox's status without deselecting the current mailbox in the first DMAP connection.

Unlike the LIST command, the STATUS command is not guaranteed to be fast in its response. Under certain circumstances, it can be quite slow. In some implementations, the server is obliged to open the mailbox read-only internally to obtain certain status information. Also unlike the LIST command, the STATUS command does not accept wildcards. [[CREF17: Remove this restriction?]]

> Note: The STATUS command is intended to access the status of mailboxes other than the currently selected mailbox. Because the STATUS command can cause the mailbox to be opened internally, and because this information is available by other means on the selected mailbox, the STATUS command SHOULD NOT be used on the currently selected mailbox.
>
> The STATUS command MUST NOT be used as a "check for new messages in the selected mailbox" operation (refer to sections 7, Section 8.3.1 for more information about the proper method for new message checking).

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 3.3.1.1 for more information.

UIDVALIDITY The unique identifier validity value of the mailbox. Refer to Section 3.3.1.1 for more information.

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

7.3.10. APPEND Command

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

Responses: no specific responses for this command

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

The APPEND command appends the literal argument as a new message to the end of the specified destination mailbox. This argument SHOULD be in the format of a DMIME message. Binary data is permitted in the message.

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

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

If the append is unsuccessful for any reason, the mailbox MUST be restored to its state before the APPEND attempt; no partial appending is permitted.

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

If the mailbox is currently selected, the normal new message actions MUST occur. Specifically, the server MUST notify the client immediately via an untagged EXISTS response.

Note: The APPEND command is not used for message submission.

7.4. Client Commands - Authenticated State - Key Ring Management

This section describes user's key ring management commands: GETKEY, ADDKEY, DELETEKEY, LISTKEYS.

7.4.1. GETKEY Command

Arguments: key ID
key part indicator (PRIVATE, PUBLIC or BOTH)

Responses: REQUIRED untagged responses: KEY

Result: OK - getkey completed
NO - getkey failure: the key with key id was not found
BAD - arguments invalid

The GETKEY command requests the server to return private key, public key or both.

7.4.2. ADDKEY Command

Arguments: key ID
Signet Signing Request (might contain public key or both public and private key)

Responses: none

Result: OK - addkey completed
NO - addkey failure: the key already exists or storage failure
BAD - arguments invalid

The ADDKEY command requests the server to add the specified public key or both public key and the corresponding private key to the key ring. [[CREF18: Whether both or just public key are uploaded depends on the DMAP mode.]]

It is an error to add a key with the key id which already exists. [[CREF19: Add more details about the response code to be returned in such case.]] DELETEKEY should be used first to delete such key.

7.4.3. DELETEKEY Command

Arguments: key ID

Responses: none

Result: OK - deletekey completed
NO - deletekey failure: the key is not found
BAD - arguments invalid

The DELETEKEY command requests the server to delete the corresponding public (and the associated private, if exists) key using the key identifier.

DELETEKEY MUST fail with a tagged NO response if there are any messages on the server associated with the key id or if the expiry of the key hasn't been reached.

7.4.4. LISTKEYS Command

Arguments: None

Responses: KEY untagged response for each key

Result: OK - listkeys completed
NO - listkeys failure: no status for that name
BAD - arguments invalid

The LISTKEYS command requests the server to return key ids of all keys in the key ring. Each key id is returned using the KEY untagged response which doesn't include anything other than the key id.

7.5. Client Commands - Authenticated State - Signet Ring Management

This section describes signet ring management commands: GETSIGNET, ADDSIGNET, DELETESIGNET, LISTSIGNETS.

[[CREF20: TBD. Email address is used instead of key id to get/add/delete/list. LISTSIGNETS should allow for wildcards.]]

7.6. 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, CREATE, DELETE, RENAME, SUBSCRIBE, UNSUBSCRIBE, LIST, STATUS and APPEND), the

following commands are valid in the selected state: CLOSE, EXPUNGE, SEARCH , FETCH, STORE, COPY, SUBMIT and UID.

7.6.1. 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 returns to the authenticated state from the selected state. No untagged EXPUNGE responses are sent.

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

Even if a mailbox is selected, a SELECT, EXAMINE, or LOGOUT command MAY be issued without previously issuing a CLOSE command. The SELECT, EXAMINE, and LOGOUT commands implicitly close the currently selected mailbox without doing an expunge. However, when many messages are deleted, a CLOSE-LOGOUT or CLOSE-SELECT sequence is considerably faster than an EXPUNGE-LOGOUT or EXPUNGE-SELECT because no untagged EXPUNGE responses (which the client would probably ignore) are sent.

7.6.2. 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

[[CREF21: Switch to returning UIDs in EXPUNGE response?]] 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. Note that if any messages with the \Recent flag set are expunged, an untagged RECENT response is sent after the untagged EXPUNGE(s) to update the client's count of RECENT messages.

7.6.3. SEARCH Command

Arguments: searching criteria (one or more)

Responses: REQUIRED untagged response: SEARCH

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

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

When multiple keys are specified, the result is the intersection (AND function) of all the messages that match those keys. For example, the criteria DELETED SINCE 1-Feb-2015 refers to all deleted messages that were placed in the mailbox since February 1, 2015. 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).

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

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

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

ALL All messages in the mailbox; the default initial key for ANDing.

BEFORE <date> Messages whose internal date (disregarding time and timezone) is earlier than the specified date.

DELETED Messages with the \Deleted flag set.

LARGER <n> Messages with an [RFC-5322] 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.

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

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

UID <sequence set> Messages with unique identifiers corresponding to the specified unique identifier set. Sequence set ranges are permitted.

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

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

7.6.4. 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.

[[CREF22: Make sure the following statement is true once ABNF is done.]] Most data items, identified in the formal syntax under the msg-att-static rule, are static and MUST NOT change for any particular message. Other data items, identified in the formal syntax under the msg-att-dynamic rule, MAY change, either as a result of a STORE command or due to external events.

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

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

FAST Macro equivalent to: (FLAGS INTERNALDATE SIZE)

FULL Macro equivalent to: (FLAGS INTERNALDATE SIZE BODYSTRUCTURE)

The currently defined data items that can be fetched are:

BODY[<section>]<<partial>>

The content of a particular chunk or of the whole message. The section specification has the following syntax: <chunk-type>.<chunk-number>. For example "0.1" - the first Tracing chunk. "67.2" - the second Display-Content chunk. [[CREF23: This needs more thought. In particular, is nesting of body parts allowed?]]

The section specification can be the empty string, in which case the content of the whole message is returned.

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[].

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

[[CREF24: Decide if this is going to be binary or human readable (e.g. a list).]]

The BODYSTRUCTURE FETCH item contains basic information about all chunks of the message which enables clients to download only specific chunks of the message without downloading the whole message. This is computed by the server by extracting available chunk types and associated data from the message. This can provide performance improvements when dealing with big attachments.

FLAGS The flags that are set for this message.

META Encrypted block of data that represents mutable state associated with the message, such as encrypted flags. [[CREF25: TBD]]

MODSEQ The message modification sequence. It is a 63 bit unsigned integer (expressed as a decimal), which changes every time message's flags or encrypted metadata block changes. [[CREF26: TBD]]

INTERNALDATE The internal date of the message.

SIZE The size of the message in octets.

UID The unique identifier for the message.

7.6.5. 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 (other than \Recent) 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
```

7.6.6. 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, and the Recent flag SHOULD be set, in the copy.

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

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

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

7.6.7. SUBMIT Command

Arguments: message number of the message to send
OPTIONAL list of delivery options (e.g. "delay submission until", etc.)

Responses: FETCH response with updated message flags

Result: OK - Message submitted for delivery
NO - Submission error: can't move to the Sent mailbox, error
in flags or date/time or message text
BAD - arguments invalid

The SUBMIT command submits the specified message using DMTP protocol. The server ensures that the current user key is used with the message being submitted, so the server MUST reject messages which don't contain a valid signature using the current signing key. The server MUST also ensure that the origin chunk provides the correct author information (which may be distinct from the "From" header embedded in the meta chunk). [[CREF27: Add DMIME reference here.]] The server also sets/clears some message flags in the process in order to prevent other DMAP clients from submitting the same message at the same time. This is described in more details below.

[[CREF28: One of the delivery options can specify whether to move the submitted message to the Sent mailbox. TBD.]]

Clients MUST NOT submit a message which is either not marked with the \SubmitPending keyword , or which is marked with the \Submitted keyword. Servers MUST reject such a command with a tagged NO bearing the SUBMISSIONRACE response code.

In the course of submission, servers MUST atomically add the \Submitted flag to the message. A transient state where the message is temporarily marked with both \Submitted and \SubmitPending flags MAY be hidden from any IMAP session or it MAY be visible in some or all of them.

If the command succeeded, the message MUST be marked with the \Submitted flag, the \SubmitPending flag MUST be cleared and a FETCH response containing the message UID and its new flags MUST be sent.

If the command fails, the server MUST clear both the \Submitted or \SubmitPending flags.

Clients MUST be prepared to handle failing submission at any time. Servers MAY reject message submission for any reason.

[[CREF29: Delivery options: TBD.]] The server MUST process all specified delivery options and their detailed options. The server MUST respond with a tagged BAD if the client used unrecognized or unannounced option, or if a recognized option is used in an invalid way. If the server cannot honor a recognized and announced option, it MUST respond with a tagged NO with the POLICYDENIED response code and the message MUST NOT be submitted, nor its flags changed.

7.6.8. UID Command

Arguments: command name
command arguments

Responses: untagged responses: FETCH, SEARCH

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

The UID command has two forms. In the first form, it takes as its arguments a COPY, FETCH, or STORE command with arguments appropriate for the associated command. However, the numbers in the 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 or UID STORE to return an OK without performing any operations.

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

7.7. Client Commands - Experimental/Expansion

7.7.1. X<atom> Command

Arguments: implementation defined

Responses: implementation defined

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

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

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

Example: C: a441 CAPABILITY
S: * CAPABILITY DMAP 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

8. Server Responses

Server responses are in three forms: status responses, server data, and command continuation request. The information contained in a server response, identified by "Contents:" in the response descriptions below, is described by function, not by syntax. The precise syntax of server responses is described in the Formal Syntax section.

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

Status responses can be tagged or untagged. Tagged status responses indicate the completion result (OK, NO, or BAD status) of a client command, and have a tag matching the command.

Some status responses, and all server data, are untagged. An untagged response is indicated by the token "*" instead of a tag. Untagged status responses indicate server greeting, or server status that does not indicate the completion of a command (for example, an impending system shutdown alert). For historical reasons, untagged server data responses are also called "unsolicited data", although

strictly speaking, only unilateral server data is truly "unsolicited".

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

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

An example of unilateral untagged server data occurs when the DMAP connection is in the selected state. In the selected state, the server checks the mailbox for new messages as part of command execution. Normally, this is part of the execution of every command; hence, a NOOP command suffices to check for new messages. If new messages are found, the server sends untagged EXISTS and RECENT responses reflecting the new size of the mailbox. Server implementations that offer multiple simultaneous access to the same mailbox SHOULD also send appropriate unilateral untagged FETCH and EXPUNGE responses if another agent changes the state of any message flags or expunges any messages.

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

8.1. Server Responses - Status Responses

Status responses are OK, NO, BAD, PREAUTH and BYE. OK, NO, and BAD can be tagged or untagged. PREAUTH and BYE are always untagged.

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

The currently defined response codes are:

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

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

PERMANENTFLAGS Followed by a parenthesized list of flags, indicates which of the known flags the client can change permanently. Any flags that are in the FLAGS untagged response, but not the PERMANENTFLAGS list, can not be set permanently. If the client attempts to STORE a flag that is not in the PERMANENTFLAGS list, the server will either ignore the change or store the state change for the remainder of the current session only. The PERMANENTFLAGS list can also include the special flag *, which indicates that it is possible to create new keywords by attempting to store those flags in the mailbox.

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

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

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

UIDNEXT Followed by a decimal number, indicates the next unique identifier value. Refer to Section 3.3.1.1 for more information.

UIDVALIDITY Followed by a decimal number, indicates the unique identifier validity value. Refer to Section 3.3.1.1 for more information.

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

8.1.1. 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 an AUTHENTICATE command is needed.

```
Example:    S: * OK DMAP server ready
            [...]
            C: A001 SELECT mailbox
            [...]
            S: * OK [ALERT] System shutdown in 10 minutes
            S: A001 OK SELECT Completed
```

8.1.2. 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.

8.1.3. 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.

8.1.4. 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 AUTHENTICATE command is needed.

Example: S: * PREAUTH DMAP server logged in as Smith

8.1.5. 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

8.2. Server Responses - Server and Mailbox Status

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

8.2.1. CAPABILITY Response

Contents: capability listing

The CAPABILITY response occurs as a result of a CAPABILITY command. The capability listing contains a space-separated listing of capability names that the server supports.

The capability listing MUST include the atom "DMAP=...", which describes in which mode DMAP operates. It MUST be followed by one of "TRUSTFUL", "CAUTIOUS" or "PARANOID".

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

Other capability names indicate that the server supports an extension, revision, or amendment to the DMAP protocol. Server responses MUST conform to this document until the client issues a command that uses the associated capability.

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

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

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

Example: S: * CAPABILITY DMAP AUTH=GSSAPI XPIG-LATIN

8.2.2. STATUS Response

Contents: encrypted mailbox name
status parenthesized list

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

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

8.2.3. FLAGS Response

Contents: flag parenthesized list

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

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

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

8.3. Server Responses - Mailbox Size

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

8.3.1. EXISTS Response

Contents: none

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

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

Example: S: * 23 EXISTS

8.4. Server Responses - Message Status

[[CREF30: Get rid of message numbers altogether?]] 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.

8.4.1. FETCH Response

Contents: message data

The FETCH response returns data about a message to the client. The data are pairs of data item names and their values in parentheses.

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

The current data items are:

BODY[<section>]<<origin octet>>

A string expressing the contents of the specified chunk or of the whole message. The section string has the following syntax: <chunk-type>.<number>. For example "0.1" - the first Tracing chunk. "67.2" - the second Display-Content chunk. [[CREF31: This needs more thought.]]

The section specification can be the empty string, in which case the content of the whole message is returned.

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.

Binary data is allowed in responses.

BODYSTRUCTURE

[[CREF32: Decide if this is going to be binary or human readable (e.g. a list).]]

The BODYSTRUCTURE FETCH item contains basic information about all chunks of the message which enables clients to download only specific chunks of the message without downloading the whole message. This can provide performance improvements when dealing with big attachments.

For each chunk of the message, the BODYSTRUCTURE includes (in the following order):

chunk type One octet (for binary representation).

body size A number giving the size of the chunk in octets (3 octets in network byte order for binary representation).

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

META Encrypted block of data that represents mutable state associated with the message, such as encrypted flags. [[CREF33: TBD]]

MODSEQ A 63 bit unsigned integer (expressed as a decimal), which represents the message modification sequence. [[CREF34: TBD]]

INTERNALDATE A string representing the internal date of the message (delivery date or date specified in the APPEND that created the message).

SIZE A number expressing the size of the message in octets.

UID A number expressing the unique identifier of the message.

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

8.5. Server Responses - Command Continuation Request

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

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

[[CREF35: Add non sync literals?]] The client is not permitted to send the octets of the 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.

9. Sample DMAP connection

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

TBD

10. Formal Syntax

The following syntax specification uses the Augmented Backus-Naur Form (ABNF) notation as specified in [ABNF].

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

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

(1) Except as noted otherwise, all alphabetic characters are case-insensitive. The use of upper or lower case characters to define token strings is for editorial clarity only. Implementations MUST accept these strings in a case-insensitive fashion.

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

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

TBD

11. Security Considerations

12. IANA Considerations

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

13. Normative References

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

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

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

[DIGEST-MD5]
Leach, P. and C. Newman, "Using Digest Authentication as a SASL Mechanism", RFC 2831, May 2000, <http://www.rfc-editor.org/info/rfc2831>.

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

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

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

[LANGUAGE-TAGS]
Alvestrand, H., "Content Language Headers", RFC 3282, May 2002, <http://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, March 1999, <http://www.rfc-editor.org/info/rfc2557>.

[MD5] Myers, J. and M. Rose, "The Content-MD5 Header Field", RFC 1864, October 1995, <http://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, November 1996, <http://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, November 1996, <http://www.rfc-editor.org/info/rfc2045>.

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

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

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

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

Appendix A. Change Log

1. TBD

Appendix B. Acknowledgement

This protocol was born after discussions with Ladar Levison. However he might not necessarily agree with its content and all errors belong to the editor of this document.

This document is heavily influenced by IMAP (RFC 3501) by Mark Crispin.

This document borrows some text from draft-kundrat-imap-submit-02.txt

Index

Author's Address

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

Email: Alexey.Melnikov@isode.com