Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 22 September 2020

S. Bosch
Dovecot Oy

21 March 2020

Sieve: Internationalized Email
draft-bosch-sieve-eai-00

Abstract

This document defines an extension to the Sieve language called "eai" which adds full support for internationalized email.

Status of This Memo

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

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

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

This Internet-Draft will expire on 22 September 2020.

Table of Contents

1. Introduction

Many parts of the Sieve mail filtering language [SIEVE] such as strings and comments are already designed primarily for use with the UTF-8 encoding [UTF-8] , thereby supporting all the international characters specified by the Unicode standard. Also, Sieve can already work with message header fields that contain UTF-8 characters, provided these are encoded using MIME encoded-word [MIME3]. However, the Sieve language was conceived before the Framework for Internationalized Email [RFC6530] was finished, which means that filtered email messages are still restricted to the conventional Internet Message Format [IMAIL], which mainly means that only the conventional US-ASCII email addresses can be used [SMTP]. This poses problems for using the Sieve language in a mail system where internationalized email is to be supported.

This document defines an extension to the Sieve language called "eai" which adds full support for internationalized email.

[FIXME: Any ideas for a better name for the extension?]

2. Conventions Used in This Document

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

3. Headers

The "eai" extension presented in this document does not alter the handling of conventional Internet messages [IMAIL], which have content type "message/rfc822". For such conventional messages, it expects UTF-8 characters in header field values to be encoded using MIME encoded-words [MIME3]. In contrast, when the filtered message (or message part) has content type "message/global" [RFC6532], the header field value can contain UTF-8 characters directly and MIME encoded-words SHOULD NOT be interpreted.

Note that internationalized email header names are still restricted to ASCII characters only [RFC6532], which means that the Sieve tests in which header fields are evaluated will never match when the

provided header name contains UTF-8 characters.

4. Addresses

Section 2.4.2.3 of [SIEVE] defines a constrained version of the US-ASCII email address format defined in [IMAIL], section 3 for use in the Sieve language. The address format defined in [IMAIL] is amended by [RFC6532], section 3.2, which adds internationalization support. The "eai" extension amends the Sieve language such that the changes in [RFC6532], section 3.2 also apply to the syntax of address values used in Sieve. Without the "eai" extension, only conventional addresses are recognized.

When the "eai" extension is active, the domain part of an email address used in Sieve MUST be evaluated as an U-Label as defined in [RFC5890], section 2.3.2.1. This means that both the domain and localpart of the email address are always evaluated as a string encoded in UTF-8.

[FIXME: Do we want to provide a special address part tag for evaluating the domain in A-label format instead?]

5. Modified commands

5.1. Test address

Refer to section Section 4 for changes to the email address format.

[FIXME: Any other changes?]

5.2. Test header

Refer to section Section 3 for changes to the email header field format.

FIXME: Any other changes?

5.3. Action redirect

The Sieve "redirect" action is used to send the message to another user at a supplied address. The only real change that the Sieve "eai" extension introduces for the "redirect" action is that the address parameter will support internationalized email address values. When such an internationalized address is used, it will need to use the SMTPUTF8 capability [RFC6531] in the SMTP session .

The "redirect" action may add headers to the message. When it amends a message that has "message/global" content type, it MUST use the header field format described in [RFC6531] when the Sieve "eai" extension is active. It SHOULD also do so when that extension is not active.

6. Modified extensions

6.1. Body Extension

The Sieve "body" extension [SIEVE-BODY] adds the "body" test. It tests for the occurrence of one or more strings in the body of an email message. Prior to matching content in a message body, transformations can be applied that filter and decode certain parts of the body. These transformations are selected by a body transform keyword parameter. If the body transform is ":content", the MIME

parts that have the specified content types are matched against independently. If the :content specification matches a "message/rfc822" MIME part, only the header of the nested message will be searched for the key strings, treating the header as a single string; the contents of the nested message body parts are only searched if their content type matches the :content specification. The Sieve "eai" extension modifies the ":content" transform of the "body" test to handle a "message/global" part the same as a "message/rfc822" part, as described above.

6.2. Convert Extension

[FIXME: Investigate RFC6558]

[FIXME: Define a conversion for downgrade?]

6.3. Editheader Extension

The Sieve "editheader" extension adds the "addheader" and "deleteheader" actions. The "addheader" action adds a header field to the filtered message and the "deleteheader" action can delete header fields. The "eai" extension presented in this document does not alter the processing of conventional Internet messages [IMAIL] with these actions. Specifically, if the specified field value does not match the [IMAIL] "unstructured" nonterminal syntax element, the implementation MUST either flag an error or encode the field using the encodings described in [MIME3] or [MIMEPARAM] to be compliant with [IMAIL]. In contrast, when the filtered message has content type "message/global" [RFC6532], the "addheader" action MUST NOT use the encodings described in [MIME3] or [MIMEPARAM]. Instead, it MUST write header values in UTF-8 encoding [UTF-8].

6.4. Envelope Extension

Refer to section Section 4 for changes to the email address format.

[FIXME: Any other changes?]

6.5. Enotify Extension

The Sieve "enotify" extension [SIEVE-NOTIFY] provides generic support for sending instant notifications. Using the specific "mailto" notification method [SIEVE-NOTIFY-MAILTO], notifications can be sent as an email message.

The "mailto" method is defined to use "mailto" URIs as specified in [URI-MAILTO], which is now obsolete. The Sieve "eai" extension updates the Sieve "mailto" notification method to use the updated "mailto" URI format instead [IRI-MAILTO], which adds better internationalization and compatibility with Internationalized Resource Identifiers [IRI].

[FIXME: Unfortunately, even the last mailto URI specification predates RFC653x, which means that no support is available for internationalized email addresses. Do we need to update the mailto URI specificiation, or am I missing an RFC?]

If one of the targets of the "mailto" notification method is an internationalized e-mail address, the produced notification message MUST be a "message/global" message, as specified by [RFC6532].

6.6. Reject and Extended Reject Extensions

The Sieve "reject" and "ereject" extensions [SIEVE-REJECT] respectively add the "reject" and "ereject" actions. These actions both cancel the implicit keep and refuse delivery of a message. One of the options for notifying the sender about the failure is sending back a Delivery Status Notification [DSN]. The format and rules for such notifications are updated by the Framework for Internationalized Email [RFC6530] in [RFC6533]. When the Sieve "eai" extension is also active, any DSN messages sent by the "reject" and "ereject" actions MUST additionally adhere to [RFC6533].

[FIXME: When the rejection message is shown in SMTP/LMTP reply, can we rely upon SMTPUTF8 to send UTF-8 messages there as well, thereby making the difference between reject and ereject mostly insignificant?]

6.7. Mime Extension

[FIXME: Investigate RFC5703]

6.8. Replace Extension

[FIXME: Investigate RFC5703]

6.9. Enclose Extension

[FIXME: Investigate RFC5703]

6.10. Other Extensions?

[FIXME: Any other extensions that need to be addressed?]

7. Downgrading

[FIXME: any words about downgrading and Sieve? RFC6530, RFC6858]

8. Mailing lists

[FIXME: Any mailing list EAI considerations in Sieve? RFC6783]

9. Examples

[FIXME: provide some]

10. Acknowledgements

[TBD; Reviews and comments are welcome.]

11. IANA Considerations

[FIXME: extension definitions]

12. Security Considerations

[FIXME: provide some]

13. References

13.1. Normative References

[DSN] Moore, K. and G. Vaudreuil, "An Extensible Message Format for Delivery Status Notifications", RFC 3464,

DOI 10.17487/RFC3464, January 2003,
<https://www.rfc-editor.org/info/rfc3464>.

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

[IRI] Duerst, M. and M. Suignard, "Internationalized Resource
Identifiers (IRIs)", RFC 3987, DOI 10.17487/RFC3987,
January 2005, <https://www.rfc-editor.org/info/rfc3987>.

[IRI-MAILTO]
Duerst, M., Masinter, L., and J. Zawinski, "The 'mailto'
URI Scheme", RFC 6068, DOI 10.17487/RFC6068, October 2010,
<https://www.rfc-editor.org/info/rfc6068>.

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

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

[MIME3] Moore, K., "MIME (Multipurpose Internet Mail Extensions)
Part Three: Message Header Extensions for Non-ASCII Text",
RFC 2047, DOI 10.17487/RFC2047, November 1996,
<https://www.rfc-editor.org/info/rfc2047>.

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

[RFC5890] Klensin, J., "Internationalized Domain Names for
Applications (IDNA): Definitions and Document Framework",
RFC 5890, DOI 10.17487/RFC5890, August 2010,
<https://www.rfc-editor.org/info/rfc5890>.

[RFC6530] Klensin, J. and Y. Ko, "Overview and Framework for
Internationalized Email", RFC 6530, DOI 10.17487/RFC6530,
February 2012, <https://www.rfc-editor.org/info/rfc6530>.

[RFC6531] Yao, J. and W. Mao, "SMTP Extension for Internationalized
Email", RFC 6531, DOI 10.17487/RFC6531, February 2012,
<https://www.rfc-editor.org/info/rfc6531>.

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

[RFC6533] Hansen, T., Ed., Newman, C., and A. Melnikov,
"Internationalized Delivery Status and Disposition
Notifications", RFC 6533, DOI 10.17487/RFC6533, February
2012, <https://www.rfc-editor.org/info/rfc6533>.

[SIEVE] Guenther, P. and T. Showalter, "Sieve: An Email Filtering
Language", RFC 5228, January 2008,
<https://www.rfc-editor.org/info/rfc5228>.

[SIEVE-BODY]

Degener, J. and P. Guenther, "Sieve Email Filtering: Body Extension", RFC 5173, DOI 10.17487/RFC5173, April 2008, <https://www.rfc-editor.org/info/rfc5173>.

[SIEVE-NOTIFY]
Melnikov, A., Ed., Leiba, B., Ed., Segmuller, W., and T. Martin, "Sieve Email Filtering: Extension for Notifications", RFC 5435, DOI 10.17487/RFC5435, January 2009, <https://www.rfc-editor.org/info/rfc5435>.

[SIEVE-NOTIFY-MAILTO]
Leiba, B. and M. Haardt, "Sieve Notification Mechanism: mailto", RFC 5436, DOI 10.17487/RFC5436, January 2009, <https://www.rfc-editor.org/info/rfc5436>.

[SIEVE-REJECT]
Stone, A., Ed., "Sieve Email Filtering: Reject and Extended Reject Extensions", RFC 5429, DOI 10.17487/RFC5429, March 2009, <https://www.rfc-editor.org/info/rfc5429>.

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

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

13.2. Informative References

[URI-MAILTO]
Hoffman, P., Masinter, L., and J. Zawinski, "The mailto URL scheme", RFC 2368, DOI 10.17487/RFC2368, July 1998, <https://www.rfc-editor.org/info/rfc2368>.

Author's Address

Dovecot Oy
Stephan Bosch
Lars Sonckin Kaari 16
FI-02600 Espoo
Finland

Email: stephan.bosch@dovecot.fi

EXTRA
Internet-Draft
Updates: 5228 (if approved)
Intended status: Standards Track
Expires: 19 December 2020

B. Gondwana, Ed.
FastMail

17 June 2020

Sieve Email Filtering: delivery by mailboxid
draft-gondwana-sieve-mailboxid-02

Abstract

The OBJECTID capability of the IMAP protocol (I-D.ietf-extra-imap-objectid) allows clients to identify mailboxes by a unique identifier which survives rename. In contrast, the Sieve mail filtering language (RFC 5228) currently has no such capability. This memo defines a Sieve extension that fills this gap: it adds a method for specifying the unique identifier of a mailbox as a target for fileinto rules, and a method for testing the existence of a mailbox by its unique identifier.

Status of This Memo

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

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

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

This Internet-Draft will expire on 19 December 2020.

Table of Contents

1. Introduction

Sieve rules are sometimes created using graphical interfaces which allow users to select the mailbox to be used as a target for a rule. If that mailbox is renamed, the client may also update its internal representation of the rule and update the sieve script to match, however this is a multi-step process and subject to partial failures. Also, if the folder is renamed by a different mechanism (e.g. another IMAP client) the rules will get out of sync.

By extending "fileinto" to reference an immutable mailboxid, sieve rules can continue to target the same mailbox, regardless of how it gets renamed.

2. Conventions Used In This Document

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

3. Sieve capability string

The server advertises the capability "mailboxid", and scripts which use the following extensions MUST explicitly request the capability "mailboxid".

Example:

```
require "mailboxid";
```

4. Argument ":mailboxid" to Command "fileinto"

Normally, the "fileinto" command delivers the message in the mailbox specified using its positional mailbox argument. However, if the optional ":mailboxid" argument is also specified, the "fileinto" command first checks whether a mailbox exists in the user's personal namespace [RFC2342] with the specified [I-D.ietf-extra-imap-objectid] MAILBOXID. If that is the case, that mailbox is used for delivery instead. If there is no such mailbox, the "fileinto" action proceeds as it would without the ":mailboxid" argument.

The tagged argument ":mailboxid" to fileinto consumes one additional token, a string with the objectid of the mailbox to file into.

Example:

```
require "fileinto";
require "mailboxid";

if header :contains ["from"] "coyote" {
    fileinto :mailboxid "F6352ae03-b7f5-463c-896f-d8b48ee3"
             "INBOX.harassment";
}
```

4.1. Interaction with "mailbox" extension

For servers which also support the [RFC5490] mailbox extension, the ":create" modifier to fileinto does not create mailbox with the specified mailboxid, however it may be specified and interacts as normal with all other extensions.

Example:

```
require "fileinto";
require "mailboxid";
require "mailbox";

fileinto :mailboxid "Fnosuch"
         :create
         "INBOX.no-such-folder";
            # creates INBOX.no-such-folder, but it doesn't
            # get the "Fnosuch" mailboxid.
```

4.2. Interaction with "specialuse" extension

For servers which also support [I-D.ietf-extra-sieve-special-use], if a fileinto command has both ":mailboxid" and ":special-use" specified, then the mailboxid is resolved first. If the mailboxid does not exist, then the special-use is evaluated next following the process specified in [I-D.ietf-extra-sieve-special-use] - this includes processing of [RFC5490] ":create" tags to add the special-use on creation.

Example:

```
require "fileinto";
require "mailboxid";
require "special-use";
if header :contains ["from"] "coyote" {
    fileinto :mailboxid "F6352ae03-b7f5-463c-896f-d8b48ee3"
             :specialuse "\\Junk"
             "INBOX.harassment";
}
```

Example:

```
require "fileinto";
require "mailboxid";
require "mailbox";
require "special-use";

fileinto :mailboxid "F1234567"
         :specialuse "\\Archive"
         :create
         "INBOX.Archive";
            # creates INBOX.Archive with use \Archive but
            # with a different mailboxid.
```

4.3. Interaction with "fcc" extension

This document extends the definition of the :fcc argument so that it can optionally be used with the ":mailboxid" argument.

FCC =/ [":mailboxid" <mailboxid: string>]

If the optional ":mailboxid" argument is specified with ":fcc", it instructs the Sieve interpreter to check whether a mailbox exists with the specific mailboxid. If such a mailbox exists, the generated message is filed into that mailbox. Otherwise, the generated message is filed into the ":fcc" target mailbox.

Example:

```
require ["enotify", "fcc", "mailboxid"];
notify :fcc "INBOX.Sent"
       :mailboxid "F6352ae03-b7f5-463c-896f-d8b48ee3"
       :message "You got mail!"
       "mailto:ken@example.com";
```

5. Test ":mailboxidexists"

The "mailboxidexists" test is true if all mailboxes listed in the "mailboxids" argument exist in the mailstore, and each allows the user in whose context the Sieve script runs to "deliver" messages into it. When the mailstore is an IMAP server, "delivery" of messages is possible if:

a) the READ-WRITE response code is present for the mailbox (see Section 7.1 of [RFC3501]), if IMAP Access Control List (ACL) [RFC4314] is not supported by the server, or

b) the user has ’p’ or ’i’ rights for the mailbox (see Section 5.2 of [RFC4314]).

Note that a successful "mailboxidexists" test for a mailbox doesn’t necessarily mean that a "fileinto :mailboxid" action on this mailbox would succeed. For example, the "fileinto" action might put user over quota. The "mailboxidexists" only verifies existence of the mailbox and whether the user in whose context the Sieve script runs has permissions to execute "fileinto" on it.

Example:

```
require "fileinto";
require "mailboxid";

if header :contains ["from"] "coyote" {
    if mailboxidexists "F6352ae03-b7f5-463c-896f-d8b48ee3" {
        fileinto :mailboxid "F6352ae03-b7f5-463c-896f-d8b48ee3"
                            "INBOX.harassment";
    } else {
        fileinto "INBOX.harassment";
    }
}
```

Not to implementers: this test behaves identically to the "mailboxexists" test defined in [RFC5490] but operates on mailboxids rather than mailbox names.

6. Formal Syntax

```
test /= ":mailboxidexists" string-list

tag /= ":mailboxid" string

If [I-D.ietf-extra-sieve-fcc] is supported:

FCC =/ [":mailboxid" <mailboxid: string>]
```

7. Security considerations

Because mailboxid is always generated by the server, implementations MUST NOT allow sieve to make an endrun around this protection by creating mailboxes with the specified ID by using ":create" and ":mailboxid" in a fileinto rule for a non-existant mailbox.

Implementers are referred to the security considerations sections of those documents in [RFC5228], [I-D.ietf-extra-imap-objectid].

8. IANA considerations

IANA are requested to add a capability to the sieve-extensions registry:

```
To: iana@iana.org
Subject: Registration of new Sieve extension

Capability name: mailboxid
Description: adds test for checking for mailbox existence by objectid
             and a new optional argument to fileinto to select the
             destination mailbox using objectid.
RFC number: this RFC
Contact address: The EXTRA discussion list <extra@ietf.org>
```

9. Acknowledgements

This document borrows heavily from [RFC5490] for the matching mailboxexists test, and from [I-D.ietf-extra-sieve-special-use] for an example of modifying the fileinto command.

Thanks to Ned Freed and Ken Murchison for feedback on the EXTRA mailing list.

10. Changes

(EDITOR: remove this section before publication)

10.1. draft-gondwana-sieve-mailboxid-02

* Update document date by a couple of years! Ooops, it got forgotten after a WGLC which got not dissent.

* Create xml2rfc v3 output.

10.2. draft-gondwana-sieve-mailboxid-01

* Switch to :mailboxid tagged parameter value with fallback mailbox name.

* Document interaction with "mailbox".

* Document interaction with "special-use".

* Document interaction with "fcc".

* Document security considerations around :mailboxid and :create.

10.3. draft-gondwana-sieve-mailboxid-00

* Initial version.

11. TODO

Is there a more explicit way to update the grammar? It seems less fully specified than IMAP.

12. Normative References

[RFC5228] Guenther, P., Ed. and T. Showalter, Ed., "Sieve: An Email Filtering Language", RFC 5228, DOI 10.17487/RFC5228, January 2008, <https://www.rfc-editor.org/info/rfc5228>.

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

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

[I-D.ietf-extra-imap-objectid] Gondwana, B., "IMAP Extension for object identifiers", Work in Progress, Internet-Draft, draft-ietf-extra-imap-objectid-08, 2 August 2018, <https://tools.ietf.org/html/draft-ietf-extra-imap-objectid-08>.

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

13. Informative References

[I-D.ietf-extra-sieve-special-use] Bosch, S., "Sieve Email Filtering: Delivering to Special-Use Mailboxes", Work in Progress, Internet-Draft, draft-ietf-extra-sieve-special-use-05, 24 January 2019, <https://tools.ietf.org/html/draft-ietf-extra-sieve-special-use-05>.

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

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

[RFC5490] Melnikov, A., "The Sieve Mail-Filtering Language -- Extensions for Checking Mailbox Status and Accessing Mailbox Metadata", RFC 5490, DOI 10.17487/RFC5490, March 2009, <https://www.rfc-editor.org/info/rfc5490>.

[I-D.ietf-extra-sieve-fcc] Murchison, K. and B. Gondwana, "Sieve Extension: File Carbon Copy (Fcc)", Work in Progress, Internet-Draft, draft-ietf-extra-sieve-fcc-09, 13 January 2019, <https://tools.ietf.org/html/draft-ietf-extra-sieve-fcc-09>.

Author's Address

Bron Gondwana (editor)
FastMail
Level 2, 114 William St
Melbourne VIC 3000
Australia

Email: brong@fastmailteam.com
URI: https://www.fastmail.com

Network Working Group
Internet-Draft
Obsoletes: 3501 (if approved)
Intended status: Standards Track
Expires: July 28, 2021

A. Melnikov, Ed.
Isode Ltd
B. Leiba, Ed.
Futurewei Technologies

January 24, 2021

Internet Message Access Protocol (IMAP) - Version 4rev2
draft-ietf-extra-imap4rev2-26

Abstract

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

IMAP4rev2 includes operations for creating, deleting, and renaming mailboxes, checking for new messages, permanently removing messages, setting and clearing flags, RFC 5322, RFC 2045 and RFC 2231 parsing, searching, and selective fetching of message attributes, texts, and portions thereof. Messages in IMAP4rev2 are accessed by the use of numbers. These numbers are either message sequence numbers or unique identifiers.

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

Status of This Memo

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

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

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

This Internet-Draft will expire on July 28, 2021.

Table of Contents

1. How to Read This Document

1.1. Organization of This Document

This document is written from the point of view of the implementor of an IMAP4rev2 client or server. Beyond the protocol overview in section 2, it is not optimized for someone trying to understand the operation of the protocol. The material in sections 3 through 5 provides the general context and definitions with which IMAP4rev2 operates.

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

1.2. Conventions Used in This Document

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

In examples, "C:" and "S:" indicate lines sent by the client and server respectively. Note that each line includes the terminating CRLF.

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

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

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

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

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

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

Characters are 8-bit UTF-8 (of which 7-bit US-ASCII is a subset) unless otherwise specified. Other character sets are indicated using a "CHARSET", as described in [MIME-IMT] and defined in [CHARSET]. CHARSETs have important additional semantics in addition to defining character set; refer to these documents for more detail.

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

1.3. Special Notes to Implementors

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

IMAP4rev2 is designed to be upwards compatible from the IMAP4rev1, the [IMAP2] and unpublished IMAP2bis protocols. IMAP4rev2 is largely compatible with the IMAP4rev1 protocol described in RFC 3501 and the IMAP4 protocol described in RFC 1730; the exception being in certain facilities added in RFC 1730 and RFC 3501 that proved problematic and were subsequently removed or replaced by better alternatives. In the course of the evolution of IMAP4rev2, some aspects in the earlier

protocols have become obsolete. Obsolete commands, responses, and data formats which an IMAP4rev2 implementation can encounter when used with an earlier implementation are described in Appendix E, Appendix A and [IMAP-OBSOLETE]. IMAP4rev2 supports 63bit body part and message sizes. IMAP4rev2 compatibility with BINARY and LIST-EXTENDED IMAP extensions are described in Appendix B and Appendix C respectively.

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

IMAP was originally developed for the older [RFC-822] standard, and as a consequence several fetch items in IMAP incorporate "RFC822" in their name. In all cases, "RFC822" should be interpreted as a reference to the updated [RFC-5322] standard.

2. Protocol Overview

2.1. Link Level

The IMAP4rev2 protocol assumes a reliable data stream such as that provided by TCP. When TCP is used, an IMAP4rev2 server listens on port 143 (cleartext port) or port 993 (Implicit TLS port).

2.2. Commands and Responses

An IMAP4rev2 connection consists of the establishment of a client/server network connection, an initial greeting from the server, and client/server interactions. These client/server interactions consist of a client command, server data, and a server completion result response.

All interactions transmitted by client and server are in the form of lines, that is, strings that end with a CRLF. The protocol receiver of an IMAP4rev2 client or server is either reading a line, or is reading a sequence of octets with a known count followed by a line.

2.2.1. Client Protocol Sender and Server Protocol Receiver

The client command begins an operation. Each client command is prefixed with an identifier (typically a short alphanumeric string, e.g., A0001, A0002, etc.) called a "tag". A different tag is generated by the client for each command. More formally: the client SHOULD generate a unique tag for every command, but a server MUST accept tag reuse.

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

There are two cases in which a line from the client does not represent a complete command. In one case, a command argument is quoted with an octet count (see the description of literal in Section 4.3); in the other case, the command arguments require server feedback (see the AUTHENTICATE command in Section 6.2.2). In either case, the server sends a command continuation request response if it is ready for the octets (if appropriate) and the remainder of the command. This response is prefixed with the token "+".

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

The protocol receiver of an IMAP4rev2 server reads a command line from the client, parses the command and its arguments, and transmits server data and a server command completion result response.

2.2.2. Server Protocol Sender and Client Protocol Receiver

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

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

The server completion result response indicates the success or failure of the operation. It is tagged with the same tag as the client command which began the operation. Thus, if more than one command is in progress, the tag in a server completion response identifies the command to which the response applies. There are three possible server completion responses: OK (indicating success),

NO (indicating failure), or BAD (indicating a protocol error such as unrecognized command or command syntax error).

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

The protocol receiver of an IMAP4rev2 client reads a response line from the server. It then takes action on the response based upon the first token of the response, which can be a tag, a "*", or a "+".

A client MUST be prepared to accept any server response at all times. This includes server data that was not requested. Server data SHOULD be remembered (cached), so that the client can reference its remembered copy rather than sending a command to the server to request the data. In the case of certain server data, the data MUST be remembered, as specified elsewhere in this document.

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

2.3. Message Attributes

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

2.3.1. Message Numbers

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

2.3.1.1. Unique Identifier (UID) Message Attribute

A UID is an unsigned non-zero 32-bit value assigned to each message, which when used with the unique identifier validity value (see below) forms a 64-bit value that MUST NOT refer to any other message in the mailbox or any subsequent mailbox with the same name forever. Unique identifiers are assigned in a strictly ascending fashion in the mailbox; as each message is added to the mailbox it is assigned a higher UID than 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 [IMAP-MODEL]); this is discussed further in [IMAP-DISC].

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

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

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

The unique identifier validity value is sent in a UIDVALIDITY response code in an OK untagged response at mailbox selection time. If unique identifiers from an earlier session fail to persist in this session, the unique identifier validity value MUST be greater than the one used in the earlier session. A good UIDVALIDITY value to use is a 32-bit representation of the current date/time when the value is assigned: this ensures that the value is unique and always increases. Another possible alternative is a global counter that gets incremented every time a mailbox is created.

> Note: Ideally, unique identifiers SHOULD persist at all times. Although this specification recognizes that failure to persist can be unavoidable in certain server environments, it strongly encourages message store implementation techniques that avoid this problem. For example:
>
> 1. Unique identifiers MUST be strictly ascending in the mailbox at all times. If the physical message store is re-ordered by a non-IMAP agent, this requires that the unique identifiers in

the mailbox be regenerated, since the former unique identifiers are no longer strictly ascending as a result of the re-ordering.

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

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

4. The combination of mailbox name, UIDVALIDITY, and UID must refer to a single immutable (or expunged) message on that server forever. In particular, the internal date, [RFC-5322] size, envelope, body structure, and message texts (all BODY[...] fetch data items) must never change. This does not include message numbers, nor does it include attributes that can be set by a STORE command (e.g., FLAGS). When a message is expunged, its UID MUST NOT be reused under the same UIDVALIDITY value.

2.3.1.2. Message Sequence Number Message Attribute

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

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

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

12345, there are exactly 286 messages which have lesser UIDs and 236 messages which have greater UIDs.

2.3.2. Flags Message Attribute

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

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

\Seen Message has been read

\Answered Message has been answered

\Flagged Message is "flagged" for urgent/special attention

\Deleted Message is "deleted" for removal by later EXPUNGE

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

\Recent This flag was in use in IMAP4rev1 and is now deprecated.

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

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

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

$MDNSent Message Disposition Notification [RFC8098] was generated and sent for this message. See [RFC3503] for more details on how this keyword is used.

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

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

$Phishing The $Phishing keyword can be used by a delivery agent to mark a message as highly likely to be a phishing email. An email that's determined to be a phishing email by the delivery agent should also be considered a junk email and have the appropriate junk filtering applied, including setting the $Junk flag and placing in the \Junk special-use mailbox (see Section 7.3.1) if available.
If both the $Phishing flag and the $Junk flag are set, the user agent should display an additional warning message to the user. Additionally the user agent may display a warning when clicking on any hyperlinks within the message.
The requirement for both $Phishing and $Junk to be set before a user agent displays a warning is for better backwards compatibility with existing clients that understand the $Junk flag but not the $Phishing flag. This so that when an unextended client removes the $Junk flag, an extended client will also show the correct state. See [IMAP-KEYWORDS-REG] for more information.

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

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

A flag can be permanent or session-only on a per-flag basis. Permanent flags are those which the client can add or remove from the message flags permanently; that is, concurrent and subsequent

sessions will see any change in permanent flags. Changes to session flags are valid only in that session.

2.3.3. Internal Date Message Attribute

An Internal Date message attribute is the internal date and time of the message on the server. This is not the date and time in the [RFC-5322] header, but rather a date and time which reflects when the message was received. In the case of messages delivered via [SMTP], this is the date and time of final delivery of the message as defined by [SMTP]. In the case of messages delivered by the IMAP4rev2 COPY or MOVE command, this SHOULD be the internal date and time of the source message. In the case of messages delivered by the IMAP4rev2 APPEND command, this SHOULD be the date and time as specified in the APPEND command description. All other cases are implementation defined.

2.3.4. [RFC-5322] Size Message Attribute

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

2.3.5. Envelope Structure Message Attribute

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

2.3.6. Body Structure Message Attribute

A Body Structure is a parsed representation of the [MIME-IMB] body structure information of the message.

2.4. Message Texts

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

3. State and Flow Diagram

Once the connection between client and server is established, an IMAP4rev2 connection is in one of four states. The initial state is identified in the server greeting. Most commands are only valid in certain states. It is a protocol error for the client to attempt a command while the connection is in an inappropriate state, and the

server will respond with a BAD or NO (depending upon server implementation) command completion result.

3.1. Not Authenticated State

In the not authenticated state, the client MUST supply authentication credentials before most commands will be permitted. This state is entered when a connection starts unless the connection has been pre-authenticated.

3.2. Authenticated State

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

3.3. Selected State

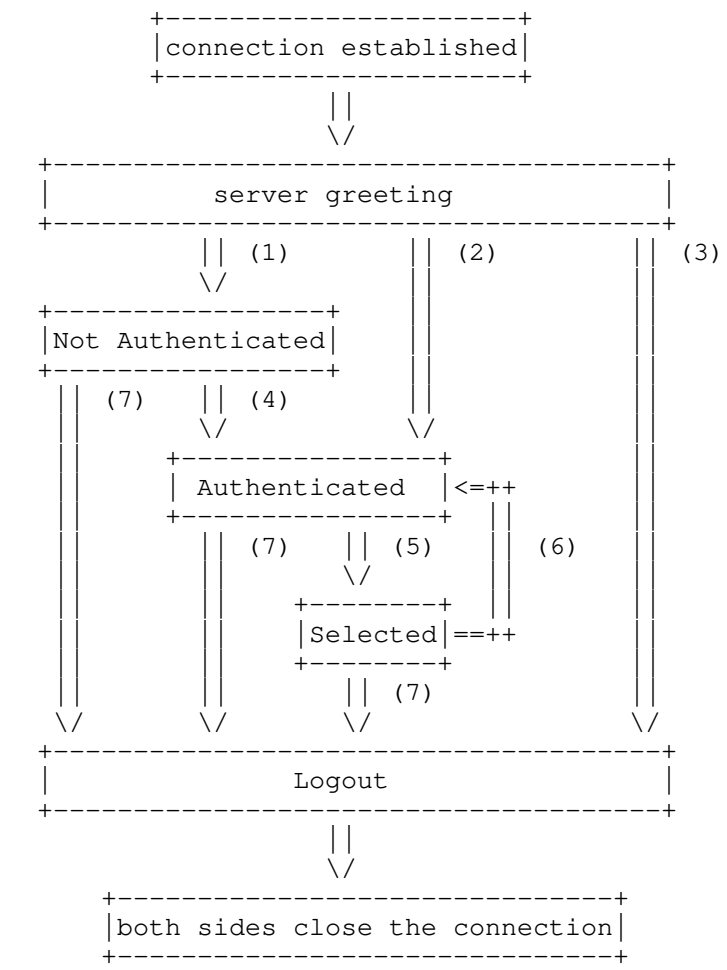
In a selected state, a mailbox has been selected to access. This state is entered when a mailbox has been successfully selected.

3.4. Logout State

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

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

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



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

4. Data Formats

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

4.1. Atom

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

4.1.1. Sequence set and UID set

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

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

4.2. Number

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

4.3. String

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

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

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

(described later in this document) before sending the octet data (and the remainder of the command).

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

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

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

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

4.3.1. 8-bit and Binary Strings

8-bit textual and binary mail is supported through the use of a [MIME-IMB] content transfer encoding. IMAP4rev2 implementations MAY transmit 8-bit or multi-octet characters in literals, but SHOULD do so only when the [CHARSET] is identified.

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

Although a BINARY content transfer encoding is defined, unencoded binary strings are not permitted, unless returned in a <literal8> in response to BINARY.PEEK[<section-binary>]<<partial>> or BINARY[<section-binary>]<<partial>> FETCH data item. A "binary string" is any string with NUL characters. A string with an excessive amount of CTL characters MAY also be considered to be binary. Unless returned in response to BINARY.PEEK[...]/BINARY[...] FETCH, client and server implementations MUST encode binary data into a textual form, such as BASE64, before transmitting the data.

4.4. Parenthesized List

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

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

4.5. NIL

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

> Note: NIL is never used for any data item 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.

Examples:

The following LIST response:

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

is equivalent to:

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

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

However, the following response

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

is not equivalent to:

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

The former means absence of the body part, while the latter means that it contains literal sequence of characters "NIL".

5. Operational Considerations

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

5.1. Mailbox Naming

In IMAP4rev2, Mailbox names are encoded in Net-Unicode [NET-UNICODE] (this differs from IMAP4rev1). Client implementations MAY attempt to create Net-Unicode mailbox names, and MUST interpret any 8-bit mailbox names returned by LIST as [NET-UNICODE]. Server implementations MUST prohibit the creation of 8-bit mailbox names that do not comply with Net-Unicode. However, servers MAY accept a de-normalized UTF-8 mailbox name and convert it to Unicode normalization form "NFC" (as per Net-Unicode requirements) prior to mailbox creation. Servers that choose to accept such de-normalized UTF-8 mailbox names MUST accept them in all IMAP commands that have a mailbox name parameter. In particular SELECT <name> must open the same mailbox that was successfully created with CREATE <name>, even if <name> is a de-normalized UTF-8 mailbox name.

The case-insensitive mailbox name INBOX is a special name reserved to mean "the primary mailbox for this user on this server". (Note that this special name may not exist on some servers for some users, for

example if the user has no access to personal namespace.) The interpretation of all other names is implementation-dependent.

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

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

1. Any character which is one of the atom-specials (see the Formal Syntax in Section 9) will require that the mailbox name be represented as a quoted string or literal.

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

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

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

5. Two characters, "#" and "&", have meanings by convention, and should be avoided except when used in that convention. See Section 5.1.2.1 and Appendix A.1 respectively.

5.1.1. Mailbox Hierarchy Naming

If it is desired to export hierarchical mailbox names, mailbox names MUST be left-to-right hierarchical using a single character to separate levels of hierarchy. The same hierarchy separator character is used for all levels of hierarchy within a single name.

5.1.2. Namespaces

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

namespace. In the typical case, there SHOULD be only one Personal Namespace per user on a server.

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

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

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

5.1.2.1. Historic Mailbox Namespace Naming Convention

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

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

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

5.1.2.2. Common namespace models

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

The "Personal Mailbox" model, in which the default namespace that is presented consists of only the user's personal mailboxes. To access shared mailboxes, the user must use an escape mechanism to reach another namespace.

The "Complete Hierarchy" model, in which the default namespace that is presented includes the user's personal mailboxes along with any other mailboxes they have access to.

5.2. Mailbox Size and Message Status Updates

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

Special rules exist for server notification of a client about the removal of messages to prevent synchronization errors; see the description of the EXPUNGE response (Section 7.5.1) for more detail. In particular, it is NOT permitted to send an EXISTS response that would reduce the number of messages in the mailbox; only the EXPUNGE response can do this.

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

5.3. Response when no Command in Progress

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

5.4. Autologout Timer

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

Note that this specification doesn't have any restrictions on autologout timer used before successful client authentication. In

particular, servers are allowed to use shortened pre-authentication timer to protect themselves from Denial of Service attacks.

5.5. Multiple Commands in Progress (Command Pipelining)

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

The exception is if an ambiguity would result because of a command that would affect the results of other commands. If the server detects a possible ambiguity, it MUST execute commands to completion in the order given by the client.

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

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

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

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

```
FETCH + NOOP + STORE

STORE + COPY + FETCH

COPY + COPY
```

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

FETCH + STORE + SEARCH + NOOP

STORE + COPY + EXPUNGE

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

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

6. Client Commands

IMAP4rev2 commands are described in this section. Commands are organized by the state in which the command is permitted. Commands which are permitted in multiple states are listed in the minimum permitted state (for example, commands valid in authenticated and selected state are listed in the authenticated state commands).

Command arguments, identified by "Arguments:" in the command descriptions below, are described by function, not by syntax. The precise syntax of command arguments is described in the Formal Syntax (Section 9).

Some commands cause specific server responses to be returned; these are identified by "Responses:" in the command descriptions below. See the response descriptions in the Responses section (Section 7) for information on these responses, and the Formal Syntax (Section 9) for the precise syntax of these responses. It is possible for server data to be transmitted as a result of any command. Thus, commands that do not specifically require server data specify "no specific responses for this command" instead of "none".

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

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

6.1. Client Commands - Any State

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

6.1.1. CAPABILITY Command

Arguments: none

Responses: REQUIRED untagged response: CAPABILITY

Result: OK - capability completed
BAD - arguments invalid

The CAPABILITY command requests a listing of capabilities (e.g. extensions and/or modifications of server behaviour) that the server supports. The server MUST send a single untagged CAPABILITY response with "IMAP4rev2" as one of the listed capabilities before the (tagged) OK response.

A capability name which begins with "AUTH=" indicates that the server supports that particular authentication mechanism as defined in [SASL]. All such names are, by definition, part of this specification.

Other capability names refer to extensions, revisions, or amendments to this specification. See the documentation of the CAPABILITY response in Section 7.2.2 for additional information. No capabilities, beyond the base IMAP4rev2 set defined in this specification, are enabled without explicit client action to invoke the capability.

Client and server implementations MUST implement the STARTTLS Section 6.2.1, LOGINDISABLED, and AUTH=PLAIN (described in [PLAIN]) capabilities. See the Security Considerations (Section 11) for important information.

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

```
   Example:    C: abcd CAPABILITY
               S: * CAPABILITY IMAP4rev2 STARTTLS AUTH=GSSAPI
                LOGINDISABLED
               S: abcd OK CAPABILITY completed
               C: efgh STARTTLS
               S: efgh OK STARTLS completed
               <TLS negotiation, further commands are under [TLS] layer>
               C: ijkl CAPABILITY
               S: * CAPABILITY IMAP4rev2 AUTH=GSSAPI AUTH=PLAIN
               S: ijkl OK CAPABILITY completed
```

6.1.2. NOOP Command

Arguments: none

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

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

The NOOP command always succeeds. It does nothing.

Since any command can return a status update as untagged data, the NOOP command can be used as a periodic poll for new messages or message status updates during a period of inactivity (the IDLE command Section 6.3.13 should be used instead of NOOP if real-time updates to mailbox state are desirable). The NOOP command can also be used to reset any inactivity autologout timer on the server.

```
   Example:    C: a002 NOOP
               S: a002 OK NOOP completed
                  . . .
               C: a047 NOOP
               S: * 22 EXPUNGE
               S: * 23 EXISTS
               S: * 14 FETCH (UID 1305 FLAGS (\Seen \Deleted))
               S: a047 OK NOOP completed
```

6.1.3. LOGOUT Command

Arguments: none

Responses: REQUIRED untagged response: BYE

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

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

```
Example:    C: A023 LOGOUT
            S: * BYE IMAP4rev2 Server logging out
            S: A023 OK LOGOUT completed
            (Server and client then close the connection)
```

6.2. Client Commands - Not Authenticated State

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

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

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

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

In addition to the universal commands (CAPABILITY, NOOP, and LOGOUT), the following commands are valid in the not authenticated state: STARTTLS, AUTHENTICATE and LOGIN. See the Security Considerations (Section 11) for important information about these commands.

6.2.1. STARTTLS Command

Arguments: none

Responses: no specific response for this command

Result: OK - starttls completed, begin TLS negotiation
NO - TLS negotiation can't be initiated, due to server configuration error

BAD - STARTTLS received after a successful TLS negotiation or arguments invalid

A TLS [TLS-1.3] negotiation begins immediately after the CRLF at the end of the tagged OK response from the server. Once a client issues a STARTTLS command, it MUST NOT issue further commands until a server response is seen and the TLS negotiation is complete. Some past server implementation incorrectly implemented STARTTLS processing and are known to contain STARTTLS plaintext command injection vulnerability [CERT-555316]. In order to avoid this vulnerability, server implementations MUST do one of the following If any data is received in the same TCP buffer after the CRLF that starts the STARTTLS command:

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

2. Extra data from the TCP buffer is thrown away.

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

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

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

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

6.2.2. AUTHENTICATE Command

Arguments: SASL authentication mechanism name
OPTIONAL initial response

Responses: continuation data can be requested

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

The AUTHENTICATE command indicates a [SASL] authentication mechanism to the server. If the server supports the requested authentication mechanism, it performs an authentication protocol exchange to authenticate and identify the client. It MAY also negotiate an OPTIONAL security layer for subsequent protocol interactions. If the requested authentication mechanism is not supported, the server SHOULD reject the AUTHENTICATE command by sending a tagged NO response.

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

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

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

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

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

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

If a security layer is negotiated through the [SASL] authentication exchange, it takes effect immediately following the CRLF that concludes the authentication exchange for the client, and the CRLF of the tagged OK response for the server.

While client and server implementations MUST implement the AUTHENTICATE command itself, it is not required to implement any authentication mechanisms other than the PLAIN mechanism described in [PLAIN]. Also, an authentication mechanism is not required to support any security layers.

> Note: a server implementation MUST implement a configuration in which it does NOT permit any plaintext password mechanisms, unless either the STARTTLS command has been negotiated, TLS has been negotiated on an Implicit TLS port, or some other mechanism that protects the session from password snooping has been provided. Server sites SHOULD NOT use any configuration which permits a plaintext password mechanism without such a protection mechanism against password snooping. Client and server implementations SHOULD implement additional [SASL] mechanisms that do not use plaintext passwords, such the GSSAPI mechanism described in [RFC4752], the SCRAM-SHA-256/SCRAM-SHA-256-PLUS [SCRAM-SHA-256] mechanisms and/or EXTERNAL [SASL] mechanism for mutual TLS authentication. (Note that SASL framework allows creation of SASL mechanisms that support 2FA (2-factor authentication), however none are fully ready to be recommended by this document.)

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

A server MAY include a CAPABILITY response code in the tagged OK response of a successful AUTHENTICATE command in order to send capabilities automatically. It is unnecessary for a client to send a separate CAPABILITY command if it recognizes these automatic

capabilities. This should only be done if a security layer was not negotiated by the AUTHENTICATE command, because the tagged OK response as part of an AUTHENTICATE command is not protected by encryption/integrity checking. [SASL] requires the client to re-issue a CAPABILITY command in this case. The server MAY advertise different capabilities after a successful AUTHENTICATE command.

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

The authorization identity passed from the client to the server during the authentication exchange is interpreted by the server as the user name whose privileges the client is requesting.

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

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

6.2.3. LOGIN Command

Arguments: user name
password

Responses: no specific responses for this command

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

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

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

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

Note: Use of the LOGIN command over an insecure network (such as the Internet) is a security risk, because anyone monitoring network traffic can obtain plaintext passwords. For that reason clients MUST NOT use LOGIN on unsecure networks.

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

6.3. Client Commands - Authenticated State

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

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

6.3.1. ENABLE Command

Arguments: capability names

Responses: no specific responses for this command

Result: OK - Relevant capabilities enabled
BAD - No arguments, or syntax error in an argument

Several IMAP extensions allow the server to return unsolicited responses specific to these extensions in certain circumstances. However, servers cannot send those unsolicited responses (with the exception of response codes (see Section 7.1) included in tagged or untagged OK/NO/BAD responses, which can always be sent) until they know that the clients support such extensions and thus won't choke on the extension response data.

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

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

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

If the ENABLE command is successful, the server MUST send an untagged ENABLED response Section 7.2.1, which includes all enabled extensions

as specified above. The ENABLED response is sent even if no extensions were enabled.

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

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

The ENABLE command can be issued multiple times in a session. It is additive; i.e., "ENABLE a b", followed by "ENABLE c" is the same as a single command "ENABLE a b c". When multiple ENABLE commands are issued, each corresponding ENABLED response SHOULD only contain extensions enabled by the corresponding ENABLE command, i.e. for the above example, the ENABLED response to "ENABLE c" should not contain "a" or "b".

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

The server MUST NOT change the CAPABILITY list as a result of executing ENABLE; i.e., a CAPABILITY command issued right after an ENABLE command MUST list the same capabilities as a CAPABILITY command issued before the ENABLE command. This is demonstrated in the following example. Note that below "X-GOOD-IDEA" is a fictitious extension capability that can be ENABLEd.

```
C: t1 CAPABILITY
S: * CAPABILITY IMAP4rev2 ID LITERAL+ X-GOOD-IDEA
S: t1 OK foo
C: t2 ENABLE CONDSTORE X-GOOD-IDEA
S: * ENABLED X-GOOD-IDEA
S: t2 OK foo
C: t3 CAPABILITY
S: * CAPABILITY IMAP4rev2 ID LITERAL+ X-GOOD-IDEA
S: t3 OK foo again
```

In the following example, the client enables CONDSTORE:

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

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

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

6.3.2. SELECT Command

Arguments: mailbox name

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

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

The SELECT command selects a mailbox so that messages in the mailbox can be accessed. Before returning an OK to the client, the server MUST send the following untagged data to the client. (The order of individual responses is not important.) Note that earlier versions of this protocol (e.g. IMAP2bis) only required the FLAGS and EXISTS untagged data; consequently, client implementations SHOULD implement default behavior for missing data as discussed with the individual item.

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

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

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

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

OK [UIDNEXT <n>] The next unique identifier value. Refer to Section 2.3.1.1 for more information.

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

Only one mailbox can be selected at a time in a connection; simultaneous access to multiple mailboxes requires multiple connections. The SELECT command automatically deselects any currently selected mailbox before attempting the new selection. Consequently, if a mailbox is selected and a SELECT command that fails is attempted, no mailbox is selected. When deselecting a selected mailbox, the server MUST return an untagged OK response with the "[CLOSED]" response code when the currently selected mailbox is closed (see Paragraph 10).

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

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

```
Example:    C: A142 SELECT INBOX
            S: * 172 EXISTS
            S: * OK [UIDVALIDITY 3857529045] UIDs valid
            S: * OK [UIDNEXT 4392] Predicted next UID
            S: * FLAGS (\Answered \Flagged \Deleted \Seen \Draft)
            S: * OK [PERMANENTFLAGS (\Deleted \Seen \*)] Limited
            S: * LIST () "/" INBOX
            S: A142 OK [READ-WRITE] SELECT completed
```

```
Example:    C: A142 SELECT INBOX
            S: * 172 EXISTS
            S: * OK [UIDVALIDITY 3857529045] UIDs valid
            S: * OK [UIDNEXT 4392] Predicted next UID
            S: * FLAGS (\Answered \Flagged \Deleted \Seen \Draft)
            S: * OK [PERMANENTFLAGS (\Deleted \Seen \*)] Limited
            S: A142 OK [READ-WRITE] SELECT completed
            [...some time later...]
            C: A143 SELECT Drafts
            S: * OK [CLOSED] Previous mailbox is now closed
            S: * 5 EXISTS
            S: * OK [UIDVALIDITY 9877410381] UIDs valid
            S: * OK [UIDNEXT 102] Predicted next UID
            S: * LIST () "/" Drafts
            S: * FLAGS (\Answered \Flagged \Deleted \Seen \Draft)
            S: * OK [PERMANENTFLAGS (\Deleted \Seen \Answered
                \Flagged \Draft \*)] System flags and keywords allowed
            S: A143 OK [READ-WRITE] SELECT completed
```

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

6.3.3. EXAMINE Command

Arguments: mailbox name

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

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

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

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

```
Example:    C: A932 EXAMINE blurdybloop
            S: * 17 EXISTS
            S: * OK [UIDVALIDITY 3857529045] UIDs valid
            S: * OK [UIDNEXT 4392] Predicted next UID
            S: * LIST () "/" blurdybloop
            S: * FLAGS (\Answered \Flagged \Deleted \Seen \Draft)
            S: * OK [PERMANENTFLAGS ()] No permanent flags permitted
            S: A932 OK [READ-ONLY] EXAMINE completed
```

6.3.4. CREATE Command

Arguments: mailbox name

Responses: OPTIONAL untagged response: LIST

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

The CREATE command creates a mailbox with the given name. An OK response is returned only if a new mailbox with that name has been created. It is an error to attempt to create INBOX or a mailbox with a name that refers to an extant mailbox. Any error in creation will return a tagged NO response. If a client attempts to create a UTF-8 mailbox name that is not a valid Net-Unicode name, the server MUST reject the creation or convert the name to Net-Unicode prior to creating the mailbox. If the server decides to convert (normalize) the name, it SHOULD return an untagged LIST with OLDNAME extended data item, with the OLDNAME value being the supplied mailbox name and the name parameter being the normalized mailbox name. (See Section 6.3.9.7 for more details.)

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

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

If the server's hierarchy separator character appears elsewhere in the name, the server SHOULD create any superior hierarchical names that are needed for the CREATE command to be successfully completed.

In other words, an attempt to create "foo/bar/zap" on a server in which "/" is the hierarchy separator character SHOULD create foo/ and foo/bar/ if they do not already exist.

If a new mailbox is created with the same name as a mailbox 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 in Section 6.4.9 for more detail.

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

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

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

6.3.5. DELETE Command

Arguments: mailbox name

Responses: OPTIONAL untagged response: LIST

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

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

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

inferior hierarchical names and also has the \Noselect mailbox name attribute (see the description of the LIST response (Section 7.3.1) for more details).

It is permitted to delete a name that has inferior hierarchical names and does not have the \Noselect mailbox name attribute. If the server implementation does not permit deleting the name while inferior hierarchical names exists then it SHOULD disallow the DELETE command by returning a tagged NO response. The NO response SHOULD include the HASCHILDREN response code. Alternatively the server MAY allow the DELETE command, but sets the \Noselect mailbox name attribute for that name.

If the server returns OK response, all messages in that mailbox are removed by the DELETE command.

The value of the highest-used unique identifier of the deleted mailbox MUST be preserved so that a new mailbox created with the same name will not reuse the identifiers of the former incarnation, unless the new incarnation has a different unique identifier validity value. See the description of the UID command in Section 6.4.9 for more detail.

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

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

```
Examples:   C: A682 LIST "" *
            S: * LIST () "/" blurdybloop
            S: * LIST (\Noselect) "/" foo
            S: * LIST () "/" foo/bar
            S: A682 OK LIST completed
            C: A683 DELETE blurdybloop
            S: A683 OK DELETE completed
            C: A684 DELETE foo
            S: A684 NO Name "foo" has inferior hierarchical names
            C: A685 DELETE foo/bar
            S: A685 OK DELETE Completed
            C: A686 LIST "" *
            S: * LIST (\Noselect) "/" foo
            S: A686 OK LIST completed
            C: A687 DELETE foo
            S: A687 OK DELETE Completed
            C: A82 LIST "" *
            S: * LIST () "." blurdybloop
            S: * LIST () "." foo
            S: * LIST () "." foo.bar
            S: A82 OK LIST completed
            C: A83 DELETE blurdybloop
            S: A83 OK DELETE completed
            C: A84 DELETE foo
            S: A84 OK DELETE Completed
            C: A85 LIST "" *
            S: * LIST () "." foo.bar
            S: A85 OK LIST completed
            C: A86 LIST "" %
            S: * LIST (\Noselect) "." foo
            S: A86 OK LIST completed
```

6.3.6. RENAME Command

```
Arguments:  existing mailbox name
            new mailbox name

Responses:  OPTIONAL untagged response: LIST

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

The RENAME command changes the name of a mailbox. A tagged OK response is returned only if the mailbox has been renamed. It is an error to attempt to rename from a mailbox name that does not exist or

to a mailbox name that already exists. Any error in renaming will return a tagged NO response.

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

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

The value of the highest-used unique identifier of the old mailbox name MUST be preserved so that a new mailbox created with the same name will not reuse the identifiers of the former incarnation, unless the new incarnation has a different unique identifier validity value. See the description of the UID command in Section 6.4.9 for more detail.

Renaming INBOX is permitted (i.e. it doesn't result in a tagged BAD response), and has special behavior. (Note that some servers disallow renaming INBOX by returning a tagged NO response, so clients need to be able to handle such RENAME failing). 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.

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

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

In both of the above cases: if the server automatically subscribes a mailbox when it is renamed, then the unsolicited LIST response for each affected subscribed mailbox name MUST include the \Subscribed

attribute. No unsolicited LIST responses need to be sent for children mailboxes, if any. When INBOX is successfully renamed, a new INBOX is assumed to be created. No unsolicited LIST responses need to be sent for INBOX in this case.

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

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

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

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

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

6.3.7. SUBSCRIBE Command

Arguments: mailbox

Responses: no specific responses for this command

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

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

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

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

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

6.3.8. UNSUBSCRIBE Command

Arguments: mailbox name

Responses: no specific responses for this command

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

The UNSUBSCRIBE command removes the specified mailbox name from the server's set of "active" or "subscribed" mailboxes as returned by the LIST (SUBSCRIBED) command. This command returns a tagged OK response if the unsubscription is successful or if the mailbox is not subscribed.

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

6.3.9. LIST Command

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

Arguments (extended): selection options (OPTIONAL)

reference name
mailbox patterns
return options (OPTIONAL)

Responses: untagged responses: LIST

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

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

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

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

1. if the first word after the command name begins with a parenthesis ("LIST selection options");

2. if the second word after the command name begins with a parenthesis;

3. if the LIST command has more than 2 parameters ("LIST return options")

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

In the basic syntax only, an empty ("" string) mailbox name argument is a special request to return the hierarchy delimiter and the root name of the name given in the reference. The value returned as the

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

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

The reference and mailbox name arguments are interpreted into a canonical form that represents an unambiguous left-to-right hierarchy. The returned mailbox names will be in the interpreted form, that we call "canonical LIST pattern" later in this document. To define the term "canonical LIST pattern" formally: it refers to the canonical pattern constructed internally by the server from the reference and mailbox name arguments.

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

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

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

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

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

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

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

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

The character "*" is a wildcard, and matches zero or more characters at this position. The character "%" is similar to "*", but it does not match a hierarchy delimiter. If the "%" wildcard is the last character of a mailbox name argument, matching levels of hierarchy are also returned. If these levels of hierarchy are not also selectable mailboxes, they are returned with the \Noselect mailbox name attribute (see the description of the LIST response (Section 7.3.1) for more details).

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

Selection options tell the server to limit the mailbox names that are selected by the LIST operation. If selection options are used, the mailboxes returned are those that match both the list of canonical LIST patterns and the selection options. Unless a particular selection option provides special rules, the selection options are cumulative: a mailbox that matches the mailbox patterns is selected

only if it also matches all of the selection options. (An example of a selection option with special rules is the RECURSIVEMATCH option.)

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

Initial selection options and return options are defined in the following subsections, and new ones will also be defined in extensions. Initial options defined in this document MUST be supported. Each non-initial option will be enabled by a capability string (one capability may enable multiple options), and a client MUST NOT send an option for which the server has not advertised support. A server MUST respond to options it does not recognize with a BAD response. The client SHOULD NOT specify any option more than once; however, if the client does this, the server MUST act as if it received the option only once. The order in which options are specified by the client is not significant.

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

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

The special name INBOX is included in the output from LIST, if INBOX is supported by this server for this user and if the uppercase string "INBOX" matches the interpreted reference and mailbox name arguments with wildcards as described above. The criteria for omitting INBOX is whether SELECT INBOX will return failure; it is not relevant whether the user’s real INBOX resides on this or some other server.

6.3.9.1. LIST Selection Options

The selection options defined in this specification are as follows:

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

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

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

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

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

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

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

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

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

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

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

6.3.9.2. LIST Return Options

The return options defined in this specification are as follows:

SUBSCRIBED - causes the LIST command to return subscription state for all matching mailbox names. The "\Subscribed" attribute MUST be supported and MUST be accurately computed when the SUBSCRIBED return option is specified. Further, all other mailbox attributes MUST be accurately computed (this differs from the behavior of the obsolete LSUB command from RFC 3501). Note that the above requirements don't override the requirement for the LIST command to return results quickly (see Section 6.3.9), i.e. server implementations need to compute results quickly and accurately. For example, server implementors might need to create quick access indices.

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

STATUS - requests STATUS response for each matching mailbox.

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

If an attempted STATUS for a listed mailbox fails because the mailbox can't be selected (e.g., if the "l" ACL right [RFC4314] is granted to the mailbox and the "r" right is not granted, or

due to a race condition between LIST and STATUS changing the mailbox to \NoSelect), the STATUS response MUST NOT be returned and the LIST response MUST include the \NoSelect attribute. This means the server may have to buffer the LIST reply until it has successfully looked up the necessary STATUS information.

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

6.3.9.3. General Principles for Returning LIST Responses

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

1. At most one LIST response should be returned for each mailbox name that matches the canonical LIST pattern. Server implementors must not assume that clients will be able to assemble mailbox attributes and other information returned in multiple LIST responses.

2. There are only two reasons for including a matching mailbox name in the responses to the LIST command (note that the server is allowed to return unsolicited responses at any time, and such responses are not governed by this rule):

   A. The mailbox name also satisfies the selection criteria.

   B. The mailbox name doesn't satisfy the selection criteria, but it has at least one descendant mailbox name that satisfies the selection criteria and that doesn't match the canonical LIST pattern.

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

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

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

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

6.3.9.4. Additional LIST-related Requirements on Clients

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

The following table summarizes inference rules.

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

6.3.9.5. The CHILDREN Return Option

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

Many IMAP4 clients present to the user a hierarchical view of the mailboxes that a user has access to. Rather than initially presenting to the user the entire mailbox hierarchy, it is often preferable to show to the user a collapsed outline list of the mailbox hierarchy (particularly if there is a large number of mailboxes). The user can then expand the collapsed outline hierarchy as needed. It is common to include within the collapsed hierarchy a visual clue (such as a ''+'') to indicate that there are child mailboxes under a particular mailbox. When the visual clue is clicked, the hierarchy list is expanded to show the child mailboxes. The CHILDREN return option provides a mechanism for a client to efficiently determine whether a particular mailbox has children, without issuing a LIST "" * or a LIST "" % for each mailbox name. The CHILDREN return option defines two new attributes that MUST be returned within a LIST response: \HasChildren and \HasNoChildren. Although these attributes MAY be returned in response to any LIST command, the CHILDREN return option is provided to indicate that the client particularly wants this information. If the CHILDREN return option is present, the server MUST return these attributes even if their computation is expensive.

\HasChildren

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

\HasNoChildren

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

It is an error for the server to return both a \HasChildren and a \HasNoChildren attribute in the same LIST response.

Note: the \HasNoChildren attribute should not be confused with the the \NoInferiors attribute, which indicates that no child mailboxes exist now and none can be created in the future.

6.3.9.6. CHILDINFO Extended Data Item

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

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

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

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

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

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

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

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

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

6.3.9.7. OLDNAME Extended Data Item

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

specify other conditions when OLDNAME extended data item should be included.

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

A deleted mailbox can be announced like this:

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

Example of a renamed mailbox:

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

6.3.9.8. LIST Command Examples

This example shows some uses of the basic LIST command:

```
Example:    C: A101 LIST "" ""
            S: * LIST (\Noselect) "/" ""
            S: A101 OK LIST Completed
            C: A102 LIST #news.comp.mail.misc ""
            S: * LIST (\Noselect) "." #news.
            S: A102 OK LIST Completed
            C: A103 LIST /usr/staff/jones ""
            S: * LIST (\Noselect) "/" /
            S: A103 OK LIST Completed
            C: A202 LIST ~/Mail/ %
            S: * LIST (\Noselect) "/" ~/Mail/foo
            S: * LIST () "/" ~/Mail/meetings
            S: A202 OK LIST completed
```

Extended examples:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Let's assume there is the following hierarchy:

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

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

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

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

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

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

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

(i.e., the mailbox "Foo" is not subscribed, but it has a child that is.)

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

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

or even this:

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

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

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

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

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

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

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

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

Let's assume there is the following hierarchy:

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

And that the following mailboxes are subscribed:

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

The client issues the following command first:

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

and the server may also include (but this would violate a SHOULD NOT in Section 3.5, because CHILDINFO is redundant)

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

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

Note that if the client issues

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

6.3.10. NAMESPACE Command

Arguments: none

Responses: REQUIRED untagged responses: NAMESPACE

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

The NAMESPACE command causes a single untagged NAMESPACE response to be returned. The untagged NAMESPACE response contains the prefix and hierarchy delimiter to the server's Personal Namespace(s), Other Users' Namespace(s), and Shared Namespace(s) that the server wishes to expose. The response will contain a NIL for any namespace class that is not available. The namespace-response-extensions ABNF non terminal is defined for extensibility and MAY be included in the NAMESPACE response.

Example 1:

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

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

Example 2:

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

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

Example 3:

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

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

Example 4:

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

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

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

Example 5:

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

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

< Automatically create a mailbox to store sent items.>

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

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

Example 6:

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

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

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

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

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

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

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

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

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

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

A client can construct a LIST command by appending a "%" to the Other Users' Namespace prefix to discover the Personal Namespaces of other users that are available to the currently authenticated user.

In response to such a LIST command, a server SHOULD NOT return user names that have not granted access to their personal mailboxes to the user in question.

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

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

Example 7:

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

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

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

Example 8:

A server that does not support providing a list of other user's mailboxes that are accessible to the currently logged on user. The mailboxes are listable if the client includes the name of the other user with the Other Users' Namespace prefix.

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

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

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

C: A003 LIST "" "#Users/Mike/%"
S: * LIST () "/" "#Users/Mike/INBOX"
S: * LIST () "/" "#Users/Mike/Foo"
S: A003 OK LIST command completed.
```

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

Example 9:

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

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

< List the mailboxes for user mark >

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

6.3.11. STATUS Command

Arguments: mailbox name
status data item names

Responses: REQUIRED untagged responses: STATUS

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

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

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

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

> Note: The STATUS command is intended to access the status of mailboxes other than the currently selected mailbox. Because the STATUS command can cause the mailbox to be opened internally, and

because this information is available by other means on the selected mailbox, the STATUS command SHOULD NOT be used on the currently selected mailbox. However, servers MUST be able to execute STATUS command on the selected mailbox. (This might also implicitly happen when STATUS return option is used in a LIST command).

The STATUS command MUST NOT be used as a "check for new messages in the selected mailbox" operation (refer to Section 7 and Section 7.4.1 for more information about the proper method for new message checking).

STATUS SIZE (see below) can take a significant amount of time, depending upon server implementation. Clients should use STATUS SIZE cautiously.

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

MESSAGES The number of messages in the mailbox.

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

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

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

DELETED The number of messages which have the \Deleted flag set.

SIZE The total size of the mailbox in octets. This is not strictly required to be an exact value, but it MUST be equal to or greater than the sum of the values of the RFC822.SIZE FETCH message data items (see Section 6.4.5) of all messages in the mailbox.

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

6.3.12. APPEND Command

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

Responses:  OPTIONAL untagged response: LIST
```

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

The APPEND command appends the literal argument as a new message to the end of the specified destination mailbox. This argument SHOULD be in the format of an [RFC-5322] or [I18N-HDRS] message. 8-bit characters are permitted in the message. A server implementation that is unable to preserve 8-bit data properly MUST be able to reversibly convert 8-bit APPEND data to 7-bit using a [MIME-IMB] content transfer encoding.

> Note: There may be exceptions, e.g., draft messages, in which required [RFC-5322] header fields are omitted in the message literal argument to APPEND. The full implications of doing so must be understood and carefully weighed.

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

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

If the append is unsuccessful for any reason, the mailbox MUST be restored to its state before the APPEND attempt (other than possibly keeping the changed mailbox's UIDNEXT value); no partial appending is permitted.

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

On successful completion of an APPEND, the server returns an APPENDUID response code (see Section 7.1), unless specified otherwise below.

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

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

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

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

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

```
Example:    C: A003 APPEND saved-messages (\Seen) {297}
            C: Date: Mon, 7 Feb 1994 21:52:25 -0800 (PST)
            C: From: Fred Foobar <foobar@example.com>
            C: Subject: afternoon meeting
            C: To: mooch@example.com
            C: Message-Id: <B27397-0100000@example.com>
            C: MIME-Version: 1.0
            C: Content-Type: TEXT/PLAIN; CHARSET=US-ASCII
            C:
            C: Hello Joe, do you think we can meet at 3:30 tomorrow?
            C:
            S: A003 OK [APPENDUID 38505 3955] APPEND completed
            C: A004 COPY 2:4 meeting
            S: A004 OK [COPYUID 38505 304,319:320 3956:3958] Done
            C: A005 UID COPY 305:310 meeting
            S: A005 OK No matching messages, so nothing copied
            C: A006 COPY 2 funny
            S: A006 OK Done
            C: A007 SELECT funny
            S: * 1 EXISTS
            S: * OK [UIDVALIDITY 3857529045] Validity session-only
            S: * OK [UIDNEXT 2] Predicted next UID
            S: * NO [UIDNOTSTICKY] Non-persistent UIDs
            S: * FLAGS (\Answered \Flagged \Deleted \Seen \Draft)
            S: * OK [PERMANENTFLAGS (\Deleted \Seen)] Limited
            S: * LIST () "." funny
            S: A007 OK [READ-WRITE] SELECT completed
```

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

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

6.3.13. IDLE Command

Arguments: none

Responses: continuation data will be requested; the client sends the continuation data "DONE" to end the command

Result: OK - IDLE completed after client sent "DONE"
NO - failure: the server will not allow the IDLE command at this time
BAD - command unknown or arguments invalid

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

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

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

The server MAY consider a client inactive if it has an IDLE command running, and if such a server has an inactivity timeout it MAY log the client off implicitly at the end of its timeout period. Because of that, clients using IDLE are advised to terminate the IDLE and re-issue it at least every 29 minutes to avoid being logged off. This still allows a client to receive immediate mailbox updates even though it need only "poll" at half hour intervals.

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

6.4. Client Commands - Selected State

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

In addition to the universal commands (CAPABILITY, NOOP, and LOGOUT), and the authenticated state commands (SELECT, EXAMINE, NAMESPACE, CREATE, DELETE, RENAME, SUBSCRIBE, UNSUBSCRIBE, LIST, STATUS, and APPEND), the following commands are valid in the selected state: CLOSE, UNSELECT, EXPUNGE, SEARCH, FETCH, STORE, COPY, MOVE, and UID.

6.4.1. CLOSE Command

Arguments: none

Responses: no specific responses for this command

Result: OK - close completed, now in authenticated state
BAD - command unknown or arguments invalid

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

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

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

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

6.4.2. UNSELECT Command

Arguments: none

Responses: no specific responses for this command

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

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

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

6.4.3. EXPUNGE Command

Arguments: none

Responses: untagged responses: EXPUNGE

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

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

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

Note: In this example, messages 3, 4, 7, and 11 had the \Deleted flag set. See the description of the EXPUNGE response (Section 7.5.1) for further explanation.

6.4.4. SEARCH Command

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

Responses: OPTIONAL untagged response: ESEARCH

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

The SEARCH command searches the mailbox for messages that match the given searching criteria.

The SEARCH command may contain result options. Result options control what kind of information is returned about messages matching the search criteria in an untagged ESEARCH response. If no result option is specified or empty list of options is specified "()", ALL is assumed (see below). The order of individual options is

arbitrary. Individual options may contain parameters enclosed in parentheses. (However, if an option has a mandatory parameter, which can always be represented as a number or a sequence-set, the option parameter does not need the enclosing parentheses. See the Formal Syntax (Section 9) for more details). If an option has parameters, they consist of atoms and/or strings and/or lists in a specific order. Any options not defined by extensions that the server supports MUST be rejected with a BAD response.

This document specifies the following result options:

MIN

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

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

MAX

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

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

ALL

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

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

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

SAVE

This option tells the server to remember the result of the SEARCH or UID SEARCH command (as well as any command based on

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

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

Note: future extensions to this document can allow servers to return multiple ESEARCH responses for a single extended SEARCH command. However all options specified above MUST result in a single ESEARCH response if used by themselves or in combination. This guaranty simplifies processing in IMAP4rev2 clients. Future SEARCH extensions that relax this restriction will have to describe how results from multiple ESEARCH responses are to be combined.

Searching criteria consist of one or more search keys.

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

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

The OPTIONAL [CHARSET] specification consists of the word "CHARSET" followed by a registered [CHARSET] [CHARSET-REG]. It indicates the [CHARSET] of the strings that appear in the search criteria. [MIME-IMB] content transfer encodings, and [MIME-HDRS] strings in [RFC-5322]/[MIME-IMB] headers, MUST be decoded before comparing text. Servers MUST support US-ASCII and UTF-8 charsets; other [CHARSET]s MAY be supported. Clients SHOULD use UTF-8. Note that if "CHARSET" is not provided IMAP4rev2 server MUST assume UTF-8, so selecting CHARSET UTF-8 is redundant. It is permitted for improved compatibility with existing IMAP4rev1 clients.

If the server does not support the specified [CHARSET], it MUST return a tagged NO response (not a BAD). This response SHOULD contain the BADCHARSET response code, which MAY list the [CHARSET]s supported by the server.

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

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

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

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

ANSWERED Messages with the \Answered flag set.

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

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

BODY <string> Messages that contain the specified string in the body of the message. Unlike TEXT (see below), this doesn't match any header fields. Servers are allowed to implement flexible matching for this search key, for example matching "swim" to both "swam" and "swum" in English language text or only doing full word matching (where "swim" will not match "swimming").

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

DELETED Messages with the \Deleted flag set.

DRAFT Messages with the \Draft flag set.

FLAGGED Messages with the \Flagged flag set.

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

HEADER <field-name> <string> Messages that have a header field with the specified field-name (as defined in [RFC-5322]) and that contains the specified string in the text of the header field (what comes after the colon). If the string to search is zero-length, this matches all messages that have a header field with the specified field-name regardless of the contents. Servers should use substring search for this SEARCH item, as clients can use it for automatic processing not initiated by end users. For example this can be used for searching for Message-ID or Content-Type header field values that need to be exact, or for searches in header fields that the IMAP server might not know anything about.

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

LARGER <n> Messages with an [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.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Note: Since this document is restricted to 7-bit ASCII text, it is not possible to show actual UTF-8 data. The "XXXXXX" is a

placeholder for what would be 6 octets of 8-bit data in an actual transaction.

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

```
Example:    C: A284 SEARCH RETURN (MIN) UNSEEN
            S: * ESEARCH (TAG "A284") MIN 4
            S: A284 OK SEARCH completed
```

The following example demonstrates that if the ESEARCH UID indicator is present, all data in the ESEARCH response is referring to UIDs; for example, the MIN result specifier will be followed by a UID.

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

The following example demonstrates returning the number of deleted messages:

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

6.4.4.1. SAVE result option and SEARCH result variable

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

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

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

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

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

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

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

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

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

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

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

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

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

Combination of Result option	"$" marker value
SAVE MIN	MIN
SAVE MAX	MAX
SAVE MIN MAX	MIN & MAX
SAVE * [m]	all found messages

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

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

6.4.4.2. Multiple Commands in Progress

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

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

6.4.4.3. Refusing to Save Search Results

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

6.4.4.4. Examples showing use of SAVE result option

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

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

Example 1:

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

The client can also pipeline the two commands:

Example 2:

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

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

Example 3:

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

Note that the second command in Example 3 can be replaced with:
C: A301 UID SEARCH $ SMALLER 4096
and the result of the command would be the same.

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

```
   Example 4:
            C: P282 SEARCH RETURN (SAVE) SINCE 1-Feb-1994
                NOT FROM "Smith"
            S: P282 OK SEARCH completed
            C: P283 SEARCH CHARSET UTF-8 (OR $ 1,3000:3021) TEXT {8}
            C: YYYYYYYY
            S: * ESEARCH (TAG "P283") ALL 882,1102,3003,3005:3006
            S: P283 OK completed
```

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

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

```
  Example 5:
            C: B282 SEARCH RETURN (SAVE) SINCE 1-Feb-1994
                NOT FROM "Smith"
            S: B282 OK SEARCH completed
            C: B283 SEARCH RETURN (SAVE) CHARSET KOI8-R
                (OR $ 1,3000:3021) TEXT {4}
            C: XXXX
            S: B283 NO [BADCHARSET UTF-8] KOI8-R is not supported
           //After this command the saved result variable contains
           //no messages.  A client that wants to reissue the B283
           //SEARCH command with another CHARSET would have to reissue
           //the B282 command as well.  One possible workaround for
           //this is to include the desired CHARSET parameter
           //in the earliest SEARCH RETURN (SAVE) command in a
           //sequence of related SEARCH commands, to cause
           //the earliest SEARCH in the sequence to fail.
           //A better approach might be to always use CHARSET UTF-8
           //instead.
```

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

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

```
   Example 6:
            C: E282 SEARCH RETURN (SAVE) SINCE 28-Oct-2006
                NOT FROM "Eric"
            C: E283 COPY $ "Other Messages"
           //The "$" contains no messages
            S: E282 OK SEARCH completed
            S: E283 OK COPY completed, nothing copied

   Example 7:
            C: F282 SEARCH RETURN (SAVE) KEYWORD $Junk
            C: F283 COPY $ "Junk"
            C: F284 STORE $ +FLAGS.Silent (\Deleted)
            S: F282 OK SEARCH completed
            S: F283 OK COPY completed
            S: F284 OK STORE completed

   Example 8:
            C: G282 SEARCH RETURN (SAVE) KEYWORD $Junk
            C: G283 SEARCH RETURN (ALL) SINCE 28-Oct-2006
                FROM "Eric"
           // The server can execute the two SEARCH commands
           // in any order, as they don't have any dependency.
           // For example, it may return:
            S: * ESEARCH (TAG "G283") ALL 3:15,27,29:103
            S: G283 OK SEARCH completed
            S: G282 OK SEARCH completed
```

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

```
   Example 9:
               C: H282 SEARCH RETURN (SAVE) KEYWORD $Junk
               C: H283 SEARCH RETURN (SAVE) SINCE 28-Oct-2006
                   FROM "Eric"
               S: H282 OK SEARCH completed
               S: H283 OK SEARCH completed
              // At this point "$" would contain results of H283
```

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

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

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

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

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

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

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

6.4.5. FETCH Command

```
   Arguments:  sequence set
               message data item names or macro

   Responses:  untagged responses: FETCH

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

The FETCH command retrieves data associated with a message in the mailbox. The data items to be fetched can be either a single atom or a parenthesized list.

Most data items, identified in the formal syntax (Section 9) under the msg-att-static rule, are static and MUST NOT change for any particular message. Other data items, identified in the formal syntax under the msg-att-dynamic rule, MAY change, either as a result of a STORE command or due to external events.

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

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

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

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

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

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

The currently defined data items that can be fetched are:

BINARY[<section-binary>]<<partial>>

Requests that the specified section be transmitted after performing Content-Transfer-Encoding-related decoding.

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

Note that this data item can only be requested for leaf (i.e. non multipart/*, non message/rfc822 and non message/global) body parts.

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

BINARY.SIZE[<section-binary>]

Requests the decoded size of the section (i.e., the size to expect in response to the corresponding FETCH BINARY request).

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

Note that this data item can only be requested for leaf (i.e. non multipart/*, non message/rfc822 and non message/global) body parts.

BODY Non-extensible form of BODYSTRUCTURE.

BODY[<section>]<<partial>>

The text of a particular body section.

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

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

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

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

The \Seen flag is implicitly set; if this causes the flags to change, they SHOULD be included as part of the FETCH responses.

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

BODYSTRUCTURE The [MIME-IMB] body structure of the message. This is computed by the server by parsing the [MIME-IMB] header fields in the [RFC-5322] header and [MIME-IMB] headers. See Section 7.5.2 for more details.

ENVELOPE The envelope structure of the message. This is computed by the server by parsing the [RFC-5322] header into the component parts, defaulting various fields as necessary. See Section 7.5.2 for more details.

FLAGS The flags that are set for this message.

INTERNALDATE The internal date of the message.

RFC822.SIZE The [RFC-5322] size of the message.

UID The unique identifier for the message.

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

6.4.5.1. FETCH section specification

Several FETCH data items reference "section" or "section-binary". The section specification is a set of zero or more part specifiers delimited by periods. A part specifier is either a part number or one of the following: HEADER, HEADER.FIELDS, HEADER.FIELDS.NOT, MIME, and TEXT. (Non numeric part specifiers have to be the last specifier in a section specification.) An empty section specification refers to the entire message, including the header.

Every message has at least one part number. Non-[MIME-IMB] messages, and non-multipart [MIME-IMB] messages with no encapsulated message, only have a part 1.

Multipart messages are assigned consecutive part numbers, as they occur in the message. If a particular part is of type message or multipart, its parts MUST be indicated by a period followed by the part number within that nested multipart part.

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

The HEADER, HEADER.FIELDS, HEADER.FIELDS.NOT, and TEXT part specifiers can be the sole part specifier or can be prefixed by one

or more numeric part specifiers, provided that the numeric part specifier refers to a part of type MESSAGE/RFC822 or MESSAGE/GLOBAL. The MIME part specifier MUST be prefixed by one or more numeric part specifiers.

The HEADER, HEADER.FIELDS, and HEADER.FIELDS.NOT part specifiers refer to the [RFC-5322] header of the message or of an encapsulated [MIME-IMT] MESSAGE/RFC822 or MESSAGE/GLOBAL message. HEADER.FIELDS and HEADER.FIELDS.NOT are followed by a list of field-name (as defined in [RFC-5322]) names, and return a subset of the header. The subset returned by HEADER.FIELDS contains only those header fields with a field-name that matches one of the names in the list; similarly, the subset returned by HEADER.FIELDS.NOT contains only the header fields with a non-matching field-name. The field-matching is ASCII range case-insensitive but otherwise exact. Subsetting does not exclude the [RFC-5322] delimiting blank line between the header and the body; the blank line is included in all header fetches, except in the case of a message which has no body and no blank line.

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

The TEXT part specifier refers to the text body of the message, omitting the [RFC-5322] header.

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

```
HEADER     ([RFC-5322] header of the message)
TEXT       ([RFC-5322] text body of the message) MULTIPART/MIXED
1          TEXT/PLAIN
2          APPLICATION/OCTET-STREAM
3          MESSAGE/RFC822
3.HEADER   ([RFC-5322] header of the message)
3.TEXT     ([RFC-5322] text body of the message) MULTIPART/MIXED
3.1        TEXT/PLAIN
3.2        APPLICATION/OCTET-STREAM
4          MULTIPART/MIXED
4.1        IMAGE/GIF
4.1.MIME   ([MIME-IMB] header for the IMAGE/GIF)
4.2        MESSAGE/RFC822
4.2.HEADER ([RFC-5322] header of the message)
4.2.TEXT   ([RFC-5322] text body of the message) MULTIPART/MIXED
4.2.1      TEXT/PLAIN
4.2.2      MULTIPART/ALTERNATIVE
4.2.2.1    TEXT/PLAIN
4.2.2.2    TEXT/RICHTEXT
```

6.4.6. STORE Command

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

Responses: untagged responses: FETCH

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

The STORE command alters data associated with a message in the mailbox. Normally, STORE will return the updated value of the data with an untagged FETCH response. A suffix of ".SILENT" in the data item name prevents the untagged FETCH, and the server SHOULD assume that the client has determined the updated value itself or does not care about the updated value.

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

The currently defined data items that can be stored are:

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

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

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

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

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

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

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

6.4.7. COPY Command

Arguments: sequence set
mailbox name

Responses: no specific responses for this command

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

The COPY command copies the specified message(s) to the end of the specified destination mailbox. The flags and internal date of the message(s) SHOULD be preserved in the copy.

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

If the COPY command is unsuccessful for any reason, server implementations MUST restore the destination mailbox to its state before the COPY attempt (other than possibly incrementing UIDNEXT), i.e. partial copy MUST NOT be done.

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

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

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

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

6.4.8. MOVE Command

```
   Arguments:  sequence set
               mailbox name

   Responses:  no specific responses for this command

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

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

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

1. [UID] COPY

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

3. UID EXPUNGE

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

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

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

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

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

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

An example:

```
C: a UID MOVE 42:69 foo
S: * OK [COPYUID 432432 42:69 1202:1229]
S: * 22 EXPUNGE
...More EXPUNGE responses from the server...
S: a OK Done
```

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

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

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

MOVE and UID MOVE can be pipelined with other commands, but care has to be taken. Both commands modify sequence numbers and also allow unrelated EXPUNGE responses. The renumbering of other messages in

the source mailbox following any EXPUNGE response can be surprising and makes it unsafe to pipeline any command that relies on message sequence numbers after a MOVE or UID MOVE. Similarly, MOVE cannot be pipelined with a command that might cause message renumbering. See Section 5.5, for more information about ambiguities as well as handling requirements for both clients and servers.

6.4.9. UID Command

Arguments: command name
command arguments

Responses: untagged responses: FETCH, ESEARCH, EXPUNGE

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

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

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

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

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

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

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

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

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

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

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

```
Example:    C: A999 UID FETCH 4827313:4828442 FLAGS
            S: * 23 FETCH (FLAGS (\Seen) UID 4827313)
            S: * 24 FETCH (FLAGS (\Seen) UID 4827943)
            S: * 25 FETCH (FLAGS (\Seen) UID 4828442)
            S: A999 OK UID FETCH completed
```

6.5. Client Commands - Experimental/Expansion

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

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

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

```
Example:    C: a441 CAPABILITY
            S: * CAPABILITY IMAP4rev2 XPIG-LATIN
            S: a441 OK CAPABILITY completed
            C: A442 XPIG-LATIN
            S: * XPIG-LATIN ow-nay eaking-spay ig-pay atin-lay
            S: A442 OK XPIG-LATIN ompleted-cay
```

7. Server Responses

Server responses are in three forms: status responses, server data, and command continuation 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 9).

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

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

Some status responses, and all server data, are untagged. An untagged response is indicated by the token "*" instead of a tag. Untagged status responses indicate server greeting, or server status

that does not indicate the completion of a command (for example, an impending system shutdown alert). For historical reasons, untagged server data responses are also called "unsolicited data", although strictly speaking, only unilateral server data is truly "unsolicited".

Certain server data MUST be remembered by the client when it is received; this is noted in the description of that data. Such data conveys critical information 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 remembered for later reference; if the client does not need to remember the data, or if remembering the data has no obvious purpose (e.g., a SEARCH response when no SEARCH command is in progress), the data can be ignored.

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

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

7.1. Server Responses - Generic Status Responses

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

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

The currently defined response codes are:

ALERT

The human-readable text contains a special alert that are presented to the user in a fashion that calls the user's attention to the message. Content of ALERT response codes received on a connection without TLS or SASL security layer confidentiality SHOULD be ignored by clients. If displayed, such alerts MUST be clearly marked as potentially suspicious. (Note that some existing clients are known to hyperlink returned text which make them very dangerous.) Alerts received after successful establishment of a TLS/SASL confidentiality layer MUST be presented to the user.

ALREADYEXISTS

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

C: o356 RENAME this that
S: o356 NO [ALREADYEXISTS] Mailbox "that" already exists

APPENDUID

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

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

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

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

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

AUTHENTICATIONFAILED

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

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

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

AUTHORIZATIONFAILED

Authentication succeeded in using the authentication identity, but the server cannot or will not allow the authentication identity to act as the requested authorization identity. This is only applicable when the authentication and authorization identities are different.

```
C: c1 AUTHENTICATE PLAIN
[...]
S: c1 NO [AUTHORIZATIONFAILED] No such authorization-ID


C: c2 AUTHENTICATE PLAIN
[...]
S: c2 NO [AUTHORIZATIONFAILED] Authenticator is not an admin
```

BADCHARSET

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

CANNOT

The operation violates some invariant of the server and can never succeed.

```
C: l create "///////"
```

```
S: l NO [CANNOT] Adjacent slashes are not supported
```

CAPABILITY

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

CLIENTBUG

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

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

CLOSED

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

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

CONTACTADMIN

The user should contact the system administrator or support desk.

```
C: e login "fred" "foo"
S: e NO [CONTACTADMIN]
```

COPYUID

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

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

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

This response code is returned in a tagged OK response to the COPY/UID COPY command or in the untagged OK response to the MOVE/UID MOVE command.

CORRUPTION

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

C: i select "/archive/projects/experiment-iv"
S: i NO [CORRUPTION] Cannot open mailbox

EXPIRED

Either authentication succeeded or the server no longer had the necessary data; either way, access is no longer permitted using that passphrase. The client or user should get a new passphrase.

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

EXPUNGEISSUED

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

C: h search from fred@example.com

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

HASCHILDREN

The mailbox delete operation failed because the mailbox has one or more children and the server doesn't allow deletion of mailboxes with children.

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

INUSE

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

The operation may succeed if the client tries again later.

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

LIMIT

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

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

NONEXISTENT

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

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

NOPERM

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

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

OVERQUOTA

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

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

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


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

PARSE

The human-readable text represents an error in parsing the [RFC-5322] header or [MIME-IMB] headers of a message in the mailbox.

PERMANENTFLAGS

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

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

PRIVACYREQUIRED

The operation is not permitted due to a lack of privacy. If Transport Layer Security (TLS) is not in use, the client could try STARTTLS (see Section 6.2.1) or alternatively reconnect to Implicit TLS port, and then repeat the operation.

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


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

READ-ONLY

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

READ-WRITE

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

SERVERBUG

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

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

TRYCREATE

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

UIDNEXT

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

UIDNOTSTICKY

The selected mailbox is supported by a mail store that does not support persistent UIDs; that is, UIDVALIDITY will be different each time the mailbox is selected. Consequently, APPEND or

COPY to this mailbox will not return an APPENDUID or COPYUID response code.

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

Note: servers SHOULD NOT have any UIDNOTSTICKY mail stores. This facility exists to support legacy mail stores in which it is technically infeasible to support persistent UIDs. This should be avoided when designing new mail stores.

UIDVALIDITY

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

UNAVAILABLE

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

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

UNKNOWN-CTE

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

Client implementations MUST ignore response codes that they do not recognize.

7.1.1. OK Response

Contents: OPTIONAL response code
human-readable text

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

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

```
Example:    S: * OK IMAP4rev2 server ready
            C: A001 LOGIN fred blurdybloop
            S: * OK [ALERT] System shutdown in 10 minutes
            S: A001 OK LOGIN Completed
```

7.1.2. NO Response

Contents: OPTIONAL response code
human-readable text

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

```
Example:    C: A222 COPY 1:2 owatagusiam
            S: * NO Disk is 98% full, please delete unnecessary data
            S: A222 OK COPY completed
            C: A223 COPY 3:200 blurdybloop
            S: * NO Disk is 98% full, please delete unnecessary data
            S: * NO Disk is 99% full, please delete unnecessary data
            S: A223 NO COPY failed: disk is full
```

7.1.3. BAD Response

Contents: OPTIONAL response code
human-readable text

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

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

7.1.4. PREAUTH Response

Contents: OPTIONAL response code
human-readable text

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

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

Example: S: * PREAUTH IMAP4rev2 server logged in as Smith

7.1.5. BYE Response

Contents: OPTIONAL response code
human-readable text

The BYE response is always untagged, and indicates that the server is about to close the connection. The human-readable text MAY be displayed to the user in a status report by the client. The BYE response is sent under one of four conditions:

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

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

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

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

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

response data from the server until the connection is closed; this will ensure that any pending untagged or completion responses are read and processed.

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

7.2. Server Responses - Server Status

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

7.2.1. ENABLED Response

Contents: capability listing

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

Example: S: * ENABLED CONDSTORE QRESYNC

7.2.2. CAPABILITY Response

Contents: capability listing

The CAPABILITY response occurs as a result of a CAPABILITY command. The capability listing contains a space-separated listing of capability names that the server supports. The capability listing MUST include the atom "IMAP4rev2", but note that it doesn't have to be the first capability listed. The order of capability names has no significance.

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

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

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

Other capability names indicate that the server supports an extension, revision, or amendment to the IMAP4rev2 protocol. Server responses MUST conform to this document until the client issues a command that uses the associated capability.

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

Client implementations SHOULD NOT require any capability name other than "IMAP4rev2", and possibly "STARTTLS" and "LOGINDISABLED" (on a non implicit TLS port). Client implementations MUST ignore any unknown capability names.

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

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

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

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

7.3. Server Responses - Mailbox Status

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

7.3.1. LIST Response

Contents: name attributes
hierarchy delimiter
name
OPTIONAL extension data

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

The following base mailbox name attributes are defined:

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

1. The mailbox name also satisfies the selection criteria (for example, it is subscribed and the "SUBSCRIBED" selection option has been specified).

2. "RECURSIVEMATCH" has been specified, and the mailbox name has at least one descendant mailbox name that does not match the LIST pattern and does match the selection criteria.

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

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

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

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

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

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

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

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

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

\Remote The mailbox is a remote mailbox.

It is an error for the server to return both a \HasChildren and a \HasNoChildren attribute in the same LIST response. A client that encounters a LIST response with both \HasChildren and \HasNoChildren attributes present should act as if both are absent in the LIST response.

Note: the \HasNoChildren attribute should not be confused with the \NoInferiors attribute, which indicates that no child mailboxes exist now and none can be created in the future.

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

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

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

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

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

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

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

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

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

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

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

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

The hierarchy delimiter is a character used to delimit levels of hierarchy in a mailbox name. A client can use it to create child

mailboxes, and to search higher or lower levels of naming hierarchy. All children of a top-level hierarchy node MUST use the same separator character. A NIL hierarchy delimiter means that no hierarchy exists; the name is a "flat" name.

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

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

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

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

7.3.2. NAMESPACE Response

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

The NAMESPACE response occurs as a result of a NAMESPACE command. It contains the prefix and hierarchy delimiter to the server's Personal Namespace(s), Other Users' Namespace(s), and Shared Namespace(s) that the server wishes to expose. The response will contain a NIL for any namespace class that is not available. Namespace-Response-Extensions ABNF non terminal is defined for extensibility and MAY be included in the response.

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

7.3.3. STATUS Response

Contents: name
status parenthesized list

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

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

7.3.4. ESEARCH Response

Contents: one or more search-return-data pairs

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

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

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

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

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

Example: S: * ESEARCH UID COUNT 5 ALL 4:19,21,28

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

Example: S: * ESEARCH COUNT 5 ALL 1:17,21

7.3.5. FLAGS Response

Contents: flag parenthesized list

The FLAGS response occurs as a result of a SELECT or EXAMINE command. The flag parenthesized list identifies the flags (at a minimum, the system-defined flags) that are applicable for this mailbox. Flags

other than the system flags can also exist, depending on server implementation.

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

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

7.4. Server Responses - Mailbox Size

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

7.4.1. EXISTS Response

Contents: none

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

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

Example: S: * 23 EXISTS

7.5. Server Responses - Message Status

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

7.5.1. EXPUNGE Response

Contents: none

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

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

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

An EXPUNGE response MUST NOT be sent when no command is in progress, nor while responding to a FETCH, STORE, or SEARCH command. This rule is necessary to prevent a loss of synchronization of message sequence numbers between client and server. A command is not "in progress" until the complete command has been received; in particular, a command is not "in progress" during the negotiation of command continuation.

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

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

Example: S: * 44 EXPUNGE

7.5.2. FETCH Response

Contents: message data

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

BINARY[<section-binary>]<<number>>

An <nstring> or <literal8> expressing the content of the specified section after removing any Content-Transfer-Encoding-related encoding. If <number> is present it refers to the offset within the DECODED section data.

If the domain of the decoded data is "8bit" and the data does not contain the NUL octet, the server SHOULD return the data in a <string> instead of a <literal8>; this allows the client to

determine if the "8bit" data contains the NUL octet without having to explicitly scan the data stream for for NULs.

Messaging clients and servers have been notoriously lax in their adherence to the Internet CRLF convention for terminating lines of textual data (text/* media types) in Internet protocols. When sending data in BINARY[...] FETCH data item, servers MUST ensure that textual line-oriented sections are always transmitted using the IMAP4 CRLF line termination syntax, regardless of the underlying storage representation of the data on the server.

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

BINARY.SIZE[<section-binary>]

The size of the section after removing any Content-Transfer-Encoding-related encoding. The value returned MUST match the size of the <nstring> or <literal8> that will be returned by the corresponding FETCH BINARY request.

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

BODY A form of BODYSTRUCTURE without extension data.

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

A string expressing the body contents of the specified section. The string SHOULD be interpreted by the client according to the content transfer encoding, body type, and subtype.

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

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

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

header portion of a MESSAGE/RFC822 or MESSAGE/GLOBAL part), MAY be in UTF-8. Note also that the [RFC-5322] delimiting blank line between the header and the body is not affected by header line subsetting; the blank line is always included as part of header data, except in the case of a message which has no body and no blank line.

Non-textual data such as binary data MUST be transfer encoded into a textual form, such as BASE64, prior to being sent to the client. To derive the original binary data, the client MUST decode the transfer encoded string.

BODYSTRUCTURE

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

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

Multiple parts are indicated by parenthesis nesting. Instead of a body type as the first element of the parenthesized list, there is a sequence of one or more nested body structures. The second element of the parenthesized list is the multipart subtype (mixed, digest, parallel, alternative, etc.).

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

Extension data follows the multipart subtype. Extension data is never returned with the BODY fetch, but can be returned with a BODYSTRUCTURE fetch. Extension data, if present, MUST be in the defined order. The extension data of a multipart body part are in the following order:

body parameter parenthesized list A parenthesized list of attribute/value pairs [e.g., ("foo" "bar" "baz" "rag") where "bar" is the value of "foo", and "rag" is the value of "baz"] as defined in [MIME-IMB]. Servers SHOULD decode parameter value continuations and parameter value character sets as described in [RFC2231], for example, if the message

contains parameters "baz*0", "baz*1" and "baz*2", the server should RFC2231-decode them, concatenate and return the resulting value as a parameter "baz". Similarly, if the message contains parameters "foo*0*" and "foo*1*", the server should RFC2231-decode them, convert to UTF-8, concatenate and return the resulting value as a parameter "foo*".

body disposition A parenthesized list, consisting of a disposition type string, followed by a parenthesized list of disposition attribute/value pairs as defined in [DISPOSITION]. Servers SHOULD decode parameter value continuations as described in [RFC2231].

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

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

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

The basic fields of a non-multipart body part are in the following order:

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

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

body parameter parenthesized list A parenthesized list of attribute/value pairs [e.g., ("foo" "bar" "baz" "rag") where "bar" is the value of "foo" and "rag" is the value of "baz"] as defined in [MIME-IMB].

body id A string giving the Content-ID header field value as defined in Section 7 of [MIME-IMB].

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

body encoding A string giving the content transfer encoding as defined in Section 6 of [MIME-IMB].

body size A number giving the size of the body in octets. Note that this size is the size in its transfer encoding and not the resulting size after any decoding.

A body type of type MESSAGE and subtype RFC822 contains, immediately after the basic fields, the envelope structure, body structure, and size in text lines of the encapsulated message.

A body type of type TEXT contains, immediately after the basic fields, the size of the body in text lines. Note that this size is the size in its content transfer encoding and not the resulting size after any decoding.

Extension data follows the basic fields and the type-specific fields listed above. Extension data is never returned with the BODY fetch, but can be returned with a BODYSTRUCTURE fetch. Extension data, if present, MUST be in the defined order.

The extension data of a non-multipart body part are in the following order:

body MD5 A string giving the body MD5 value as defined in [MD5].

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

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

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

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

ENVELOPE

A parenthesized list that describes the envelope structure of a message. This is computed by the server by parsing the [RFC-5322] header into the component parts, defaulting various fields as necessary.

The fields of the envelope structure are in the following order: date, subject, from, sender, reply-to, to, cc, bcc, in-reply-to, and message-id. The date, subject, in-reply-to, and message-id fields are strings. The from, sender, reply-to, to, cc, and bcc fields are parenthesized lists of address structures.

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

[RFC-5322] group syntax is indicated by a special form of address structure in which the host name field is NIL. If the mailbox name field is also NIL, this is an end of group marker (semi-colon in RFC 822 syntax). If the mailbox name field is non-NIL, this is a start of group marker, and the mailbox name field holds the group name phrase.

If the Date, Subject, In-Reply-To, and Message-ID header fields are absent in the [RFC-5322] header, the corresponding member of the envelope is NIL; if these header fields are present but empty the corresponding member of the envelope is the empty string.

> Note: some servers may return a NIL envelope member in the "present but empty" case. Clients SHOULD treat NIL and empty string as identical.
>
> Note: [RFC-5322] requires that all messages have a valid Date header field. Therefore, for a well-formed message the date member in the envelope can not be NIL or the empty string. However it can be NIL for a malformed or a draft message.
>
> Note: [RFC-5322] requires that the In-Reply-To and Message-ID header fields, if present, have non-empty content. Therefore, for a well-formed message the in-reply-to and message-id members in the envelope can not be the empty string. However they can still be the empty string for a malformed message.

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

If the Sender or Reply-To header fields are absent in the [RFC-5322] header, or are present but empty, the server sets

the corresponding member of the envelope to be the same value as the from member (the client is not expected to know to do this).

Note: [RFC-5322] requires that all messages have a valid From header field. Therefore, for a well-formed message the from, sender, and reply-to members in the envelope can not be NIL. However they can be NIL for a malformed or a draft message.

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

INTERNALDATE A string representing the internal date of the message.

RFC822.SIZE A number expressing the [RFC-5322] size of the message.

UID A number expressing the unique identifier of the message.

If the server chooses to send unsolicited FETCH responses, they MUST include UID FETCH item. Note that this is a new requirement when compared to RFC 3501.

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

7.6. Server Responses - Command Continuation Request

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

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

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

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

8. Sample IMAP4rev2 connection

The following is a transcript of an IMAP4rev2 connection. A long line in this sample is broken for editorial clarity.

```
S:   * OK IMAP4rev2 Service Ready
C:   a001 login mrc secret
S:   a001 OK LOGIN completed
C:   a002 select inbox
S:   * 18 EXISTS
S:   * FLAGS (\Answered \Flagged \Deleted \Seen \Draft)
S:   * OK [UIDVALIDITY 3857529045] UIDs valid
S:   * LIST () "/" INBOX ("OLDNAME" ("inbox"))
S:   a002 OK [READ-WRITE] SELECT completed
C:   a003 fetch 12 full
S:   * 12 FETCH (FLAGS (\Seen) INTERNALDATE "17-Jul-1996 02:44:25 -0700"
      RFC822.SIZE 4286 ENVELOPE ("Wed, 17 Jul 1996 02:23:25 -0700 (PDT)"
      "IMAP4rev2 WG mtg summary and minutes"
      (("Terry Gray" NIL "gray" "cac.washington.edu"))
      (("Terry Gray" NIL "gray" "cac.washington.edu"))
      (("Terry Gray" NIL "gray" "cac.washington.edu"))
      ((NIL NIL "imap" "cac.washington.edu"))
      ((NIL NIL "minutes" "CNRI.Reston.VA.US")
      ("John Klensin" NIL "KLENSIN" "MIT.EDU")) NIL NIL
      "<B27397-0100000@cac.washington.edu>")
       BODY ("TEXT" "PLAIN" ("CHARSET" "US-ASCII") NIL NIL "7BIT" 3028
       92))
S:    a003 OK FETCH completed
C:    a004 fetch 12 body[header]
S:    * 12 FETCH (BODY[HEADER] {342}
S:    Date: Wed, 17 Jul 1996 02:23:25 -0700 (PDT)
S:    From: Terry Gray <gray@cac.washington.edu>
S:    Subject: IMAP4rev2 WG mtg summary and minutes
S:    To: imap@cac.washington.edu
S:    cc: minutes@CNRI.Reston.VA.US, John Klensin <KLENSIN@MIT.EDU>
S:    Message-Id: <B27397-0100000@cac.washington.edu>
S:    MIME-Version: 1.0
S:    Content-Type: TEXT/PLAIN; CHARSET=US-ASCII
S:
S:    )
S:    a004 OK FETCH completed
C:    a005 store 12 +flags \deleted
S:    * 12 FETCH (FLAGS (\Seen \Deleted))
S:    a005 OK +FLAGS completed
C:    a006 logout
S:    * BYE IMAP4rev2 server terminating connection
S:    a006 OK LOGOUT completed
```

9. Formal Syntax

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

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

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

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

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

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

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

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

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

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

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

addr-name       = nstring
```

```
                    ; If non-NIL, holds phrase from [RFC-5322]
                    ; mailbox after removing [RFC-5322] quoting

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

append-uid      = uniqueid

astring         = 1*ASTRING-CHAR / string

ASTRING-CHAR   = ATOM-CHAR / resp-specials

atom            = 1*ATOM-CHAR

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

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

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

auth-type       = atom
                    ; Defined by [SASL]

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

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

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

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

body-extension  = nstring / number / number64 /
                   "(" body-extension *(SP body-extension) ")"
                    ; Future expansion.  Client implementations
                    ; MUST accept body-extension fields.  Server
                    ; implementations MUST NOT generate
                    ; body-extension fields except as defined by
                    ; future standard or standards-track
                    ; revisions of this specification.

body-ext-1part  = body-fld-md5 [SP body-fld-dsp [SP body-fld-lang
                  [SP body-fld-loc *(SP body-extension)]]]
                    ; MUST NOT be returned on non-extensible
                    ; "BODY" fetch
```

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

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

body-fld-desc   = nstring

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

body-fld-enc    = (DQUOTE ("7BIT" / "8BIT" / "BINARY" / "BASE64"/
                  "QUOTED-PRINTABLE") DQUOTE) / string
                  ; Content-Transfer-Encoding header field value.
                  ; Defaults to "7BIT" (as per RFC 2045)
                  ; if not present in the body part.

body-fld-id     = nstring

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

body-fld-loc    = nstring

body-fld-lines  = number64

body-fld-md5    = nstring

body-fld-octets = number

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

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

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

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

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

body-type-text  = media-text SP body-fields SP body-fld-lines

capability      = ("AUTH=" auth-type) / atom
```

```
                    ; New capabilities MUST begin with "X" or be
                    ; registered with IANA in
                    ; a standards-track, an experimental
                    ; or an informational RFC.

capability-data = "CAPABILITY" *(SP capability) SP "IMAP4rev2"
                  *(SP capability)
                    ; Servers MUST implement the STARTTLS, AUTH=PLAIN,
                    ; and LOGINDISABLED capabilities.
                    ; Servers which offer RFC 1730 compatibility MUST
                    ; list "IMAP4" as the first capability.
                    ; Servers which offer RFC 3501 compatibility MUST
                    ; list "IMAP4rev1" as one of capabilities.

CHAR            = <defined in [ABNF]>

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

charset         = atom / quoted

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

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

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

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

command-auth    = append / create / delete / enable / examine / list /
                  Namespace-Command /
                  rename / select / status / subscribe / unsubscribe /
```

```
                  idle
                    ; Valid only in Authenticated or Selected state

command-nonauth = login / authenticate / "STARTTLS"
                    ; Valid only when in Not Authenticated state

command-select  = "CLOSE" / "UNSELECT" / "EXPUNGE" / copy /
                   move / fetch / store / search / uid
                    ; Valid only when in Selected state

continue-req    = "+" SP (resp-text / base64) CRLF

copy            = "COPY" SP sequence-set SP mailbox

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

date            = date-text / DQUOTE date-text DQUOTE

date-day        = 1*2DIGIT
                    ; Day of month

date-day-fixed  = (SP DIGIT) / 2DIGIT
                    ; Fixed-format version of date-day

date-month      = "Jan" / "Feb" / "Mar" / "Apr" / "May" / "Jun" /
                  "Jul" / "Aug" / "Sep" / "Oct" / "Nov" / "Dec"

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

date-year       = 4DIGIT

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

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

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

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

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

```
enable          = "ENABLE" 1*(SP capability)

enable-data     = "ENABLED" *(SP capability)

envelope        = "(" env-date SP env-subject SP env-from SP
                  env-sender SP env-reply-to SP env-to SP env-cc SP
                  env-bcc SP env-in-reply-to SP env-message-id ")"

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

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

env-date        = nstring

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

env-in-reply-to = nstring

env-message-id  = nstring

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

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

env-subject     = nstring

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

esearch-response  = "ESEARCH" [search-correlator] [SP "UID"]
                    *(SP search-return-data)
                  ; ESEARCH response replaces SEARCH response
                  ; from IMAP4rev1.

examine         = "EXAMINE" SP mailbox

fetch           = "FETCH" SP sequence-set SP ("ALL" / "FULL" / "FAST" /
                  fetch-att / "(" fetch-att *(SP fetch-att) ")")

fetch-att       = "ENVELOPE" / "FLAGS" / "INTERNALDATE" /
                  "RFC822.SIZE" /
                  "BODY" ["STRUCTURE"] / "UID" /
                  "BODY" section [partial] /
                  "BODY.PEEK" section [partial] /
                  "BINARY" [".PEEK"] section-binary [partial] /
                  "BINARY.SIZE" section-binary

flag            = "\Answered" / "\Flagged" / "\Deleted" /
                  "\Seen" / "\Draft" / flag-keyword / flag-extension
```

```
                    ; Does not include "\Recent"

flag-extension  = "\" atom
                    ; Future expansion.  Client implementations
                    ; MUST accept flag-extension flags.  Server
                    ; implementations MUST NOT generate
                    ; flag-extension flags except as defined by
                    ; future standard or standards-track
                    ; revisions of this specification.
                    ; "\Recent" was defined in RFC 3501
                    ; and is now deprecated.

flag-fetch      = flag

flag-keyword    = "$MDNSent" / "$Forwarded" / "$Junk" /
                  "$NotJunk" / "$Phishing" / atom

flag-list       = "(" [flag *(SP flag)] ")"

flag-perm       = flag / "\*"

greeting        = "*" SP (resp-cond-auth / resp-cond-bye) CRLF

header-fld-name = astring

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

idle            = "IDLE" CRLF "DONE"

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

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


list-mailbox    = 1*list-char / string

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

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

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

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

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

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

list-select-opt =  list-select-base-opt / list-select-independent-opt
                   / list-select-mod-opt
            ; An option registration template is described in
            ; Section 9.3 of this document.

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

list-wildcards  = "%" / "*"

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

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

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

mailbox         = "INBOX" / astring
                    ; INBOX is case-insensitive.  All case variants of
                    ; INBOX (e.g., "iNbOx") MUST be interpreted as INBOX
                    ; not as an astring.  An astring which consists of
                    ; the case-insensitive sequence "I" "N" "B" "O" "X"
                    ; is considered to be INBOX and not an astring.
                    ;  Refer to section 5.1 for further
                    ; semantic details of mailbox names.

mailbox-data    =  "FLAGS" SP flag-list / "LIST" SP mailbox-list /
                   esearch-response /
                   "STATUS" SP mailbox SP "(" [status-att-list] ")" /
                   number SP "EXISTS" / Namespace-Response

mailbox-list    = "(" [mbx-list-flags] ")" SP
                    (DQUOTE QUOTED-CHAR DQUOTE / nil) SP mailbox
                    [SP mbox-list-extended]
            ; This is the list information pointed to by the ABNF
            ; item "mailbox-data", which is defined in [IMAP4]

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

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

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

mbox-or-pat =  list-mailbox / patterns

mbx-list-flags  = *(mbx-list-oflag SP) mbx-list-sflag
                  *(SP mbx-list-oflag) /
                  mbx-list-oflag *(SP mbx-list-oflag)

mbx-list-oflag  = "\Noinferiors" / child-mbox-flag /
                  "\Subscribed" / "\Remote" / flag-extension
                    ; Other flags; multiple possible per LIST response

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

media-basic     = ((DQUOTE ("APPLICATION" / "AUDIO" / "IMAGE" /
                  "FONT" / "MESSAGE" / "MODEL" / "VIDEO" ) DQUOTE)
                  / string)
```

```
                  SP media-subtype
                    ; Defined in [MIME-IMT].
                    ; FONT defined in RFC 8081.

media-message   = DQUOTE "MESSAGE" DQUOTE SP
                  DQUOTE ("RFC822" / "GLOBAL") DQUOTE
                    ; Defined in [MIME-IMT]

media-subtype   = string
                    ; Defined in [MIME-IMT]

media-text      = DQUOTE "TEXT" DQUOTE SP media-subtype
                    ; Defined in [MIME-IMT]

message-data    = nz-number SP ("EXPUNGE" / ("FETCH" SP msg-att))

move            = "MOVE" SP sequence-set SP mailbox

msg-att         = "(" (msg-att-dynamic / msg-att-static)
                   *(SP (msg-att-dynamic / msg-att-static)) ")"

msg-att-dynamic = "FLAGS" SP "(" [flag-fetch *(SP flag-fetch)] ")"
                    ; MAY change for a message

msg-att-static  = "ENVELOPE" SP envelope / "INTERNALDATE" SP date-time /
                  "RFC822.SIZE" SP number64 /
                  "BODY" ["STRUCTURE"] SP body /
                  "BODY" section ["<" number ">"] SP nstring /
                  "BINARY" section-binary SP (nstring / literal8) /
                  "BINARY.SIZE" section-binary SP number /
                  "UID" SP uniqueid
                    ; MUST NOT change for a message

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

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

namespace-command = "NAMESPACE"

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

namespace-response-extensions = *namespace-response-extension

namespace-response-extension = SP string SP
                  "(" string *(SP string) ")"
```

```
namespace-response = "NAMESPACE" SP namespace
                       SP namespace SP namespace
                     ; The first Namespace is the Personal Namespace(s).
                     ; The second Namespace is the Other Users'
                     ; Namespace(s).
                     ; The third Namespace is the Shared Namespace(s).

nil              = "NIL"

nstring          = string / nil

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

number64         = 1*DIGIT
                     ; Unsigned 63-bit integer
                     ; (0 <= n <= 9,223,372,036,854,775,807)

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

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


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

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

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

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

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

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

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

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

password         = astring

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

quoted           = DQUOTE *QUOTED-CHAR DQUOTE

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

quoted-specials = DQUOTE / "\"

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

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

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

response-done    = response-tagged / response-fatal

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

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

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

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

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

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

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

resp-specials   = "]"

resp-text       = ["[" resp-text-code "]" SP] [text]

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

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

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

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

search-key      = "ALL" / "ANSWERED" / "BCC" SP astring /
                  "BEFORE" SP date / "BODY" SP astring /
                  "CC" SP astring / "DELETED" / "FLAGGED" /
                  "FROM" SP astring / "KEYWORD" SP flag-keyword /
                  "ON" SP date / "SEEN" /
                  "SINCE" SP date / "SUBJECT" SP astring /
                  "TEXT" SP astring / "TO" SP astring /
```

```
                  "UNANSWERED" / "UNDELETED" / "UNFLAGGED" /
                  "UNKEYWORD" SP flag-keyword / "UNSEEN" /
                    ; Above this line were in [IMAP2]
                  "DRAFT" / "HEADER" SP header-fld-name SP astring /
                  "LARGER" SP number64 / "NOT" SP search-key /
                  "OR" SP search-key SP search-key /
                  "SENTBEFORE" SP date / "SENTON" SP date /
                  "SENTSINCE" SP date / "SMALLER" SP number64 /
                  "UID" SP sequence-set / "UNDRAFT" / sequence-set /
                  "(" search-key *(SP search-key) ")"

search-modifier-name = tagged-ext-label

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

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

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

search-return-data = "MIN" SP nz-number /
                     "MAX" SP nz-number /
                     "ALL" SP sequence-set /
                     "COUNT" SP number /
                     search-ret-data-ext
                     ; All return data items conform to
                     ; search-ret-data-ext syntax.
                     ; Note that "$" marker is not allowed
                     ; after the ALL return data item.

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

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

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

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

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

section-binary  = "[" [section-part] "]"

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

section-part    = nz-number *("." nz-number)
                    ; body part reference.
                    ; Allows for accessing nested body parts.

section-spec    = section-msgtext / (section-part ["." section-text])

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

select          = "SELECT" SP mailbox

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

seq-range       = seq-number ":" seq-number
                    ; two seq-number values and all values between
                    ; these two regardless of order.
                    ; Example: 2:4 and 4:2 are equivalent and indicate
                    ; values 2, 3, and 4.
```

```
                    ; Example: a unique identifier sequence range of
                    ; 3291:* includes the UID of the last message in
                    ; the mailbox, even if that value is less than 3291.

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

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

seq-last-command   = "$"

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

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

status-att-val  = ("MESSAGES" SP number) /
                  ("UIDNEXT" SP nz-number) /
                  ("UIDVALIDITY" SP nz-number) /
                  ("UNSEEN" SP number) /
                  ("DELETED" SP number) /
                  ("SIZE" SP number64)
                    ; Extensions to the STATUS responses
                    ; should extend this production.
                    ; Extensions should use the generic
                    ; syntax defined by tagged-ext.

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

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

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

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

```
string          = quoted / literal

subscribe       = "SUBSCRIBE" SP mailbox

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

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

tagged-ext-label    = tagged-label-fchar *tagged-label-char
                      ; Is a valid RFC 3501 "atom".

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

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

tagged-ext-comp     = astring /
                      tagged-ext-comp *(SP tagged-ext-comp) /
                      "(" tagged-ext-comp ")"
                      ; Extensions that follow this general
                      ; syntax should use nstring instead of
                      ; astring when appropriate in the context
                      ; of the extension.
                      ; Note that a message set or a "number"
                      ; can always be represented as an "atom".
                      ; An URL should be represented as
                      ; a "quoted" string.

tagged-ext-simple   = sequence-set / number / number64

tagged-ext-val      = tagged-ext-simple /
                      "(" [tagged-ext-comp] ")"

text            = 1*(TEXT-CHAR / UTF8-2 / UTF8-3 / UTF8-4)
                    ; Non ASCII text can only be returned
                    ; after ENABLE IMAP4rev2 command

TEXT-CHAR       = <any CHAR except CR and LF>

time            = 2DIGIT ":" 2DIGIT ":" 2DIGIT
                    ; Hours minutes seconds

uid             = "UID" SP
                  (copy / move / fetch / search / store / uid-expunge)
                    ; Unique identifiers used instead of message
                    ; sequence numbers

uid-expunge     = "EXPUNGE" SP sequence-set
```

```
                    ; Unique identifiers used instead of message
                    ; sequence numbers

uid-set         = (uniqueid / uid-range) *("," uid-set)

uid-range       = (uniqueid ":" uniqueid)
                    ; two uniqueid values and all values
                    ; between these two regards of order.
                    ; Example: 2:4 and 4:2 are equivalent.

uniqueid        = nz-number
                    ; Strictly ascending

unsubscribe     = "UNSUBSCRIBE" SP mailbox

userid          = astring

UTF8-CHAR       = <Defined in Section 4 of RFC 3629>

UTF8-2          = <Defined in Section 4 of RFC 3629>

UTF8-3          = <Defined in Section 4 of RFC 3629>

UTF8-4          = <Defined in Section 4 of RFC 3629>

vendor-token    = "vendor." name-component
                    ; Definition copied from RFC 2244.
                    ; MUST be registered with IANA

x-command       = "X" atom <experimental command arguments>

zone            = ("+" / "-") 4DIGIT
                    ; Signed four-digit value of hhmm representing
                    ; hours and minutes east of Greenwich (that is,
                    ; the amount that the given time differs from
                    ; Universal Time).  Subtracting the timezone
                    ; from the given time will give the UT form.
                    ; The Universal Time zone is "+0000".
```

10. Author's Note

This document is a revision or rewrite of earlier documents, and supercedes the protocol specification in those documents: RFC 3501, RFC 2060, RFC 1730, unpublished IMAP2bis.TXT document, RFC 1176, and RFC 1064.

11. Security Considerations

IMAP4rev2 protocol transactions, including electronic mail data, are sent in the clear over the network unless protection from snooping is negotiated. This can be accomplished either by the use of Implicit TLS port, STARTTLS command, negotiated privacy protection in the AUTHENTICATE command, or some other protection mechanism.

11.1. TLS related Security Considerations

This section applies to both use of STARTTLS command and Implicit TLS port.

IMAP client and server implementations MUST comply with relevant TLS recommendations from [RFC8314].

Clients and servers MUST implement TLS 1.2 [TLS-1.2] or newer. Use of TLS 1.3 [TLS-1.3] is RECOMMENDED. TLS 1.2 may be used only in cases where the other party has not yet implemented TLS 1.3. Additionally, when using TLS 1.2, IMAP implementations MUST implement TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 cipher suite. This is important as it assures that any two compliant implementations can be configured to interoperate. Other TLS cipher suites recommended in RFC 7525 are RECOMMENDED: TLS_DHE_RSA_WITH_AES_128_GCM_SHA256, TLS_DHE_RSA_WITH_AES_256_GCM_SHA384 and TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384. All other cipher suites are OPTIONAL. Note that this is a change from section 2.1 of [IMAP-TLS].

The list of mandatory-to-implement TLS 1.3 cipher suites is described in Section 9.1 of [TLS-1.3].

During the TLS negotiation [TLS-1.3][TLS-1.2], the client MUST check its understanding of the server hostname against the server's identity as presented in the server Certificate message, in order to prevent man-in-the-middle attacks. This procedure is described in [RFC7817].

Both the client and server MUST check the result of the STARTTLS command and subsequent TLS ([TLS-1.3][TLS-1.2]) negotiation to see whether acceptable authentication and/or privacy was achieved.

11.2. STARTTLS command versa use of Implicit TLS port

For maximum backward compatibility clients MUST implement both TLS negotiation on implicit TLS port and TLS negotiation using STARTTLS command.

Servers MUST implement TLS negotiation on implicit TLS port and SHOULD implement STARTTLS command on cleartext port.

Some site/firewall maintainers insist on TLS site-wide and prefer not to rely on a configuration option in each higher-level protocol. For this reason, IMAP4rev2 clients SHOULD try both ports 993 and 143 (and both IPv4 and IPv6) concurrently by default, unless overriden by either user configuration or DNS SRV records [RFC6186]. Note that if a server answers on both ports, it MUST allow STARTTLS command on port 143.

11.3. Client handling of unsolicited responses not suitable for the current connection state

Cleartext mail transmission (whether caused by firewall configuration errors that result in TLS stripping or weak security policies in email clients that choose not to negotiate TLS in the first place) can enable injection of responses that can confuse or even cause crashes in email clients. The following measures are recommended to minimize damage from them.

See Section 7.1.4 for special security considerations related to PREAUTH response.

Many server responses and response codes are only meaningful in authenticated or even selected state. However, nothing prevents a server (or a man-in-the-middle attacker) from sending such invalid responses in cleartext before STARTTLS/AUTHENTICATE commands are issued. Before authentication clients SHOULD ignore any responses other than CAPABILITY and server status responses (Section 7.1), as well as any response codes other than CAPABILITY. Client SHOULD ignore the ALERT response code until after TLS has been successfully negotiated (whether using STARTTLS or TLS negotiation on implicit TLS port). Unless explicitly allowed by an IMAP extension, when not in selected state clients MUST ignore responses/response codes related to message and mailbox status such as FLAGS, EXIST, EXPUNGE and FETCH.

11.4. COPYUID and APPENDUID response codes

The COPYUID and APPENDUID response codes return information about the mailbox, which may be considered sensitive if the mailbox has permissions set that permit the client to COPY or APPEND to the mailbox, but not SELECT or EXAMINE it.

Consequently, these response codes SHOULD NOT be issued if the client does not have access to SELECT or EXAMINE the mailbox.

11.5. LIST command and Other Users' namespace

In response to a LIST command containing an argument of the Other Users' Namespace prefix, a server SHOULD NOT list users that have not granted list access to their personal mailboxes to the currently authenticated user. Providing such a list, could compromise security by potentially disclosing confidential information of who is located on the server, or providing a starting point of a list of user accounts to attack.

11.6. Other Security Considerations

A server error message for an AUTHENTICATE command which fails due to invalid credentials SHOULD NOT detail why the credentials are invalid.

Use of the LOGIN command sends passwords in the clear. This can be avoided by using the AUTHENTICATE command with a [SASL] mechanism that does not use plaintext passwords, by first negotiating encryption via STARTTLS or some other protection mechanism.

A server implementation MUST implement a configuration that, at the time of authentication, requires:
(1) The STARTTLS command has been negotiated or TLS negotiated on implicit TLS port.
OR
(2) Some other mechanism that protects the session from password snooping has been provided.
OR
(3) The following measures are in place:
(a) The LOGINDISABLED capability is advertised, and [SASL] mechanisms (such as PLAIN) using plaintext passwords are NOT advertised in the CAPABILITY list.
AND
(b) The LOGIN command returns an error even if the password is correct.
AND
(c) The AUTHENTICATE command returns an error with all [SASL] mechanisms that use plaintext passwords, even if the password is correct.

A server error message for a failing LOGIN command SHOULD NOT specify that the user name, as opposed to the password, is invalid.

A server SHOULD have mechanisms in place to limit or delay failed AUTHENTICATE/LOGIN attempts.

A server SHOULD report any authentication failure and analyze such authentication failure attempt with regard to a password brute force attack as well as a password spraying attack. Accounts that match password spraying attacks MUST be blocked and request to change their passwords and only password with significant strength SHOULD be accepted.

Additional security considerations are discussed in the section discussing the AUTHENTICATE (see Section 6.2.2) and LOGIN (see Section 6.2.3) commands.

12. IANA Considerations

IANA is requested to update "Service Names and Transport Protocol Port Numbers" registry as follows:

1. Registration for TCP port 143 and the corresponding "imap" service name should be updated to point to this document and RFC 3501.

2. Registration for TCP port 993 and the corresponding "imaps" service name should be updated to point to this document, RFC 8314 and RFC 3501.

3. Both UDP port 143 and UDP port 993 should be marked as "Reserved" in the registry.

Additional IANA actions are specified in subsection of this section.

12.1. Updates to IMAP4 Capabilities registry

IMAP4 capabilities are registered by publishing a standards track or IESG approved informational or experimental RFC. The registry is currently located at: https://www.iana.org/assignments/imap4-capabilities

As this specification revises the AUTH= prefix, STARTTLS and LOGINDISABLED extensions, IANA is requested to update registry entries for these 3 extensions to point to this document and RFC 3501.

12.2. GSSAPI/SASL service name

GSSAPI/Kerberos/SASL service names are registered by publishing a standards track or IESG approved experimental RFC. The registry is currently located at: https://www.iana.org/assignments/gssapi-service-names

IANA is requested to update the "imap" service name previously registered in RFC 3501, to point to both this document and RFC 3501.

12.3. LIST Selection Options, LIST Return Options, LIST extended data items

[RFC5258] specifies IANA registration procedures for LIST Selection Options, LIST Return Options, LIST extended data items. This document doesn't change these registration procedures. In particular LIST selection options (Section 6.3.9.1) and LIST return options (Section 6.3.9.2) are registered using the procedure specified in Section 9 of [RFC5258] (and using the registration template from Section 9.3 of [RFC5258]). LIST Extended Data Items are registered using the registration template from Section 9.6 of [RFC5258]).

IANA is requested to add a reference to [RFCXXXX] for the "OLDNAME" LIST-EXTENDED extended data item entry. This is in addition to the existing reference to [RFC5465].

12.4. IMAP Mailbox Name Attributes and IMAP Response Codes

IANA is requested to update the "IMAP Mailbox Name Attributes" registry to point to this document in addition to RFC 3501.

IANA is requested to update the "IMAP Response Codes" registry to point to this document in addition to RFC 3501.

13. References

13.1. Normative References

[RFC4752] Melnikov, A., Ed., "The Kerberos V5 ("GSSAPI") Simple Authentication and Security Layer (SASL) Mechanism", RFC 4752, DOI 10.17487/RFC4752, November 2006, <https://www.rfc-editor.org/info/rfc4752>.

[RFC5258] Leiba, B. and A. Melnikov, "Internet Message Access Protocol version 4 - LIST Command Extensions", RFC 5258, DOI 10.17487/RFC5258, June 2008, <https://www.rfc-editor.org/info/rfc5258>.

[RFC5788] Melnikov, A. and D. Cridland, "IMAP4 Keyword Registry", RFC 5788, DOI 10.17487/RFC5788, March 2010, <https://www.rfc-editor.org/info/rfc5788>.

[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>.

[SCRAM-SHA-256]
Hansen, T., "SCRAM-SHA-256 and SCRAM-SHA-256-PLUS Simple Authentication and Security Layer (SASL) Mechanisms", RFC 7677, DOI 10.17487/RFC7677, November 2015, <https://www.rfc-editor.org/info/rfc7677>.

[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>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[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>.

[RFC2231] Freed, N. and K. Moore, "MIME Parameter Value and Encoded Word Extensions: Character Sets, Languages, and Continuations", RFC 2231, DOI 10.17487/RFC2231, November 1997, <https://www.rfc-editor.org/info/rfc2231>.

[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-1.2] 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>.

[TLS-1.3] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[UTF-7] Goldsmith, D. and M. Davis, "UTF-7 A Mail-Safe Transformation Format of Unicode", RFC 2152, May 1997, <http://www.rfc-editor.org/info/rfc2152>.

[UTF-8] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <http://www.rfc-editor.org/info/rfc3629>.

[MULTIAPPEND]
Crispin, M., "Internet Message Access Protocol (IMAP) - MULTIAPPEND Extension", RFC 3502, March 2003, <http://www.rfc-editor.org/info/rfc3502>.

[NET-UNICODE]
Klensin, J. and M. Padlipsky, "Unicode Format for Network Interchange", RFC 5198, DOI 10.17487/RFC5198, March 2008, <https://www.rfc-editor.org/info/rfc5198>.

[I18N-HDRS]
Yang, A., Steele, S., and N. Freed, "Internationalized Email Headers", RFC 6532, DOI 10.17487/RFC6532, February 2012, <https://www.rfc-editor.org/info/rfc6532>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/info/rfc4648>.

[RFC7817] Melnikov, A., "Updated Transport Layer Security (TLS) Server Identity Check Procedure for Email-Related Protocols", RFC 7817, DOI 10.17487/RFC7817, March 2016, <https://www.rfc-editor.org/info/rfc7817>.

[RFC8098] Hansen, T., Ed. and A. Melnikov, Ed., "Message Disposition Notification", STD 85, RFC 8098, DOI 10.17487/RFC8098, February 2017, <https://www.rfc-editor.org/info/rfc8098>.

[RFC8314] Moore, K. and C. Newman, "Cleartext Considered Obsolete: Use of Transport Layer Security (TLS) for Email Submission and Access", RFC 8314, DOI 10.17487/RFC8314, January 2018, <https://www.rfc-editor.org/info/rfc8314>.

[IMAP-IMPLEMENTATION]
Leiba, B., "IMAP4 Implementation Recommendations", RFC 2683, September 1999, <http://www.rfc-editor.org/info/rfc2683>.

[IMAP-MULTIACCESS]
Gahrns, M., "IMAP4 Multi-Accessed Mailbox Practice", RFC 2180, July 1997, <http://www.rfc-editor.org/info/rfc2180>.

13.2. Informative References (related protocols)

[CERT-555316]
CERT, "Vulnerability Note VU#555316: STARTTLS plaintext command injection vulnerability", September 2011, <https://www.kb.cert.org/vuls/id/555316>.

[RFC2193] Gahrns, M., "IMAP4 Mailbox Referrals", RFC 2193, DOI 10.17487/RFC2193, September 1997, <https://www.rfc-editor.org/info/rfc2193>.

[RFC3348] Gahrns, M. and R. Cheng, "The Internet Message Action Protocol (IMAP4) Child Mailbox Extension", RFC 3348, DOI 10.17487/RFC3348, July 2002, <https://www.rfc-editor.org/info/rfc3348>.

[RFC3503] Melnikov, A., "Message Disposition Notification (MDN) profile for Internet Message Access Protocol (IMAP)", RFC 3503, DOI 10.17487/RFC3503, March 2003, <https://www.rfc-editor.org/info/rfc3503>.

[RFC5256] Crispin, M. and K. Murchison, "Internet Message Access Protocol - SORT and THREAD Extensions", RFC 5256, DOI 10.17487/RFC5256, June 2008, <https://www.rfc-editor.org/info/rfc5256>.

[RFC5465] Gulbrandsen, A., King, C., and A. Melnikov, "The IMAP NOTIFY Extension", RFC 5465, DOI 10.17487/RFC5465, February 2009, <https://www.rfc-editor.org/info/rfc5465>.

[RFC6186] Daboo, C., "Use of SRV Records for Locating Email Submission/Access Services", RFC 6186, DOI 10.17487/RFC6186, March 2011, <https://www.rfc-editor.org/info/rfc6186>.

[RFC7888] Melnikov, A., Ed., "IMAP4 Non-synchronizing Literals", RFC 7888, DOI 10.17487/RFC7888, May 2016, <https://www.rfc-editor.org/info/rfc7888>.

[IMAP-DISC]
Melnikov, A., Ed., "Synchronization Operations for Disconnected IMAP4 Clients", RFC 4549, June 2006, <http://www.rfc-editor.org/info/rfc4549>.

[IMAP-I18N]
Newman, C., Gulbrandsen, A., and A. Melnikov, "Internet Message Access Protocol Internationalization", RFC 5255, DOI 10.17487/RFC5255, June 2008, <http://www.rfc-editor.org/info/rfc5255>.

[IMAP-MODEL]
Crispin, M., "Distributed Electronic Mail Models in IMAP4", RFC 1733, December 1994, <http://www.rfc-editor.org/info/rfc1733>.

[IMAP-UTF-8]
Resnick, P., Ed., Newman, C., Ed., and S. Shen, Ed., "IMAP Support for UTF-8", RFC 6855, DOI 10.17487/RFC6855, March 2013, <http://www.rfc-editor.org/info/rfc6855>.

[SMTP] Klensin, J., "Simple Mail Transfer Protocol", RFC 5321, October 2008, <http://www.rfc-editor.org/info/rfc5321>.

[RFC3516] Nerenberg, L., "IMAP4 Binary Content Extension", RFC 3516, DOI 10.17487/RFC3516, April 2003, <https://www.rfc-editor.org/info/rfc3516>.

[RFC4314] Melnikov, A., "IMAP4 Access Control List (ACL) Extension", RFC 4314, December 2005, <http://www.rfc-editor.org/info/rfc4314>.

[RFC2087] Myers, J., "IMAP4 QUOTA extension", RFC 2087, January 1997, <http://www.rfc-editor.org/info/rfc2087>.

[IMAP-URL]
Melnikov, A., Ed. and C. Newman, "IMAP URL Scheme", RFC 5092, DOI 10.17487/RFC5092, November 2007, <http://www.rfc-editor.org/info/rfc5092>.

[IMAP-KEYWORDS-REG]
IANA, "IMAP and JMAP Keywords", December 2009, <https://www.iana.org/assignments/imap-jmap-keywords/imap-jmap-keywords.xhtml>.

[IMAP-MAILBOX-NAME-ATTRS-REG]
IANA, "IMAP Mailbox Name Attributes", June 2018, <https://www.iana.org/assignments/imap-mailbox-name-attributes/imap-mailbox-name-attributes.xhtml>.

[CHARSET-REG]
IANA, "Character Set Registrations", May 2015, <https://www.iana.org/assignments/charset-reg/charset-reg.xhtml>.

13.3. Informative References (historical aspects of IMAP and related protocols)

[RFC3501] Crispin, M., "INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1", RFC 3501, DOI 10.17487/RFC3501, March 2003, <https://www.rfc-editor.org/info/rfc3501>.

[IMAP-COMPAT] Crispin, M., "IMAP4 Compatibility with IMAP2bis", RFC 2061, December 1996, <http://www.rfc-editor.org/info/rfc2061>.

[IMAP-HISTORICAL] Crispin, M., "IMAP4 Compatibility with IMAP2 and IMAP2bis", RFC 1732, December 1994, <http://www.rfc-editor.org/info/rfc1732>.

[IMAP-OBSOLETE] Crispin, M., "Internet Message Access Protocol - Obsolete Syntax", RFC 2062, December 1996, <http://www.rfc-editor.org/info/rfc2062>.

[IMAP2] Crispin, M., "Interactive Mail Access Protocol: Version 2", RFC 1176, August 1990, <http://www.rfc-editor.org/info/rfc1176>.

[RFC-822] Crocker, D., "STANDARD FOR THE FORMAT OF ARPA INTERNET TEXT MESSAGES", STD 11, RFC 822, August 1982, <http://www.rfc-editor.org/info/rfc822>.

[IMAP-TLS] Newman, C., "Using TLS with IMAP, POP3 and ACAP", RFC 2595, June 1999, <http://www.rfc-editor.org/info/rfc2595>.

Appendix A. Backward compatibility with IMAP4rev1

An implementation that wants to remain compatible with IMAP4rev1 can advertise both IMAP4rev1 and IMAP4rev2 in its CAPABILITY response/ response code. While some IMAP4rev1 responses were removed in IMAP4rev2, their presence will not break IMAP4rev2-only clients.

If both IMAP4rev1 and IMAP4rev2 are advertised, an IMAP client that wants to use IMAP4rev2 MUST issue an "ENABLE IMAP4rev2" command.

Servers advertising both IMAP4rev1 and IMAP4rev2 MUST NOT generate UTF-8 quoted strings unless the client has issued "ENABLE IMAP4rev2".

Consider implementation of mechanisms described or referenced in [IMAP-UTF-8] to achieve this goal.

Servers advertising both IMAP4rev1 and IMAP4rev2, and clients intending to be compatible with IMAP4rev1 servers MUST be compatible with the international mailbox naming convention described in the following subsection.

Also see Appendix D for special considerations for servers that support 63 bit body part/message sizes and want to advertise support for both IMAP4rev1 and IMAP4rev2.

A.1. Mailbox International Naming Convention for compatibility with IMAP4rev1

Support for the Mailbox International Naming Convention described in this section is not required for IMAP4rev2-only clients and servers. It is only used for backward compatibility with IMAP4rev1 implementations.

By convention, international mailbox names in IMAP4rev1 are specified using a modified version of the UTF-7 encoding described in [UTF-7]. Modified UTF-7 may also be usable in servers that implement an earlier version of this protocol.

In modified UTF-7, printable US-ASCII characters, except for "&", represent themselves; that is, characters with octet values 0x20-0x25 and 0x27-0x7e. The character "&" (0x26) is represented by the two-octet sequence "&-".

All other characters (octet values 0x00-0x1f and 0x7f-0xff) are represented in modified BASE64, with a further modification from [UTF-7] that "," is used instead of "/". Modified BASE64 MUST NOT be used to represent any printing US-ASCII character which can represent itself. Only characters inside the modified BASE64 alphabet are permitted in modified BASE64 text.

"&" is used to shift to modified BASE64 and "-" to shift back to US-ASCII. There is no implicit shift from BASE64 to US-ASCII, and null shifts ("-&" while in BASE64; note that "&-" while in US-ASCII means "&") are not permitted. However, all names start in US-ASCII, and MUST end in US-ASCII; that is, a name that ends with a non-ASCII ISO-10646 character MUST end with a "-").

The purpose of these modifications is to correct the following problems with UTF-7:

1. UTF-7 uses the "+" character for shifting; this conflicts with the common use of "+" in mailbox names, in particular USENET newsgroup names.

2. UTF-7's encoding is BASE64 which uses the "/" character; this conflicts with the use of "/" as a popular hierarchy delimiter.

3. UTF-7 prohibits the unencoded usage of "\"; this conflicts with the use of "\" as a popular hierarchy delimiter.

4. UTF-7 prohibits the unencoded usage of "~"; this conflicts with the use of "~" in some servers as a home directory indicator.

5. UTF-7 permits multiple alternate forms to represent the same string; in particular, printable US-ASCII characters can be represented in encoded form.

Although modified UTF-7 is a convention, it establishes certain requirements on server handling of any mailbox name with an embedded "&" character. In particular, server implementations MUST preserve the exact form of the modified BASE64 portion of a modified UTF-7 name and treat that text as case-sensitive, even if names are otherwise case-insensitive or case-folded.

Server implementations SHOULD verify that any mailbox name with an embedded "&" character, used as an argument to CREATE, is: in the correctly modified UTF-7 syntax, has no superfluous shifts, and has no encoding in modified BASE64 of any printing US-ASCII character which can represent itself. However, client implementations MUST NOT depend upon the server doing this, and SHOULD NOT attempt to create a mailbox name with an embedded "&" character unless it complies with the modified UTF-7 syntax.

Server implementations which export a mail store that does not follow the modified UTF-7 convention MUST convert to modified UTF-7 any mailbox name that contains either non-ASCII characters or the "&" character.

For example, here is a mailbox name which mixes English, Chinese, and Japanese text: ~peter/mail/&U,BTFw-/&ZeVnLIqe-

For example, the string "&Jjo!" is not a valid mailbox name because it does not contain a shift to US-ASCII before the "!". The correct form is "&Jjo-!". The string "&U,BTFw-&ZeVnLIqe-" is not permitted because it contains a superfluous shift. The correct form is "&U,BTF2XlZyyKng-".

Appendix B. Backward compatibility with BINARY extension

IMAP4rev2 incorporates subset of functionality provided by the BINARY extension [RFC3516], in particular it includes additional FETCH items (BINARY, BINARY.PEEK and BINARY.SIZE), but not extensions to the APPEND command. IMAP4rev2 implementations that supports full RFC 3516 functionality need to also advertise the BINARY capability in the CAPABILITY response/response code.

Appendix C. Backward compatibility with LIST-EXTENDED extension

IMAP4rev2 incorporates most of functionality provided by the LIST-EXTENDED extension [RFC5258]. In particular, multiple mailbox patterns syntax is not supported in IMAP4rev2, unless LIST-EXTENDED capability is also advertised in the CAPABILITY response/response code.

Appendix D. 63 bit body part and message sizes

IMAP4rev2 increases allowed body part and message sizes that servers can support from 32 to 63 bits. Server implementations don't have to support 63 bit long body parts/message sizes, however client implementations have to expect them.

As IMAP4rev1 didn't support 63 bit long body part/message sizes, there is an interoperability issue exposed by 63 bit capable servers that are accessible by both IMAP4rev1 and IMAP4rev2 email clients. As IMAP4rev1 would be unable to retrieve full content of messages bigger than 4Gb, such servers either need to replace messages bigger that 4Gb with messages under 4Gb or hide them from IMAP4rev1 clients. This document doesn't prescribe any implementation strategy to address this issue.

Appendix E. Changes from RFC 3501 / IMAP4rev1

Below is the summary of changes since RFC 3501:

1. Support for 64bit message and body part sizes.

2. Folded in IMAP NAMESPACE (RFC 2342), UNSELECT (RFC 3691), UIDPLUS (RFC 4315), ESEARCH (RFC 4731), SEARCHRES (RFC 5182), ENABLE (RFC 5161), IDLE (RFC 2177), SASL-IR (RFC 4959), LIST-EXTENDED (RFC 5258), LIST-STATUS (RFC 5819), MOVE (RFC 6851) and LITERAL- (RFC 7888) extensions. Also folded RFC 4466 (IMAP ABNF extensions), RFC 5530 (response codes), the FETCH side of the BINARY extension (RFC 3516) and the list of new mailbox attributes from SPECIAL-USE (RFC 6154).

3. Added STATUS SIZE (RFC 8438) and STATUS DELETED.

4. SEARCH command now requires to return ESEARCH response (SEARCH response is now deprecated).

5. Clarified which SEARCH keys have to use substring match and which don't.

6. Clarified that server should decode parameter value continuations as described in [RFC2231]. This requirement was hidden in RFC 2231 itself.

7. Clarified that COPYUID response code is returned for both MOVE and UID MOVE.

8. Tighen requirements about COPY/MOVE commands not creating target mailbox. Also require them to return TRYCREATE response code, if the target mailbox doesn't exist and can be created.

9. Added CLOSED response code from RFC 7162. SELECT/EXAMINE when a mailbox is already selected now requires a CLOSED response code to be returned.

10. SELECT/EXAMINE are now required to return untagged LIST response.

11. UNSEEN response code on SELECT/EXAMINE is now deprecated.

12. RECENT response on SELECT/EXAMINE, \Recent flag, RECENT STATUS, SEARCH NEW items are now deprecated.

13. Clarified that the server doesn't need to send a new PERMANENTFLAGS response code when a new keyword was successfully added and the server advertised * earlier for the same mailbox.

14. For future extensibility extended ABNF for tagged-ext-simple to allow for bare number64.

15. Added SHOULD level requirement on IMAP servers to support $MDNSent, $Forwarded, $Junk, $NonJunk and $Phishing keywords.

16. Mailbox names and message headers now allow for UTF-8. Support for Modified UTF-7 in mailbox names is not required, unless compatibility with IMAP4rev1 is desired.

17. Removed the CHECK command. Clients should use NOOP instead.

18. RFC822, RFC822.HEADER and RFC822.TEXT FETCH data items were deprecated. Clients should use the corresponding BODY[] variants instead.

19. LSUB command was deprecated. Clients should use LIST (SUBSCRIBED) instead.

20. IDLE command can now return updates not related to the currently selected mailbox state.

21. All unsolicited FETCH updates are required to include UID.

22. Clarified that client implementations MUST ignore response codes that they do not recognize. (Change from a SHOULD to a MUST.)

23. resp-text ABNF non terminal was updated to allow for empty text.

24. After ENABLE IMAP4rev2 human readable response text can include non ASCII encoded in UTF-8.

25. Updated to use modern TLS-related recommendations as per RFC 8314, RFC 7817, RFC 7525.

26. Added warnings about use of ALERT response codes and PREAUTH response.

27. Replaced DIGEST-MD5 SASL mechanism with SCRAM-SHA-256. DIGEST-MD5 was deprecated.

28. Clarified that any command received from the client resets server autologout timer.

29. Revised IANA registration procedure for IMAP extensions and removed "X" convention.

30. Loosened requirements on servers when closing connections to be more aligned with existing practices.

Appendix F. Other Recommended IMAP Extensions

Support for the following extensions is recommended for all IMAP client and servers. While they significantly reduce bandwidth and/or number of round trips used by IMAP in certain situations, the EXTRA WG decided that requiring them as a part of IMAP4rev2 would push the bar to implement too high for new implementations. Also note that absence of any IMAP extension from this list doesn’t make it somehow deficient or not recommended for use with IMAP4rev2.

1. QRESYNC and CONDSTORE extensions (RFC 7162). They make discovering changes to IMAP mailboxes more efficient, at the expense of storing a bit more state.

2. OBJECTID extension (RFC 8474) helps with preserving IMAP client cache when messages moved/copied or mailboxes are renamed.

Appendix G. Acknowledgement

Earlier versions of this document were edited by Mark Crispin. Sadly, he is no longer available to help with this work. Editors of this revisions are hoping that Mark would have approved.

Chris Newman has contributed text on I18N and use of UTF-8 in messages and mailbox names.

Thank you to Tony Hansen for helping with the index generation. Thank you to Murray Kucherawy, Timo Sirainen, Bron Gondwana, Stephan Bosch, Robert Sparks, Arnt Gulbrandsen and Daniel Migault for extensive feedback.

This document incorporates text from RFC 4315 (by Mark Crispin), RFC 4466 (by Cyrus Daboo), RFC 4731 (by Dave Cridland), RFC 5161 (by Arnt Gulbrandsen), RFC 5465 (by Arnt Gulbrandsen and Curtis King), RFC 5530 (by Arnt Gulbrandsen), RFC 5819 (by Timo Sirainen), RFC 6154 (by Jamie Nicolson), RFC 8438 (by Stephan Bosch) so work done by authors/editors of these documents is appreciated. Note that editors of this document were redacted from the above list.

The CHILDREN return option was originally proposed by Mike Gahrns and Raymond Cheng in [RFC3348]. Most of the information in Section 6.3.9.5 is taken directly from their original specification [RFC3348].

Thank you to Damian Poddebniak, Fabian Ising, Hanno Boeck and Sebastian Schinzel for pointing out that the ENABLE command should be a member of "command-auth" and not "command-any" ABNF production, as well as pointing out security issues associated with ALERT, PREAUTH and other responses received before authentication.

Index

Authors' Addresses

Alexey Melnikov (editor)
Isode Ltd
14 Castle Mews
Hampton, Middlesex TW12 2NP
UK

Email: Alexey.Melnikov@isode.com

Barry Leiba (editor)
Futurewei Technologies

Phone: +1 646 827 0648
Email: barryleiba@computer.org
URI: http://internetmessagingtechnology.org/

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 21, 2021

A. Melnikov
Isode

November 17, 2020

IMAP QUOTA Extension
draft-ietf-extra-quota-03

Abstract

The QUOTA extension of the Internet Message Access Protocol (RFC 3501) permits administrative limits on resource usage (quotas) to be manipulated through the IMAP protocol.

This memo obsoletes RFC 2087, but attempts to remain backwards compatible whenever possible.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on May 21, 2021.

Copyright Notice

Table of Contents

1. Document Conventions

In protocol examples, this document uses a prefix of "C: " to denote lines sent by the client to the server, and "S: " for lines sent by the server to the client. Lines prefixed with "// " are comments explaining the previous protocol line. These prefixes and comments are not part of the protocol. Lines without any of these prefixes are continuations of the previous line, and no line break is present in the protocol unless specifically mentioned.

Again, for examples, the hierarchy separator on the server is presumed to be "/" throughout. None of these assumptions is required nor recommended by this document.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 RFC2119 [RFC2119] 8174 [RFC8174] when, and only when, they appear in all capitals, as shown here.

Other capitalised words are IMAP4 [RFC3501] keywords or keywords from this document.

2. Introduction and Overview

This document defines a couple of extension to the Internet Message Access Protocol [RFC3501] for querying and manipulating administrative limits on resource usage (quotas).

The capability "QUOTA", denotes a RFC2087 [RFC2087] compliant server. Some responses and response codes defined in this document are not present in such servers (see Section 13 for more details), and clients MUST NOT rely on their presence in the absence of any capability beginning with "QUOTA=".

Any server compliant with this document MUST also return at least one capability starting with "QUOTA=RES-" prefix, as described in Section 3.1.

Any server compliant with this document that implements the SETQUOTA command (see Section 4.1.3) MUST also return the "QUOTASET" capability.

This document also reserves all other capabilities starting with "QUOTA=" prefix for future IETF stream standard track or experimental extensions to this document.

Quotas can be used to restrict clients for administrative reasons, but the QUOTA extension can also be used to indicate system limits and current usage levels to clients.

Although RFC2087 [RFC2087] specified an IMAP4 QUOTA extension, and this has seen deployment in servers, it has seen little deployment in clients. Since the meaning of the resources was left implementation-dependant, it was impossible for a client implementation to determine which resources were supported, and impossible to determine which mailboxes were in a given quota root, without a priori knowledge of the implementation.

3. Terms

3.1. Resource

A resource has a name, a formal definition.

3.1.1. Name

The resource name is an atom, as defined in IMAP4rev1 [RFC3501]. These MUST be registered with IANA. Implementation specific resources begin with "V-" .

Supported resource names MUST be advertised as a capability, by prepending the resource name with "QUOTA=RES-". A server compliant with this specification is not required to support all reported resource types on all quota roots.

3.1.2. Definition

The resource definition or document containing it, while not visible through the protocol, SHOULD be registered with IANA.

The usage of a resource MUST be represented as a 32 bit unsigned integer. 0 indicates that the resource is exhausted. Usage integers don’t necessarily represent proportional use, so clients MUST NOT compare available resource between two separate quota roots on the same or different servers.

Limits will be specified as, and MUST be represented as, an integer. 0 indicates that any usage is prohibited.

Limits may be hard or soft - that is, an implementation MAY choose, or be configured, to disallow any command if the limit on a resource is or would be exceeded.

All resources which the server handles must be advertised in a CAPABILITY constisting of the resource name prefixed by "QUOTA=RES-".

The resources STORAGE (Section 5.1), MESSAGE (Section 5.2), MAILBOX (Section 5.3) and ANNOTATION-STORAGE (Section 5.4) are defined in this document.

3.2. Quota Root

Each mailbox has zero or more implementation-defined named "quota roots". Each quota root has zero or more resource limits (quotas). All mailboxes that share the same named quota root share the resource limits of the quota root.

Quota root names need not be mailbox names, nor is there any relationship defined by this memo between a Quota root name and a mailbox name. A quota root name is an astring, as defined in IMAP4 [RFC3501]. It SHOULD be treated as an opaque string by any clients.

Quota roots are used since not all implementations may be able to calculate usage, or apply quotas, on arbitary mailboxes or mailbox hierarchies.

Not all resources may be limitable or calculatable for all quota roots. Further, not all resources may support all limits - some limits may be present in the underlying system. A server implementation of this memo SHOULD advise the client of such inherent limits, by generating QUOTA (Section 4.2.1) responses and SHOULD advise the client of which resources are limitable for a particular quota root. A SETQUOTA (Section 4.1.3) command MAY also round a quota limit in an implementation dependant way, if the granularity of the underlying system demands it. A client MUST be prepared for a SETQUOTA (Section 4.1.3) command to fail if a limit cannot be set.

Implementation Notes:
This means that, for example under UNIX, a quota root may have a MESSAGE (Section 5.2) quota always set due to the number of inodes available on the filesystem, and similarly STORAGE (Section 5.1) may be rounded to the nearest block and limited by free filesystem space.

4. Definitions

4.1. Commands

The following commands exist for manipulation and querying quotas.

4.1.1. GETQUOTA

Arguments: quota root

Responses: REQUIRED untagged responses: QUOTA

Result: OK - getquota completed
NO - getquota error: no such quota root, permission denied
BAD - command unknown or arguments invalid

The GETQUOTA command takes the name of a quota root and returns the quota root's resource usage and limits in an untagged QUOTA response. The client can try using any of the resource types returned in CAPABILITY response (i.e. all capability items with "QUOTA=RES-" prefix), however the server is not required to support any specific resource type for any particular quota root.

Example:

```
S: * CAPABILITY [...] QUOTA QUOTA=RES-STORAGE [...]
[...]
C: G0001 GETQUOTA "!partition/sda4"
S: * QUOTA "!partition/sda4" (STORAGE 104 10923847)
S: G0001 OK Getquota complete
```

4.1.2. GETQUOTAROOT

Arguments: mailbox name

Responses: REQUIRED untagged responses: QUOTAROOT, QUOTA

Result: OK - getquotaroot completed
NO - getquotaroot error: permission denied
BAD - command unknown or arguments invalid

The GETQUOTAROOT command takes a mailbox name and returns the list of quota roots for the mailbox in an untagged QUOTAROOT response. For each listed quota root, it also returns the quota root's resource usage and limits in an untagged QUOTA response.

Note that the mailbox name parameter doesn't have to reference an existing mailbox. This can be handy in order to determine which quotaroot would apply to a mailbox when it gets created.

Example:

```
S: * CAPABILITY [...] QUOTA QUOTA=RES-STORAGE QUOTA=RES-MESSAGE
[...]
[...]
C: G0002 GETQUOTAROOT INBOX
S: * QUOTAROOT INBOX "#user/alice" "!partition/sda4"
S: * QUOTA "#user/alice" (MESSAGE 42 1000)
S: * QUOTA "!partition/sda4" (STORAGE 104 10923847)
S: G0002 OK Getquotaroot complete
```

4.1.3. SETQUOTA

Arguments: quota root

list of resource limits

Responses: untagged responses: QUOTA

Result: OK - setquota completed
NO - setquota error: can't set that data
BAD - command unknown or arguments invalid

Note that unlike other command/responses/response codes defined in this document, support for SETQUOTA command requires the server to advertise "QUOTASET" capability.

The SETQUOTA command takes the name of a mailbox quota root and a list of resource limits. The resource limits for the named quota root are changed to be the specified limits. Any previous resource limits for the named quota root are discarded.

If the named quota root did not previously exist, an implementation may optionally create it and change the quota roots for any number of existing mailboxes in an implementation-defined manner.

If the implementation chooses to change the quota roots for some existing mailboxes such changes SHOULD be announced with untagged QUOTA responses.

Example:

```
S: * CAPABILITY [...] QUOTA QUOTASET QUOTA=RES-STORAGE QUOTA=RES-
MESSAGE [...]
```

```
[...]
C: S0000 GETQUOTA "#user/alice"
S: * QUOTA "#user/alice" (STORAGE 54 111 MESSAGE 42 1000)
S: S0000 OK Getquota completed
C: S0001 SETQUOTA "#user/alice" (STORAGE 510)
S: * QUOTA "#user/alice" (STORAGE 58 512)

// The server has rounded the STORAGE quota limit requested to the
nearest 512 blocks of 1024 octects, or else another client has
performed a near simultaneous SETQUOTA, using a limit of 512.

S: S0001 OK Rounded quota
C: S0002 SETQUOTA "!partition/sda4" (STORAGE 99999999)
S: * QUOTA "!partition/sda4" (STORAGE 104 10923847)

// The server has not changed the quota, since this is a
filesystem limit, and cannot be changed.  The QUOTA response here
is entirely optional.

S: S0002 NO Cannot change system limit
```

4.1.4. New STATUS attributes

DELETED and DELETED-STORAGE status data items allow to estimate the amount of resource freed by an EXPUNGE on a mailbox.

DELETED status data item requests the server to return the number of messages with \Deleted flag set. DELETED status data item is only required to be implemented when the server advertises QUOTA=RES-MESSAGE capability.

DELETED-STORAGE status data item requests the server to return the amount of storage space that can be reclaimed by performing EXPUNGE on the mailbox. The server SHOULD return the exact value, however it is recognized that the server may have to do non-trivial amount of work to calculate it. If the calculation of the exact value would take a long time, the server MAY instead return the sum of RFC822.SIZEs of messages with the \Deleted flag set. DELETED-STORAGE status data item is only required to be implemented when the server advertises QUOTA=RES-STORAGE capability.

Example:

```
S: * CAPABILITY [...] QUOTA QUOTA=RES-STORAGE QUOTA-RES-MESSAGE
[...]
[...]
C: S0003 STATUS INBOX (MESSAGES DELETED DELETED-STORAGE)
S: * STATUS INBOX (MESSAGES 12 DELETED 4 DELETED-STORAGE 8)
```

```
// 12 messages, 4 of which would be deleted when an EXPUNGE
happens.

S: S0003 OK Status complete.
```

4.2. Responses

The following responses may be sent by the server.

4.2.1. QUOTA

```
Data: quota root name
list of resource names, usages, and limits
```

This response occurs as a result of a GETQUOTA or GETQUOTAROOT command. The first string is the name of the quota root for which this quota applies.

The name is followed by a S-expression format list of the resource usage and limits of the quota root. The list contains zero or more triplets. Each triplet contains a resource name, the current usage of the resource, and the resource limit.

Resources not named in the list are not limited in the quota root. Thus, an empty list means there are no administrative resource limits in the quota root.

Example: S: * QUOTA "" (STORAGE 10 512)

4.2.2. QUOTAROOT

```
Data: mailbox name
zero or more quota root names
```

This response occurs as a result of a GETQUOTAROOT command. The first string is the mailbox and the remaining strings are the names of the quota roots for the mailbox.

Example:

S: * QUOTAROOT INBOX ""

S: * QUOTAROOT comp.mail.mime

4.3. Response Codes

4.3.1. OVERQUOTA

OVERQUOTA response code SHOULD be returned in the tagged NO response to an APPEND/COPY/MOVE when the addition of the message(s) puts the target mailbox over any one of its quota limits.

Example:

```
S: C: A003 APPEND Drafts (\Seen $MDNSent) {310}
S: + Ready for literal data
C: Date: Mon, 7 Feb 1994 21:52:25 -0800 (PST)
C: From: Fred Foobar <foobar@Blurdybloop.COM>
C: Subject: afternoon meeting
C: To: mooch@owatagu.siam.edu
C: Message-Id: <B27397-0100000@Blurdybloop.COM>
C: MIME-Version: 1.0
C: Content-Type: TEXT/PLAIN; CHARSET=US-ASCII
C:
C: Hello Joe, do you think we can meet at 3:30 tomorrow?
C:
S: A003 NO [OVERQUOTA] APPEND Failed
```

The OVERQUOTA response code MAY also be returned in an untagged NO response when a mailbox exceeds soft quota. Such responses have 2 forms. If it is followed by a tag, the tag refers to the command that caused this (such as APPEND or COPY) and the OVERQUOTA response code applies to the target mailbox specified by such command. If the OVERQUOTA response code is not followed by the tag, this means that an external event (e.g. LMTP delivery or APPEND/COPY in another IMAP connection) caused this event and the event applies to the currently selected mailbox. In particular, this means that such OVERQUOTA response codes MUST NOT be returned if there is no mailbox selected or if a mailbox other than the currently selected one exceeds soft quota.

Example:

```
S: C: A003 APPEND Drafts (\Seen $MDNSent) {310}
S: + Ready for literal data
C: Date: Mon, 7 Feb 1994 21:52:25 -0800 (PST)
C: From: Fred Foobar <foobar@Blurdybloop.COM>
C: Subject: afternoon meeting
C: To: mooch@owatagu.siam.edu
C: Message-Id: <B27397-0100000@Blurdybloop.COM>
C: MIME-Version: 1.0
C: Content-Type: TEXT/PLAIN; CHARSET=US-ASCII
```

```
C:
C: Hello Joe, do you think we can meet at 3:30 tomorrow?
C:
S: * NO [OVERQUOTA A003] Soft quota has been exceeded
S: A003 OK [APPENDUID 38505 3955] APPEND completed
```

5. Resource Type Definitions

The following resource types are defined in this memo. A server supporting a resource type MUST advertise this as a CAPABILITY with a name consisting of the resource name prefixed by "QUOTA=RES-". A server MAY support mupltiple resource types, and MUST advertise all resource types it supports.

5.1. STORAGE

The physical space estimate, in units of 1024 octets, of the mailboxes governed by the quota root. This MAY not be the same as the sum of the RFC822.SIZE of the messages. Some implementations MAY include metadata sizes for the messages and mailboxes, other implementations MAY store messages in such a way that the physical space used is smaller, for example due to use of compression. Additional messages might not increase the usage. Client MUST NOT use the usage figure for anything other than informational purposes, for example, they MUST NOT refuse to APPEND a message if the limit less the usage is smaller than the RFC822.SIZE divided by 1024 of the message, but it MAY warn about such condition.

The usage figure may change as a result of performing actions not associated with adding new messages to the mailbox, such as SEARCH, since this may increase the amount of metadata included in the calculations.

When the server supports this resource type, it MUST also support DELETED-STORAGE status data item.

Support for this resource MUST be indicated by the server by advertising the CAPABILITY "QUOTA=RES-STORAGE".

A resource named the same was also given as an example in RFC2087 [RFC2087]. This document provides a more precise definition.

5.2. MESSAGE

The number of messages stored within the mailboxes governed by the quota root. This MUST be an exact number, however, clients MUST NOT assume that a change in the usage indicates a change in the number of

messages available, since the quota root may include mailboxes the client has no access to.

When the server supports this resource type, it MUST also support DELETED status data item.

Support for this resource MUST be indicated by the server by advertising the CAPABILITY "QUOTA=RES-MESSAGE".

A resource named the same was also given as an example in RFC2087 [RFC2087]. This document provides a more precise definition.

5.3. MAILBOX

The number of mailboxes governed by the quota root. This MUST be an exact number, however, clients MUST NOT assume that a change in the usage indicates a change in the number of mailboxes, since the quota root may include mailboxes the client has no access to.

Support for this resource MUST be indicated by the server by advertising the CAPABILITY "QUOTA=RES-MAILBOX".

5.4. ANNOTATION-STORAGE

[[CREF1: Bron to check whether this is a sensible description and whether it is needed at all:]] The maximum size of all annotations [RFC5257], in units of 1024 octets, associated with all messages in the mailboxes governed by the quota root.

Support for this resource MUST be indicated by the server by advertising the CAPABILITY "QUOTA=RES-ANNOTATION-STORAGE".

6. Interaction with IMAP ACL extension (RFC 4314)

This section lists [RFC4314] rights required to execute quota related commands when both RFC 4314 and this document are implemented.

Operations\Rights	l	r	s	w	i	c	x	t	e	a	Any	Non
GETQUOTA												+
GETQUOTAROOT		*										*
SETQUOTA										+		

See Section 4 of RFC 4314 for conventions used in this table.

[[CREF2: The above table needs to be reviewed based on feedback from existing and planned implementations.]]

7. Formal syntax

The following syntax specification uses the Augmented Backus-Naur Form (ABNF) notation as specified in [ABNF].

Non-terminals referenced but not defined below are as defined by IMAP4 [RFC3501].

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.

```
getquota           =  "GETQUOTA" SP quota-root-name

getquotaroot       =  "GETQUOTAROOT" SP mailbox

quota-list         =  "(" quota-resource *(SP quota-resource) ")"

quota-resource     =  resource-name SP resource-usage SP resource-
                      limit

quota-response     =  "QUOTA" SP quota-root-name SP quota-list

quotaroot-response =  "QUOTAROOT" SP mailbox *(SP quota-root-name)

setquota           =  "SETQUOTA" SP quota-root-name SP setquota-list

setquota-list      =  "(" [setquota-resource *(SP setquota-resource)]
                      ")"

setquota-resource  =  resource-name SP resource-limit

quota-root-name    =  astring

resource-limit     =  number64

resource-name      =  "STORAGE" / "MESSAGE" / "MAILBOX" /
                      "ANNOTATION-STORAGE" / resource-name-vnd /
                      resource-name-ext

resource-name-vnd  =  "V-" atom
                      ;; Vendor specific, must be registered with IANA.
                      ;; The "V-" prefix should be followed by a domain
                      name
```

```
                      ;; under vendor's control.

   resource-name-ext  =  atom
                      ;; Not starting with V- and defined
                      ;; in a Standard Track or Experimental RFC

   resource-names =    "(" [resource-name *(SP resource-name)] ")"

   resource-usage     =  number64
                      ;; must be less than corresponding resource-limit

   capability-quota   =  capa-quota-res / "QUOTASET"
                      ;; One or more capa-quota-res must be returned.
                      ;; Also "QUOTASET" can optionally be returned.

   capa-quota-res     =  "QUOTA=RES-" resource-name

   status-att         =/  "DELETED" / "DELETED-STORAGE"
                      ;; DELETED status data item MUST be supported
                      ;; when "QUOTA=RES-MESSAGE" capability is
                      ;; advertised.
                      ;; DELETED-STORAGE status data item MUST be
                      ;; supported when "QUOTA=RES-STORAGE" capability
                      ;; is advertised.

   status-att-val     =/  status-att-deleted /
                      status-att-deleted-storage

   status-att-deleted =/  "DELETED" SP number
                      ;; DELETED status data item MUST be supported
                      ;; when "QUOTA=RES-MESSAGE" capability is
                      ;; advertised.

   status-att-deleted-storage =/  "DELETED-STORAGE" SP number64
                      ;; DELETED-STORAGE status data item MUST be
                      ;; supported when "QUOTA=RES-STORAGE" capability
                      ;; is advertised.

   resp-text-code     =/  "OVERQUOTA" [SP tag]

   number64           =  1*DIGIT
                      ;; Unsigned 63-bit integer.
                      ;; (0 <= n <= 9,223,372,036,854,775,807)
```

8. Security Considerations

Implementors should be careful to make sure the implementation of these commands does not violate the site's security policy. The resource usage of other users is likely to be considered confidential information and should not be divulged to unauthorized persons.

9. 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/imap4-capabilities

IANA is requested to update definition of the QUOTA extension to point to this document.

IANA is also requested to create a new registry for IMAP quota resource types. Registration policy for this registry is "Specification Required". When registering a new quota resource type, the registrant need to provide the following: Name of the quota resource type, Author/Change Controller name and email address, short description, extra (if any) required and optional IMAP commands/ responses, and a reference to a specification that describes the quota resource type in more details.

This document includes initial registrations for the following IMAP quota resource type: STORAGE (Section 5.1), MESSAGE (Section 5.2), MAILBOX (Section 5.3) and "ANNOTATION-STORAGE" (Section 5.4). See details below.

IANA is requested to reserve the prefix "QUOTA=RES-" in the IMAP4 capabilities registry and add a pointer to this document and to the IMAP quota resource type registry established above.

IANA is requested to reserve all other capabilities starting with "QUOTA=" prefix for future IETF stream standard track or experimental extensions to this document.

9.1. Registrations of IMAP Quota Resource Types

Name of the quota resource type: STORAGE

Author: Alexey Melnikov <alexey.melnikov@isode.com>

Change Controller: IESG <iesg@ietf.org>

Description: The physical space estimate, in units of 1024 octets, of the mailboxes governed by the quota root.

Extra required IMAP commands/responses: DELETED-STORAGE STATUS request data item and response data item

Extra optional IMAP commands/responses: N/A

Reference: Section 5.1 of RFCXXXX

Name of the quota resource type: MESSAGE

Author: Alexey Melnikov <alexey.melnikov@isode.com>

Change Controller: IESG <iesg@ietf.org>

Description: The number of messages stored within the mailboxes governed by the quota root.

Extra required IMAP commands/responses: DELETED STATUS request data item and response data item

Extra optional IMAP commands/responses: N/A

Reference: Section 5.2 of RFCXXXX

Name of the quota resource type: MAILBOX

Author: Alexey Melnikov <alexey.melnikov@isode.com>

Change Controller: IESG <iesg@ietf.org>

Description: The number of mailboxes governed by the quota root.

Extra required IMAP commands/responses: N/A

Extra optional IMAP commands/responses: N/A

Reference: Section 5.3 of RFCXXXX

Name of the quota resource type:

Author: Alexey Melnikov <alexey.melnikov@isode.com>

Change Controller: IESG <iesg@ietf.org>

Description: The maximum size of all annotations [RFC5257], in units of 1024 octets, associated with all messages in the mailboxes

governed by the quota root. [[CREF3: Recheck against the final description of "ANNOTATION-STORAGE".]]

Extra required IMAP commands/responses: N/A

Extra optional IMAP commands/responses: N/A

Reference: Section 5.4 of RFCXXXX

10. Open Issues

'"OVERQUOTA" SP tag' form has syntactic issues, as "tag" allows for "]", which is not allowed in response codes. Should we drop this variant or change IMAP4rev2 to disallow "]" in tags?

11. Contributors

Dave Cridland wrote lots of text in an earlier draft that became the basis for this document.

12. Acknowledgments

Editors of this document would like to thank the following people who provided useful comments or participated in discussions that lead to this update to RFC 2087:
John Myers,
Cyrus Daboo,
Lyndon Nerenberg

This document is a revision of RFC 2087. It borrows a lot of text from RFC 2087. Thus work of the RFC 2087 author John Myers is appreciated.

13. Changes since RFC 2087

This document is a revision of RFC 2087. It tries to clarify meaning of different terms used by RFC 2087. It also provides more examples, gives guidance on allowed server behaviour, defines IANA registry for quota resource types and provides initial registrations for 3 of them.

When compared with RFC 2087, this document defines two more commonly used resource type, adds optional OVERQUOTA response code and defines two extra STATUS data items ("DELETED" and "DELETED-STORAGE") that must be implemented. For extensibility quota usage and quota limits are now 63 bit unsigned integers.

14. References

14.1. Normative References

[ABNF] Crocker, D., Ed. and P. Overell, Ed., "Augmented BNF for Syntax Specifications: ABNF", RFC 5234, January 2008.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC3501] Crispin, M., "INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1", RFC 3501, DOI 10.17487/RFC3501, March 2003, <https://www.rfc-editor.org/info/rfc3501>.

[RFC4314] Melnikov, A., "IMAP4 Access Control List (ACL) Extension", RFC 4314, DOI 10.17487/RFC4314, December 2005, <https://www.rfc-editor.org/info/rfc4314>.

[RFC5257] Daboo, C. and R. Gellens, "Internet Message Access Protocol - ANNOTATE Extension", RFC 5257, DOI 10.17487/RFC5257, June 2008, <https://www.rfc-editor.org/info/rfc5257>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

14.2. Informative References

[RFC2087] Myers, J., "IMAP4 QUOTA extension", RFC 2087, DOI 10.17487/RFC2087, January 1997, <https://www.rfc-editor.org/info/rfc2087>.

Author's Address

Alexey Melnikov
Isode Limited

Email: alexey.melnikov@isode.com
URI: https://www.isode.com

JMAP
Internet-Draft
Intended status: Standards Track
Expires: 29 July 2021

N.M. Jenkins, Ed.
Fastmail
M. Douglass, Ed.
Spherical Cow Group

25 January 2021

JMAP for Calendars
draft-ietf-jmap-calendars-05

Abstract

This document specifies a data model for synchronizing calendar data with a server using JMAP.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 29 July 2021.

Table of Contents

1. Introduction

JMAP ([RFC8620] - JSON Meta Application Protocol) is a generic protocol for synchronizing data, such as mail, calendars or contacts, between a client and a server. It is optimized for mobile and web environments, and aims to provide a consistent interface to different data types.

This specification defines a data model for synchronizing calendar data between a client and a server using JMAP. The data model is designed to allow a server to provide consistent access to the same data via CalDAV [RFC4791] as well as JMAP, however the functionality offered over the two protocols may differ. Unlike CalDAV, this specification does not define access to tasks or journal entries (VTODO or VJOURNAL iCalendar components in CalDAV).

1.1. Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

Type signatures, examples, and property descriptions in this document follow the conventions established in Section 1.1 of [RFC8620]. Data types defined in the core specification are also used in this document.

1.2. The LocalDate Data Type

Where "LocalDate" is given as a type, it means a string in the same format as "Date" (see [RFC8620], Section 1.4), but with the "time-offset" omitted from the end. The interpretation in absolute time depends upon the time zone for the event, which may not be a fixed offset (for example when daylight saving time occurs). For example, "2014-10-30T14:12:00".

1.3. Terminology

The same terminology is used in this document as in the core JMAP specification, see [RFC8620], Section 1.6.

The terms ParticipantIdentity, Calendar, CalendarEvent, and CalendarEventNotification (with these specific capitalizations) are used to refer to the data types defined in this document and instances of those data types.

1.4. Data Model Overview

An Account (see [RFC8620], Section 1.6.2) with support for the calendar data model contains zero or more Calendar objects, which is a named collection of CalendarEvents. Calendars can also provide defaults, such as alerts and a color to apply to events in the calendar. Clients commonly let users toggle visibility of events belonging to a particular calendar on/off. Servers may allow an event to belong to multiple Calendars within an account.

A CalendarEvent is a representation of an event or recurring series of events in JSEvent [I-D.ietf-calext-jscalendar] format. Simple clients may ask the server to expand recurrences for them within a specific time period, and optionally convert times into UTC so they do not have to handle time zone conversion. More full-featured clients will want to access the full event information and handle recurrence expansion and time zone conversion locally.

CalendarEventNotification objects keep track of the history of changes made to a calendar by other users, allowing calendar clients to notify the user of changes to their schedule.

The ParticipantIdentity data type represents the identities of the current user within an Account, which determines which events the user is a participant of and possibly their permissions related to that event.

In servers with support for JMAP Sharing [RFC XXX], data may be shared with other users. Sharing permissions are managed per calendar. For example, an individual may have separate calendars for personal and work activities, with both contributing to their free-busy availability, but only the work calendar shared in its entirety with colleagues. Principals may also represent schedulable entities, such as a meeting room.

Users can normally subscribe to any calendar to which they have access. This indicates the user wants this calendar to appear in their regular list of calendars. The separate "isVisible" property stores whether the user would currently like to view the events in a subscribed calendar.

1.4.1. UIDs and CalendarEvent Ids

Each CalendarEvent has a "uid" property ([I-D.ietf-calext-jscalendar], Section 4.1.2), which is a globally unique identifier that identifies the same event in different Accounts, or different instances of the same recurring event within an Account.

An Account MUST NOT contain more than one CalendarEvent with the same uid unless all of the CalendarEvent objects have distinct, non-null values for their "recurrenceId" property. (This situation occurs if the principal is added to one or more specific instances of a recurring event without being invited to the whole series.)

Each CalendarEvent also has an id, which is scoped to the JMAP Account and used for referencing it in JMAP methods. There is no necessary link between the uid property and the CalendarEvent's id. CalendarEvents with the same uid in different Accounts MAY have different ids.

1.5. Addition to the Capabilities Object

The capabilities object is returned as part of the JMAP Session object; see [RFC8620], Section 2. This document defines two additional capability URIs.

1.5.1. urn:ietf:params:jmap:calendars

This represents support for the Calendar, CalendarEvent, CalendarEventNotification, and ParticipantIdentity data types and associated API methods. The value of this property in the JMAP Session capabilities property is an empty object.

The value of this property in an account's accountCapabilities property is an object that MUST contain the following information on server capabilities and permissions for that account:

* *shareesActAs*: "String" This MUST be one of:
  - "self" - sharees act as themselves when using calendars in this account.
  - "secretary"- sharees act as the principal to which this account belongs.
* *maxCalendarsPerEvent*: "UnsignedInt|null" The maximum number of Calendars (see Section XXX) that can be can assigned to a single CalendarEvent object (see Section XXX). This MUST be an integer >= 1, or null for no limit (or rather, the limit is always the number of Calendars in the account).
* *minDateTime*: "LocalDate" The earliest date-time the server is willing to accept for any date stored in a CalendarEvent.
* *maxDateTime*: "LocalDate" The latest date-time the server is willing to accept for any date stored in a CalendarEvent.
* *maxExpandedQueryDuration*: "Duration" The maximum duration the user may query over when asking the server to expand recurrences.
* *maxParticipantsPerEvent*: "Number|null" The maximum number of participants a single event may have, or null for no limit.

* *mayCreateCalendar*: "Boolean" If true, the user may create a calendar in this account.

1.5.2. urn:ietf:params:jmap:principals:availability

Represents support for the Principal/getAvailability method. Any account with this capability MUST also have the "urn:ietf:params:jmap:principals" capability (see [RFC XXX]).

The value of this property in the JMAP Session capabilities property is an empty object.

The value of this property in an account's accountCapabilities property is an object that MUST contain the following information on server capabilities and permissions for that account:

* *maxAvailabilityDuration*: The maximum duration over which the server is prepared to calculate availability in a single call (see Section XXX).

2. Principals and Sharing

For systems that also support JMAP Sharing [RFC XXX], the calendars capability is used to indicate that this principal may be used with calendaring. A new method is defined to allow users to query availability when scehduling events.

2.1. Principal Capability urn:ietf:params:jmap:calendars

A "urn:ietf:params:jmap:calendars" property is added to the Principal "capabilities" object, the value of which is an object with the following properties:

* *accountId*: "Id|null" Id of Account with the "urn:ietf:params:jmap:calendars" capability that contains the calendar data for this principal, or null if none (e.g. the Principal is a group just used for permissions management), or the user does not have access to any data in the account (with the exception of free/busy, which is governed by the mayGetAvailability property).

* *account*: "Account|null" The JMAP Account object corresponding to the accountId, null if none.

* *mayGetAvailability*: "Boolean" May the user call the "Principal/getAvailability" method with this Principal?

* *sendTo*: "String[String]|null" If this principal may be added as a participant to an event, this is the map of methods for adding it, in the same format as Participant#sendTo in JSEvent (see [I-D.ietf-calext-jscalendar], Section 4.4.5).

2.2. Principal/getAvailability

This method calculates the availability of the principal for scheduling within a requested time period. It takes the following arguments:

* *accountId*: "Id" The id of the account to use.

* *id*: "Id" The id of the Principal to calculate availability for.

* *utcStart*: "UTCDate" The start time (inclusive) of the period for which to return availability.

* *utcEnd*: "UTCDate" The end time (exclusive) of the period for which to return availability.

* *showDetails*: "Boolean" If true, event details will be returned if the user has permission to view them.

* *eventProperties*: "String[]|null" A list of properties to include in any JSEvent object returned. If "null", all properties of the event will be returned. Otherwise, only properties with names in the given list will be returned.

The server will first find all relevant events, expanding any recurring events. Relevant events are ones where all of the following is true:

* The principal is subscribed to the calendar.

* Either the calendar belongs to the principal or the calendar account's "shareesActAs" property is "self".

* The "includeInAvailability" property of the calendar for the principal is "all" or "attending".

* The user has the "mayReadFreeBusy" permission for the calendar.

* The event finishes after the "utcStart" argument and starts before the "utcEnd" argument.

* The event's "privacy" property is not "secret".

* The "freeBusyStatus" property of the event is "busy" (or omitted, as this is the default).

* The "status" property of the event is not "cancelled".

* If the "includeInAvailability" property of the calendar is "attending", then the principal is a participant of the event, and has a "participationStatus" of "accepted" or "tentative".

If an event is in more than one calendar, it is relevant if all of the above are true for any one calendar that it is in.

The server then generates a BusyPeriod object for each of these events. A *BusyPeriod* object has the following properties:

* *utcStart*: "UTCDate" The start time (inclusive) of the period this represents.

* *utcEnd*: "UTCDate" The end time (exclusive) of the period this represents.

* *busyStatus*: "String" (optional, default "unavailable") This MUST be one of

  - "confirmed": The event status is "confirmed".

  - "tentative": The event status is "tentative".

  - "unavailable": The principal is not available for scheduling at this time for any other reason.

* *event*: "JSEvent|null" The JSEvent representation of the event, or null if any of the following are true:

  - The "showDetails" argument is false.

  - The "privacy" property of the event is "private".

  - The user does not have the "mayReadItems" permission for any of the calendars the event is in.

  If an eventProperties argument was given, any properties in the JSEvent that are not in the eventProperties list are removed from the returned representation.

The server MAY also generate BusyPeriod objects based on other information it has about the principal’s availability, such as office hours.

Finally, the server MUST merge and split BusyPeriod objects where the "event" property is null, such that none of them overlap and either there is a gap in time between any two objects (the utcEnd of one does not equal the utcStart of another) or those objects have a different busyStatus property. If there are overlapping BusyPeriod time ranges with different "busyStatus" properties the server MUST choose the value in the following order: confirmed > unavailable > tentative.

The response has the following argument:

* *list*: "BusyPeriod[]" The list of BusyPeriod objects calculated as described above.

The following additional errors may be returned instead of the "Principal/getAvailability" response:

"notFound": No principal with this id exists, or the user does not have permission to see that this principal exists.

"forbidden": The user does not have permission to query this principal's availability.

"tooLarge": The duration between utcStart an utcEnd is longer than the server is willing to calculate availability for.

"rateLimit": Too many availability requests have been made recently and the user is being rate limited. It may work to try again later.

3. Participant Identities

A ParticipantIdentity stores information about a URI that represents the user within that account in an event's participants. It has the following properties:

* *id*: "Id" (immutable; server-set) The id of the ParticipantIdentity.

* *name*: "String" (default: "") The display name of the participant to use when adding this participant to an event, e.g. "Joe Bloggs".

* *sendTo*: "String[String]" Represents methods by which the participant may receive invitations and updates to an event.

  The keys in the property value are the available methods and MUST only contain ASCII alphanumeric characters (A-Za-z0-9). The value is a URI for the method specified in the key.

A participant in an event corresponds to a ParticipantIdentity if any of the method/uri pairs in the sendTo property of the participant are identical to a method/uri pair in the sendTo property of the identity.

The following JMAP methods are supported.

3.1. ParticipantIdentity/get

This is a standard "/get" method as described in [RFC8620], Section 5.1. The _ids_ argument may be "null" to fetch all at once.

3.2. ParticipantIdentity/changes

This is a standard "/changes" method as described in [RFC8620], Section 5.2.

3.3. ParticipantIdentity/set

This is a standard "/set" method as described in [RFC8620], Section 5.3. The server MAY restrict the uri values the user may claim, for example only allowing "mailto:" URIs with email addresses that belong to the user. A standard "forbidden" error is returned to reject non-permissible changes.

4. Calendars

A Calendar is a named collection of events. All events are associated with one, and only one, calendar.

A *Calendar* object has the following properties:

* *id*: "Id" (immutable; server-set) The id of the calendar.

* *role*: "String|null" (default: null) Denotes the calendar has a special purpose. This MUST be one of the following:

  - "inbox": This is the principal's default calendar; when the principal is invited to an event, this is the calendar to which it will be added by the server. There MUST NOT be more than one calendar with this role in an account.

  - "templates": This calendar holds templates for creating new events. All events in this calendar MUST have the "isDraft" property set to true. Clients should not show this as a regular calendar to users, but may offer users to create new events by copying one of the events in here.

* *name*: "String" The user-visible name of the calendar. This may be any UTF-8 string of at least 1 character in length and maximum 255 octets in size.

* *description*: "String|null" (default: null) An optional longer-form description of the calendar, to provide context in shared environments where users need more than just the name.

* *color*: "String|null" (default: null) A color to be used when displaying events associated with the calendar.

  If not null, the value MUST be a case-insensitive color name taken from the set of names defined in Section 4.3 of CSS Color Module Level 3 COLORS (https://www.w3.org/TR/css-color-3/), or an RGB value in hexadecimal notation, as defined in Section 4.2.1 of CSS Color Module Level 3.

The color SHOULD have sufficient contrast to be used as text on a white backgroun
d.

* *sortOrder*: "UnsignedInt" (default: 0) Defines the sort order of calendars when presented in the client's UI, so it is consistent between devices. The number MUST be an integer in the range 0 <= sortOrder < 2^(31.)

  A calendar with a lower order should be displayed before a calendar with a higher order in any list of calendars in the client's UI. Calendars with equal order SHOULD be sorted in alphabetical order by name. The sorting should take into account locale-specific character order convention.

* *isSubscribed*: "Boolean" Has the user indicated they wish to see this Calendar in their client? This SHOULD default to false for Calendars in shared accounts the user has access to and true for any new Calendars created by the user themself.

  If false, the calendar should only be displayed when the user explicitly requests it or to offer it for the user to subscribe to.

* *isVisible*: "Boolean" (default: true) Should the calendar's events be displayed to the user at the moment? Clients MUST ignore this property if isSubscribed is false. If an event is in multiple calendars, it should be displayed if isVisible is true for any of those calendars.

* *includeInAvailability*: "String" (default: all) Should the calendar's events be used as part of availability calculation? This MUST be one of:

  - "all": all events are considered.

  - "attending": events the user is a confirmed or tentative participant of are considered.

  - "none": all events are ignored (but may be considered if also in another calendar).

* *defaultAlertsWithTime*: "Id[Alert]|null" (default: null) A map of alert ids to Alert objects (see [I-D.ietf-calext-jscalendar], Section 4.5.2) to apply for events where "showWithoutTime" is false and "useDefaultAlerts" is true. Ids MUST be unique across all default alerts in the account, including those in other calendars; a UUID is recommended.

* *defaultAlertsWithoutTime*: "Id[Alert]|null" (default: null) A map of alert ids to Alert objects (see [I-D.ietf-calext-jscalendar], Section 4.5.2) to apply for events where "showWithoutTime" is true and "useDefaultAlerts" is true. Ids MUST be unique across all default alerts in the account, including those in other calendars; a UUID is recommended.

* *timeZone*: "String|null" (default: null) The time zone to use for events without a time zone when the server needs to resolve them into absolute time, e.g., for alerts or availability calculation. The value MUST be a time zone id from the IANA Time Zone Database TZDB (https://www.iana.org/time-zones). If "null", the timeZone of the account's associated Principal will be used. Clients SHOULD use this as the default for new events in this calendar if set.

* *shareWith*: "Id[CalendarRights]|null" (default: null) A map of Principal id to rights for principals this calendar is shared with. The principal to which this calendar belongs MUST NOT be in this set. This is null if the user requesting the object does not have the mayAdmin right, or if the calendar is not shared with anyone. May be modified only if the user has the mayAdmin right. The account id for the principals may be found in the "urn:ietf:params:jmap:principals:owner" capability of the Account to which the calendar belongs.

* *myRights*: "CalendarRights" (server-set) The set of access rights the user has in relation to this Calendar. If any event is in multiple calendars, the user has the following rights:

  - The user may fetch the event if they have the mayReadItems right on any calendar the event is in.

- The user may remove an event from a calendar (by modifying the event's "calendarIds" property) if the user has the appropriate permission for that calendar.

- The user may make other changes to the event if they have the right to do so in _all_ calendars to which the event belongs.

A *CalendarRights* object has the following properties:

* *mayReadFreeBusy*: "Boolean" The user may read the free-busy information for this calendar as part of a call to Principal/getAvailability (see Section XXX).

* *mayReadItems*: "Boolean" The user may fetch the events in this calendar.

* *mayAddItems*: "Boolean" The user may create new events on this calendar or move events to this calendar. For recurring events, they may add an override to add an occurrence, or remove an existing override that is excluding an occurrence.

* *mayUpdatePrivate*: "Boolean" The user may modify the following properties on all events in the calendar. If the shareesActAs account capability is "self", these properties MUST all be stored per-user, and changes do not affect any other user of the calendar. If shareesActAs is "secretary", the values are shared between all users.

  - keywords

  - color

  - freeBusyStatus

  - useDefaultAlerts

  - alerts

  The user may also modify the above on a per-occurrence basis for recurring events.

* *mayRSVP*: "Boolean" The user may modify the "participationStatus", "participationComment", "expectReply", "scheduleAgent", "scheduleSequence", and "scheduleUpdated" properties of any Participant object that corresponds to one of the user's ParticipantIdentity objects in the account.

If the event has its "mayInviteSelf" property set to true (see Section XXX), then the user may also add a new Participant to the event with a sendTo property that is the same as the sendTo property of one of the user's ParticipantIdentity objects in the account. The roles property of the participant MUST only contain "attendee".

If the event has its "mayInviteOthers" property set to true (see Section XXX) and there is an existing Participant in the event corresponding to one of the user's ParticipantIdentity objects in the account, then the user may also add new participants. The roles property of any new participant MUST only contain "attendee".

The user may also do all of the above on a per-occurrence basis for recurring events.

* *mayUpdateOwn*: "Boolean" The user may modify an existing event on this calendar if either they are the owner of the event or the event has no owner.

* *mayUpdateAll*: "Boolean" The user may modify all existing events on this calendar.

* *mayRemoveOwn*: "Boolean" The user may delete an event or remove it from this calendar if either they are the owner of the event or the event has no owner. For recurring events, they may add an override to remove an occurrence.

* *mayRemoveAll*: "Boolean" The user may delete any event or remove it from this calendar. For recurring events, they may add an override to remove an occurrence.

* *mayAdmin*: "Boolean" The user may modify sharing for this calendar.

* *mayDelete*: "Boolean" (server-set) The user may delete the calendar itself. This property MUST be false if the account to which this calendar belongs has the _isReadOnly_ property set to true.

The user is an *owner* for an event if the CalendarEvent object has a "participants" property, and one of the Participant objects both:

a) Has the "owner" role.
b) Corresponds to one of the user's ParticipantIdentity objects in the account.

An event has no owner if its participants property is null or omitted.

4.1. Calendar/get

This is a standard "/get" method as described in [RFC8620], Section 5.1. The _ids_ argument may be "null" to fetch all at once.

If mayReadFreeBusy is the only permission the user has, the calendar MUST NOT be returned in Calendar/get and Calendar/query; it must behave as though it did not exist. The data is just used as part of Principal/getAvailability.

4.2. Calendar/changes

This is a standard "/changes" method as described in [RFC8620], Section 5.2.

4.3. Calendar/set

This is a standard "/set" method as described in [RFC8620], Section 5.3 but with the following additional request argument:

* *onDestroyRemoveEvents*: "Boolean" (default: false)

If false, any attempt to destroy a Calendar that still has CalendarEvents in it will be rejected with a "calendarHasEvent" SetError. If true, any CalendarEvents that were in the Calendar will be removed from it, and if in no other Calendars they will be destroyed. This SHOULD NOT send scheduling messages to participants or create CalendarEventNotification objects.

The "role" and "shareWith" properties may only be set by users that have the mayAdmin right. The value is shared across all users, although users without the mayAdmin right cannot see the value.

When modifying the shareWith property, the user cannot give a right to a principal if the principal did not already have that right and the user making the change also does not have that right. Any attempt to do so must be rejected with a "forbidden" SetError.

Users can subscribe or unsubscribe to a calendar by setting the "isSubscribed" property. The server MAY forbid users from subscribing to certain calendars even though they have permission to see them, rejecting the update with a "forbidden" SetError.

The "timeZone", "includeInAvailability", "defaultAlertsWithoutTime" and "defaultAlertsWithTime" properties are stored per-user if the calendar account's "shareesActAs" capability is "self", and may be set by any user who is subscribed to the calendar. Otherwise, these properties are shared, and may only be set by users that have the mayAdmin right.

The following properties may be set by anyone who is subscribed to the calendar and are all stored per-user:

* name
* color
* sortOrder
* isVisible

These properties are initially inherited from the owner's copy of the calendar, but if set by a sharee that user gets their own copy of the property; it does not change for any other principals. If the value of the property in the owner's calendar changes after this, it does not overwrite the sharee's value.

The following extra SetError types are defined:

For "destroy":

* *calendarHasEvent*: The Calendar has at least one CalendarEvent assigned to it, and the "onDestroyRemoveEvents" argument was false.

5. Calendar Events

A *CalendarEvent* object contains information about an event, or recurring series of events, that takes place at a particular time. It is a JSEvent object, as defined in [I-D.ietf-calext-jscalendar], with the following additional properties:

* *id*: "Id" The id of the CalendarEvent. This property is immutable. The id uniquely identifies a JSEvent with a particular "uid" and "recurrenceId" within a particular account.
* *calendarIds*: "Id[Boolean]" The set of Calendar ids this event belongs to. An event MUST belong to one or more Calendars at all times (until it is destroyed). The set is represented as an object, with each key being a _Calendar id_. The value for each key in the object MUST be "true".

* *isDraft*: "Boolean" If true, this event is to be considered a draft. The server will not send any scheduling messages to participants or send push notifications for alerts. This may only be set to true upon creation. Once set to false, the value cannot be updated to true. This property MUST NOT appear in "recurrenceOverrides".

* *utcStart*: "UTCDate" For simple clients that do not or cannot implement time zone support. Clients should only use this if also asking the server to expand recurrences, as you cannot accurately expand a recurrence without the original time zone.

  This property is calculated at fetch time by the server. Time zones are political and they can and do change at any time. Fetching exactly the same property again may return a different results if the time zone data has been updated on the server. Time zone data changes are not considered "updates" to the event.

  If set, server will convert to the event's current time zone using its current time zone data and store the local time.

  This is not included by default and must be requested explicitly.

  Floating events (events without a time zone) will be interpreted as per the time zone given as a CalendarEvent/get argument.

  Note that it is not possible to accurately calculate the expansion of recurrence rules or recurrence overrides with the utcStart property rather than the local start time. Even simple recurrences such as "repeat weekly" may cross a daylight-savings boundary and end up at a different UTC time. Clients that wish to use "utcStart" are RECOMMENDED to request the server expand recurrences (see Section XXX).

* *utcEnd*: "UTCDate" The server calculates the end time in UTC from the start/timeZone/duration properties of the event. This is not included by default and must be requested explicitly. Like utcStart, this is calculated at fetch time if requested and may change due to time zone data changes. Floating events will be interpreted as per the time zone given as a CalendarEvent/get argument.

CalendarEvent objects MUST NOT have a "method" property as this is only used when representing iTIP [RFC5546] scheduling messages, not events in a data store.

5.1. Additional JSCalendar properties

This document defines three new JSCalendar properties.

5.1.1. mayInviteSelf

Type: "Boolean" (default: false)

If "true", any user that has access to the event may add themselves to it as a participant with the "attendee" role. This property MUST NOT be altered in the recurrenceOverrides; it may only be set on the master object.

5.1.2. mayInviteOthers

Type: "Boolean" (default: false)

If "true", any current participant with the "attendee" role may add new participants with the "attendee" role to the event. This property MUST NOT be altered in the recurrenceOverrides; it may only be set on the master object.

5.1.3. hideAttendees

Type: "Boolean" (default: false)

If "true", only the owners of the event may see the full set of participants. Other sharees of the event may only see the owners and themselves. This property MUST NOT be altered in the recurrenceOverrides; it may only be set on the master object.

5.2. Attachments

The Link object, as defined in [I-D.ietf-calext-jscalendar] Section 4.2.7, with a "rel" property equal to "enclosure" is used to represent attachments. Instead of mandating an "href" property, clients may set a "blobId" property instead to reference a blob of binary data in the account, as per [RFC8620] Section 6.

The server MUST translate this to an embedded "data:" URL [RFC2397] when sending the event to a system that cannot access the blob. Servers that support CalDAV access to the same data are recommended to expose these files as managed attachments [?@RFC8607].

5.3. Per-user properties

In shared calendars where the account's "shareesActAs" capability is "self", the following properties MUST be stored per-user:

* keywords
* color
* freeBusyStatus
* useDefaultAlerts
* alerts

The user may also modify these properties on a per-occurrence basis for recurring events; again, these MUST be stored per-user.

When writing only per-user properties, the "updated" property MUST also be stored just for that user. When fetching the "updated" property, the value to return is whichever is later of the per-user updated time or the updated time of the master event.

5.4. Recurring events

Events may recur, in which case they represent multiple occurrences or instances. The data store will either contain a single master event, containing a recurrence rule and/or recurrence overrides; or, a set of individual instances (when invited to specific occurrences only).

The client may ask the server to expand recurrences within a specific time range in "CalendarEvent/query". This will generate synthetic ids representing individual instances in the requested time range. The client can fetch and update the objects using these ids and the server will make the appropriate changes to the master event. Synthetic ids do not appear in "CalendarEvent/changes" responses; only the ids of events as actually stored on the server.

If the user is invited to specific instances then later added to the master event, "CalendarEvent/changes" will show the ids of all the individual instances being destroyed and the id for the master event being created.

5.5. Updating for "this-and-future"

When editing a recurring event, you can either update the master event (affecting all instances unless overriden) or update an override for a specific occurrence. To update all occurrences from a specific point onwards, there are therefore two options: split the event, or update the master and override all occurrences before the split point back to their original values.

5.5.1. Splitting an event

If the event is not scheduled (has no participants), the simplest thing to do is to duplicate the event, modifying the recurrence rules of the original so it finishes before the split point, and the duplicate so it starts at the split point. As per JSCalendar [I-D.ietf-calext-jscalendar] Section 4.1.3, a "next" and "first" relation MUST be set on the new objects respectively.

Splitting an event however is problematic in the case of a scheduled event, because the iTIP messages generated make it appear like two unrelated changes, which can be confusing.

5.5.2. Updating the master and overriding previous

For scheduled events, a better approach is to avoid splitting and instead update the master event with the new property value for "this and future", then create overrides for all occurrences before the split point to restore the property to its previous value. Indeed, this may be the only option the user has permission to do if not an owner of the event.

Clients may choose to skip creating the overrides if the old data is not important, for example if the "alerts" property is being updated, it is probably not important to create overrides for events in the past with the alerts that have already fired.

5.6. CalendarEvent/get

This is a standard "/get" method as described in [RFC8620], Section 5.1, with three extra arguments:

* *recurrenceOverridesBefore*: "UTCDate|null" If given, only recurrence overrides with a recurrence id before this date (when translated into UTC) will be returned.

* *recurrenceOverridesAfter*: "UTCDate|null" If given, only recurrence overrides with a recurrence id on or after this date (when translated into UTC) will be returned.

* *reduceParticipants*: "Boolean" (default: false) If true, only participants with the "owner" role or corresponding to the user's participant identities will be returned in the "participants" property of the master event and any recurrence overrides. If false, all participants will be returned.

* *timeZone*: "String" (default "Etc/UTC") The time zone to use when calculating the utcStart/utcEnd property of floating events. This argument has no effect if those properties are not requested.

A CalendarEvent object is a JSEvent object so may have arbitrary properties. If the client makes a "CalendarEvent/get" call with a null or omitted "properties" argument, all properties defined on the JSEvent object in the store are returned, along with the "id", "calendarIds", and "isDraft" properties. The "utcStart" and "utcEnd" computed properties are only returned if explicitly requested. If either are requested, the "recurrenceOverrides" property MUST NOT be requested (recurrence overrides cannot be interpreted accurately with just the UTC times).

If specific properties are requested from the JSEvent and the property is not present on the object in the server's store, the server SHOULD return the default value if known for that property.

A requested id may represent a single instance of a recurring event if the client asked the server to expand recurrences in "CalendarEvent/query". In such a case, the server will resolve any overrides and set the appropriate "start" and "recurrenceId" properties on the CalendarEvent object returned to the client. The "recurrenceRule" and "recurrenceOverrides" properties MUST be returned as null if requested for such an event.

An event with the same uid/recurrenceId may appear in different accounts. Clients may coalesce the view of such events, but must be aware that the data may be different in the different accounts due to per-user properties, difference in permissions etc.

The "privacy" property of a JSEvent object allows the owner to override how sharees of the calendar see the event. If this is set to "private", when a sharee fetches the event the server MUST only return the basic time and metadata properties of the JSEvent object as specified in [I-D.ietf-calext-jscalendar], Section 4.4.3. If set to "secret", the server MUST behave as though the event does not exist for all users other than the owner.

This "hideAttendees" property of a JSEvent object allows the owner to reduce the visibility of sharees into the set of participants. If this is "true", when a non-owner sharee fetches the event, the server MUST only return participants with the "owner" role or corresponding to the user's participant identities.

5.7. CalendarEvent/changes

This is a standard "/changes" method as described in [RFC8620], Section 5.2.

5.8. CalendarEvent/set

This is a standard "/set" method as described in [RFC8620], Section 5.3, with the following extra argument:

* *sendSchedulingMessages*: "Boolean" (default: false) If true then any changes to scheduled events will be sent to all the participants (if the user is an owner of the event) or back to the owners (otherwise). If false, the changes only affect this account and no scheduling messages will be sent.

For recurring events, an id may represent the master event or a specific instance. When the id for a specific instance is given, the server MUST process an update as an update to the recurrence override for that instance on the master event, and a destroy as removing just that instance.

Clients MUST NOT send an update/destroy to both the master event and a specific instance in a single "/set" request; the result of this is undefined.

Servers MUST enforce the user's permissions as returned in the "myRights" property of the Calendar objects and reject changes with a "forbidden" SetError if not allowed.

The "privacy" property MUST NOT be set to anything other than "public" (the default) for events in a calendar that does not belong to the user (e.g. a shared team calendar). The server MUST reject this with an "invalidProperties" SetError.

The server MUST reject attempts to add events with a "participants" property where none of the participants correspond to one of the calendar's participant identities with a "forbidden" SetError.

If omitted on create, the server MUST set the following properties to an appropriate value:

* @type

* uid

* created

The "updated" property MUST be set to the current time by the server whenever an event is created or updated. If the client tries to set a value for this property it is not an error, but it MUST be overridden and replaced with the server's time.

When updating an event, if all of: * a non per-user property has been changed; and * the server is the source of the event (see Section XXX); and * the "sequence" property is not explicitly set in the update, or the given value is less than or equal to the current "sequence" value on the server; then the server MUST increment the "sequence" value by one.

The "created" property MUST NOT be updated after creation. The "method" property MUST NOT be set. Any attempt to do these is rejected with a standard "invalidProperties" SetError.

If "utcStart" is set, this is translated into a "start" property using the server's current time zone information. It MUST NOT be set in addition to a "start" property and it cannot be set inside "recurrenceOverrides"; this MUST be rejected with an "invalidProperties" SetError.

Similarly, the "utcEnd" property is translated into a "duration" property if set. It MUST NOT be set in addition to a "duration" property and it cannot be set inside "recurrenceOverrides"; this MUST be rejected with an "invalidProperties" SetError.

The server does not automatically reset the "partipationStatus" or "expectReply" properties of a Participant when changing other event details. Clients should either be intelligent about whether the change necessitates resending RSVP requests, or ask the user whether to send them.

The server MAY enforce that all events have an owner, for example in team calendars. If the user tries to create an event without participants in such a calendar, the server MUST automatically add a participant with the "owner" role corresponding to one of the user's ParticipantIdentities (see Section XXX).

When creating an event with participants, or adding participants to an event that previously did not have participants, the server MUST set the "replyTo" property of the event if not present. Clients SHOULD NOT set the replyTo property for events when the user adds participants; the server is better positioned to add all the methods it supports to receive replies.

5.8.1. Patching

The JMAP "/set" method allows you to update an object by sending a patch, rather than having to supply the whole object. When doing so, care must be taken if updating a property of a CalendarEvent where the value is itself a PatchObject, e.g. inside "localizations" or "recurrenceOverrides". In particular, you cannot add a property with value "null" to the CalendarEvent using a direct patch on that property, as this is interpreted instead as a patch to remove the property. This is more easily understood with an example. Suppose you have a CalendarEvent object like so:

```
{
  "id": "123",
  "title": "FooBar team meeting",
  "start": "2018-01-08T09:00:00",
  "recurrenceRules": [{
   "@type": "RecurrenceRule",
   "frequency": "weekly"
  }],
  "replyTo": {
   "imip": "mailto:6489-4f14-a57f-c1@schedule.example.com"
  },
  "participants": {
   "dG9tQGZvb2Jhci5xlLmNvbQ": {
     "@type": "Participant",
     "name": "Tom",
     "email": "tom@foobar.example.com",
     "sendTo": {
       "imip": "mailto:6489-4f14-a57f-c1@calendar.example.com"
     },
     "participationStatus": "accepted",
     "roles": {
       "attendee": true
     }
   },
   "em9lQGZvb2GFtcGxlLmNvbQ": {
     "@type": "Participant",
     "name": "Zoe",
     "email": "zoe@foobar.example.com",
     "sendTo": {
       "imip": "mailto:zoe@foobar.example.com"
     },
     "participationStatus": "accepted",
     "roles": {
       "owner": true,
       "attendee": true,
       "chair": true
     }
   },
   "recurrenceOverrides": {
     "2018-03-08T09:00:00": {
       "start": "2018-03-08T10:00:00",
       "participants/dG9tQGZvb2Jhci5xlLmNvbQ/participationStatus":
                                                          "declined"
     }
   }
  }
}
```

In this example, Tom is normally going to the weekly meeting but has declined the occurrence on 2018-03-08, which starts an hour later than normal. Now, if Zoe too were to decline that meeting, she could update the event by just sending a patch like so:

```
[[ "CalendarEvent/set", {
  "accountId": "ue150411c",
  "update": {
    "123": {
      "recurrenceOverrides/2018-03-08T09:00:00/
          participants~1em9lQGZvb2GFtcGxlLmNvbQ~1participationStatus":
              "declined"
    }
  }
}, "0" ]]
```

This patches the "2018-03-08T09:00:00" PatchObject in recurrenceOverrides so that it ends up like this:

```
"recurrenceOverrides": {
  "2018-03-08T09:00:00": {
    "start": "2018-03-08T10:00:00",
    "participants/dG9tQGZvb2Jhci5xlLmNvbQ/participationStatus":
                                                        "declined",
    "participants/em9lQGZvb2GFtcGxlLmNvbQ/participationStatus":
                                                        "declined"
  }
}
```

Now if Tom were to change his mind and remove his declined status override (thus meaning he is attending, as inherited from the top-level event), he might remove his patch from the overrides like so:

```
[[ "CalendarEvent/set", {
  "accountId": "ue150411c",
  "update": {
    "123": {
      "recurrenceOverrides/2018-03-08T09:00:00/
          participants~1dG9tQGZvb2Jhci5xlLmNvbQ~1participationStatus": null
    }
  }
}, "0" ]]
```

However, if you instead want to remove Tom from this instance altogether, you could not send this patch:

```
[[ "CalendarEvent/set", {
  "accountId": "ue150411c",
  "update": {
    "123": {
      "recurrenceOverrides/2018-03-08T09:00:00/
          participants~1dG9tQGZvb2Jhci5xlLmNvbQ": null
    }
  }
}, "0" ]]
```

This would mean remove the "participants/dG9tQGZvb2Jhci5xlLmNvbQ" property at path "recurrenceOverrides" -> "2018-03-08T09:00:00" inside the object; but this doesn't exist. We actually we want to add this property and make it map to "null". The client must instead send the full object that contains the property mapping to "null", like so:

```
[[ "CalendarEvent/set", {
  "accountId": "ue150411c",
  "update": {
    "123": {
      "recurrenceOverrides/2018-03-08T09:00:00": {
        "start": "2018-03-08T10:00:00",
        "participants/em9lQGZvb2GFtcGxlLmNvbQ/participationStatus":
                                                    "declined"
        "participants/dG9tQGZvb2Jhci5xlLmNvbQ": null
      }
    }
  }
}, "0" ]]
```

5.8.2. Sending invitations and responses

If "sendSchedulingMessages" is true, the server MUST send appropriate iTIP [RFC5546] scheduling messages after successfuly creating, updating or destroying a calendar event.

When determining which scheduling messages to send, the server must first establish whether it is the _source_ of the event. The server is the source if it will receive messages sent to any of the methods specified in the "replyTo" property of the event.

Messages are only sent to participants with a "scheduleAgent" property set to "server" or omitted. If the effective "scheduleAgent" property is changed:

* to "server" from something else: send messages to this participant as though the event had just been created.

* from "server" to something else: send messages to this participant as though the event had just been destroyed.

* any other change: do not send any messages to this participant.

The server may send the scheduling message via any of the methods defined on the sendTo property of a participant (if the server is the source) or the replyTo property of the event (otherwise) that it supports. If no supported methods are available, the server MUST reject the change with a "noSupportedScheduleMethods" SetError.

If the server is the source of the event it MUST NOT send messages to any participant corresponding to a ParticipantIdentitity in that account (see Section XXX).

If sending via iMIP [RFC6047], the server MAY choose to only send updates it deems "essential" to avoid flooding the recipient's email with changes they do not care about. For example, changes to the participationStatus of another participant, or changes to events solely in the past may be omitted.

5.8.2.1. REQUEST

When the server is the source for the event, a REQUEST message ([RFC5546], Section 3.2.2) is sent to all current participants if:

* The event is being created.

* Any non per-user property (see Section XXX) is updated on the event (including adding/removing participants), except if just modifying the recurrenceOverrides such that CANCEL messages are generated (see the next section).

Note, if the only change is adding an additional instance (not generated by the event's recurrence rule) to the recurrenceOverrides, this MAY be handled via sending an ADD message ([RFC5546], Section 3.2.4) for the single instance rather than a REQUEST message for the master. However, for interoperability reasons this is not recommended due to poor support in the wild for this type of message.

The server MUST ensure participants are only sent information about recurrence instances they are added to when sending scheduling messages for recurring events. If the participant is not invited to the master recurring event but only individual instances, scheduling messages MUST be sent for just those expanded occurrences individually. If a participant is invited to a recurring event, but removed via a recurrence override from a particular instance, any scheduling messages to this participant MUST return the instance as "excluded" (if it matches a recurrence rule for the event) or omit the instance entirely (otherwise).

If the event's "hideAttendees" property is set to "true", the recipient MUST be the only attendee in the message; all others are omitted.

5.8.2.2. CANCEL

When the server is the source for the event, a CANCEL message ([RFC5546], Section 3.2.5) is sent if:

* A participant is removed from either the master event or a single instance (the message is only sent to this participant; remaining participants will get a REQUEST, as described above).

* The event is destroyed.

* An exclusion is added to recurrenceOverrides to remove an instance generated by the event's recurrence rule.

* An additional instance (not generated by the event's recurrence rule) is removed from the recurrenceOverrides.

In each of the latter 3 cases, the message is sent to all participants.

5.8.2.3. REPLY

When the server is _not_ the source for the event, a REPLY message ([RFC5546], Section 3.2.3) is sent for any participant corresponding to one of the user's ParticipantIdentitities in the account if:

* The "participationStatus" property of the participant is changed.

* The event is destroyed and the participationStatus was not "needs-action".

* The event is created and the participationStatus is not "needs-action".

* An exclusion is added to recurrenceOverrides to remove an instance generated by the event's recurrence rule.

* An exclusion is removed from recurrenceOverrides (this is presumed to be the client undoing the deletion of a single instance).

* An instance not generated by the event's recurrence rule is removed from the recurrenceOverrides.

* An instance not generated by the event's recurrence rule is added to the recurrenceOverrides (this is presumed to be the client undoing the deletion of a single instance).

A reply is not sent when deleting an event where the current status is "needs-action" as if a junk calendar event gets added by an automated system, the user MUST be able to delete the event without sending a reply.

5.9. CalendarEvent/copy

This is a standard "/copy" method as described in [RFC8620], Section 5.4.

5.10. CalendarEvent/query

This is a standard "/query" method as described in [RFC8620], Section 5.5, with two extra arguments:

* *expandRecurrences*: "Boolean" (default: false) If true, the server will expand any recurring event. If true, the filter MUST be just a FilterCondition (not a FilterOperator) and MUST include both a before and after property. This ensures the server is not asked to return an infinite number of results.

* *timeZone*: "String" The time zone for before/after filter conditions (default: "Etc/UTC")

If expandRecurrences is true, a separate id will be returned for each instance of a recurring event that matches the query. This synthetic id is opaque to the client, but allows the server to resolve the id + recurrence id for "/get" and "/set" operations. Otherwise, a single id will be returned for matching recurring events that represents the entire event.

There is no necessary correspondence between the ids of different instances of the same expanded event.

The following additional error may be returned instead of the "CalendarEvent/query" response:

"cannotCalculateOccurrences": the server cannot expand a recurrence required to return the results for this query.

5.10.1. Filtering

A *FilterCondition* object has the following properties:

* *inCalendars*: "Id[]|null" A list of calendar ids. An event must be in ANY of these calendars to match the condition.

* *after*: "LocalDate|null" The end of the event, or any recurrence of the event, in the time zone given as the timeZone argument, must be after this date to match the condition.

* *before*: "LocalDate|null" The start of the event, or any recurrence of the event, in the time zone given as the timeZone argument, must be before this date to match the condition.

* *text*: "String|null" Looks for the text in the _title_, _description_, _locations_ (matching name/description), _participants_ (matching name/email) and any other textual properties of the event or any recurrence of the event.

* *title*: "String|null" Looks for the text in the _title_ property of the event, or the overridden _title_ property of a recurrence.

* *description*: "String|null" Looks for the text in the _description_ property of the event, or the overridden _description_ property of a recurrence.

* *location*: "String|null" Looks for the text in the _locations_ property of the event (matching name/description of a location), or the overridden _locations_ property of a recurrence.

* *owner*: "String|null" Looks for the text in the name or email fields of a participant in the _participants_ property of the event, or the overridden _participants_ property of a recurrence, where the participant has a role of "owner".

* *attendee*: "String|null" Looks for the text in the name or email fields of a participant in the _participants_ property of the event, or the overridden _participants_ property of a recurrence, where the participant has a role of "attendee".

* *participationStatus*: Must match. If owner/attendee condition, status must be of that participant. Otherwise any.

* *uid*: "String" The uid of the event is exactly the given string.

If expandRecurrences is true, all conditions must match against the same instance of a recurring event for the instance to match. If expandRecurrences is false, all conditions must match, but they may each match any instance of the event.

If zero properties are specified on the FilterCondition, the condition MUST always evaluate to "true". If multiple properties are specified, ALL must apply for the condition to be "true" (it is equivalent to splitting the object into one-property conditions and making them all the child of an AND filter operator).

The exact semantics for matching "String" fields is *deliberately not defined* to allow for flexibility in indexing implementation, subject to the following:

* Text SHOULD be matched in a case-insensitive manner.

* Text contained in either (but matched) single or double quotes SHOULD be treated as a *phrase search*, that is a match is required for that exact sequence of words, excluding the surrounding quotation marks. Use "\"", "\'" and "\\" to match a literal """, "'" and "\" respectively in a phrase.

* Outside of a phrase, white-space SHOULD be treated as dividing separate tokens that may be searched for separately in the event, but MUST all be present for the event to match the filter.

* Tokens MAY be matched on a whole-word basis using stemming (so for example a text search for "bus" would match "buses" but not "business").

5.10.2. Sorting

The following properties MUST be supported for sorting:

* start

* uid

* recurrenceId

The following properties SHOULD be supported for sorting:

* created

* updated

5.11. CalendarEvent/queryChanges

This is a standard "/queryChanges" method as described in [RFC8620], Section 5.6.

5.12. Examples

TODO: Add example of how to get event by uid: query uid=foo and backref. Return multiple with recurrenceId set (user invited to specific instances of recurring event).

6. Alerts

Alerts may be specified on events as described in [I-D.ietf-calext-jscalendar], Section 4.5.

Alerts MUST only be triggered for events in calendars where the user is subscribed and either the user owns the calendar or the calendar account's "shareesActAs" capability is "self".

When an alert with an "email" action is triggered, the server MUST send an email to the user to notify them of the event. The contents of the email is implementation specific. Clients MUST NOT perform an action for these alerts.

When an alert with a "display" action is triggered, clients SHOULD display an alert in a platform-appropriate manner to the user to remind them of the event. Clients with a full offline cache of events may choose to calculate when alerts should trigger locally. Alternatively, they can subscribe to push events from the server.

6.1. Default alerts

If the "useDefaultAlerts" property of an event is true, the alerts are taken from the "defaultAlertsWithTime" or "defaultAlertsWithoutTime" property of all Calendars the event is in, as described in Section XXX, rather than the "alerts" property of the CalendarEvent.

When using default alerts, the "alerts" property of the event is ignored except for the following:

* The "acknowledged" time for an alert is stored here when a default alert for the event is dismissed. The id of the alert MUST be the same as the id of the default alert in the calendar. See Section XXX on acknowledging alerts.

* If an alert has a relatedTo property where the parent is the id of one of the calendar default alerts, it is processed as normal and not ignored. This is to support snoozing default alerts; see Section XXX.

6.2. Acknowledging an alert

To dismiss an alert, clients set the "acknowledged" property of the Alert object to the current date-time. If the alert was a calendar default, it may need to be added to the event at this point in order to acknowledge it. When other clients fetch the updated CalendarEvent they SHOULD automatically dismiss or suppress duplicate alerts (alerts with the same alert id that triggered on or before the "acknowledged" date-time) and alerts that have been removed from the event.

Setting the "acknowledged" property MUST NOT create a new recurrence override. For a recurring calendar object, the "acknowledged" property of the parent object MUST be updated, unless the alert is already overridden in the "recurrenceOverrides" property.

6.3. Snoozing an alert

Users may wish to dismiss an alert temporarily and have it come back after a specific period of time. To do this, clients MUST:

1. Acknowledge the alert as described in Section XXX.

2. Add a new alert to the event with an "AbsoluteTrigger" for the date-time the alert has been snoozed until. Add a "relatedTo" property to the new alert, setting the "parent" relation to point to the original alert. This MUST NOT create a new recurrence override; it is added to the same "alerts" property that contains the alert that was acknowledged in step 1.

When acknowledging a snoozed alert (i.e. one with a parent relatedTo pointing to the original alert), the client SHOULD delete the alert rather than setting the "acknowledged" property.

6.4. Push events

Servers that support the "urn:ietf:params:jmap:calendars" capability MUST support registering for the pseudo-type "CalendarAlert" in push subscriptions and event source connections, as described in [RFC8620], Sections 7.2 and 7.3.

If requested, a CalendarAlert notification will be pushed whenever an alert is triggered for the user. For Event Source connections, this notification is pushed as an event called "calendarAlert".

A *CalendarAlert* object has the following properties:

* *@type*: "String" This MUST be the string "CalendarAlert".
* *accountId*: "String" The account id for the calendar in which the alert triggered.
* *calendarEventId*: "String" The CalendarEvent id for the alert that triggered.
* *uid*: "String" The uid property of the CalendarEvent for the alert that triggered.
* *recurrenceId*: "String|null" The recurrenceId for the instance of the event for which this alert is being triggered, or "null" if the event is not recurring.
* *alertId*: "String" The id for the alert that triggered.

7. Calendar Event Notifications

The CalendarEventNotification data type records changes made by external entities to events in calendars the user is subscribed to. Notifications are stored in the same Account as the CalendarEvent that was changed.

Notifications are only created by the server; users cannot create them directly. Clients SHOULD present the list of notifications to the user and allow them to dismiss them. To dismiss a notification you use a standard "/set" call to destroy it.

The server SHOULD create a CalendarEventNotification whenever an event is added, updated or destroyed by another user or due to receiving an iTIP [RFC5546] or other scheduling message in a calendar this user is subscribed to. The server SHOULD NOT create notifications for events implicitly deleted due to the containing calendar being deleted.

The CalendarEventNotification does not have any per-user data. A single instance may therefore be maintained on the server for all sharees of the notification. The server need only keep track of which users have yet to destroy the notification.

7.1. Auto-deletion of Notifications

The server MAY limit the maximum number of notifications it will store for a user. When the limit is reached, any new notification will cause the previously oldest notification to be automatically deleted.

The server MAY coalesce events if appropriate, or remove events that it deems are no longer relevant or after a certain period of time. The server SHOULD automatically destroy a notification about an event if the user updates or destroys that event (e.g. if the user sends an RSVP for the event).

7.2. Object Properties

The *CalendarEventNotification* object has the following properties:

* *id*: "String" The id of the CalendarEventNotification.

* *created*: "UTCDate" The time this notification was created.

* *changedBy*: "Person" Who made the change.

  - *name*: "String" The name of the person who made the change.

  - *email*: "String" The email of the person who made the change, or null if no email is available.

  - *principalId*: "String|null" The id of the calendar principal corresponding to the person who made the change, if any. This will be null if the change was due to receving an iTIP message.

* *comment*: "String|null" Comment sent along with the change by the user that made it. (e.g. COMMENT property in an iTIP message).

* *type*: "String" This MUST be one of

  - created

  - updated

  - destroyed

* *calendarEventId*: "String" The id of the CalendarEvent that this notification is about.

* *isDraft*: "Boolean" (created/updated only) Is this event a draft?

* *event*: "JSEvent" The data before the change (if updated or destroyed), or the data after creation (if created).

* *eventPatch*: "PatchObject" (updated only) A patch encoding the change between the data in the event property, and the data after the update.

To reduce data, if the change only affects a single instance of a recurring event, the server MAY set the event and eventPatch properties for the instance; the calendarEventId MUST still be for the master event.

7.3. CalendarEventNotification/get

This is a standard "/get" method as described in [RFC8620], Section 5.1.

7.4. CalendarEventNotification/changes

This is a standard "/changes" method as described in [RFC8620], Section 5.2.

7.5. CalendarEventNotification/set

This is a standard "/changes" method as described in [RFC8620], Section 5.3.

Only destroy is supported; any attempt to create/update MUST be rejected with a "forbidden" SetError.

7.6. CalendarEventNotification/query

This is a standard "/query" method as described in [RFC8620], Section 5.5.

7.6.1. Filtering

A *FilterCondition* object has the following properties:

* *after*: "UTCDate|null" The creation date must be on or after this date to match the condition.

* *before*: "UTCDate|null" The creation date must be before this date to match the condition.

* *type*: "String" The type property must be the same to match the condition.

* *calendarEventIds*: "Id[]|null" A list of event ids. The calendarEventId property of the notification must be in this list to match the condition.

7.6.2. Sorting

The "created" property MUST be supported for sorting.

7.7. CalendarEventNotification/queryChanges

This is a standard "/queryChanges" method as described in [RFC8620], Section 5.6.

8. Security Considerations

All security considerations of JMAP [RFC8620] and JSCalendar [I-D.ietf-calext-jscalendar] apply to this specification. Additional considerations specific to the data types and functionality introduced by this document are described in the following subsections.

8.1. Denial-of-service Expanding Recurrences

Recurrence rules can be crafted to occur as frequently as every second. Servers MUST be careful to not allow resources to be exhausted when expanding. Equally, rules can be generated that never create any occurrences at all. Servers MUST be careful to limit the work spent iterating in search of the next occurrence.

8.2. Privacy

TODO.

9. IANA Considerations

9.1. JMAP Capability Registration for "calendars"

IANA will register the "calendars" JMAP Capability as follows:

Capability Name: "urn:ietf:params:jmap:calendars"

Specification document: this document

Intended use: common

Change Controller: IETF

Security and privacy considerations: this document, Section XXX

9.2. JSCalendar Property Registrations

IANA will register the following additional properties in the JSCalendar Properties Registry.

9.2.1. id

Property Name: id

Property Type: "Id"

Property Context: JSEvent, JSTask

Intended Use: Reserved

9.2.2. calendarIds

Property Name: calendarIds

Property Type: "Id[Boolean]"

Property Context: JSEvent, JSTask

Intended Use: Reserved

9.2.3. isDraft

Property Name: isDraft

Property Type: "Boolean"

Property Context: JSEvent, JSTask

Intended Use: Reserved

9.2.4. utcStart

Property Name: utcStart

Property Type: "UTCDateTime"

Property Context: JSEvent, JSTask

Intended Use: Reserved

9.2.5. utcEnd

Property Name: utcEnd

Property Type: "UTCDateTime"

Property Context: JSEvent, JSTask

Intended Use: Reserved

9.2.6. mayInviteSelf

Property Name: mayInviteSelf

Property Type: "Boolean" (default: false)

Property Context: JSEvent, JSTask

Reference: This document, Section XXX.

Intended Use: Common

9.2.7. mayInviteOthers

Property Name: mayInviteOthers

Property Type: "Boolean" (default: false)

Property Context: JSEvent, JSTask

Reference: This document, Section XXX.

Intended Use: Common

9.2.8. hideAttendees

Property Name: hideAttendees

Property Type: "Boolean" (default: false)

Property Context: JSEvent, JSTask

Reference: This document, Section XXX.

Intended Use: Common

10. Normative References

[I-D.ietf-calext-jscalendar]
Jenkins, N. and R. Stepanek, "JSCalendar: A JSON representation of calendar data", Work in Progress, Internet-Draft, draft-ietf-calext-jscalendar-32, 15 October 2020, <https://tools.ietf.org/html/draft-ietf-calext-jscalendar-32>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC2397] Masinter, L., "The "data" URL scheme", RFC 2397, DOI 10.17487/RFC2397, August 1998, <https://www.rfc-editor.org/info/rfc2397>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8620] Jenkins, N. and C. Newman, "The JSON Meta Application Protocol (JMAP)", RFC 8620, DOI 10.17487/RFC8620, July 2019, <https://www.rfc-editor.org/info/rfc8620>.

11. Informative References

[RFC4791] Daboo, C., Desruisseaux, B., and L. Dusseault, "Calendaring Extensions to WebDAV (CalDAV)", RFC 4791, DOI 10.17487/RFC4791, March 2007, <https://www.rfc-editor.org/info/rfc4791>.

[RFC5546] Daboo, C., Ed., "iCalendar Transport-Independent Interoperability Protocol (iTIP)", RFC 5546, DOI 10.17487/RFC5546, December 2009, <https://www.rfc-editor.org/info/rfc5546>.

[RFC6047] Melnikov, A., Ed., "iCalendar Message-Based Interoperability Protocol (iMIP)", RFC 6047, DOI 10.17487/RFC6047, December 2010, <https://www.rfc-editor.org/info/rfc6047>.

Authors' Addresses

Neil Jenkins (editor)
Fastmail
PO Box 234, Collins St West

Melbourne VIC 8007
Australia

Email: neilj@fastmailteam.com
URI: https://www.fastmail.com

Michael Douglass (editor)
Spherical Cow Group
226 3rd Street
Troy, NY 12180
United States of America

Email: mdouglass@sphericalcowgroup.com
URI: http://sphericalcowgroup.com

JMAP
Internet-Draft
Intended status: Standards Track
Expires: June 17, 2021

R. Stepanek
FastMail
M. Loffredo
IIT-CNR

December 14, 2020

JSContact: A JSON representation of contact data
draft-ietf-jmap-jscontact-03

Abstract

This specification defines a data model and JSON representation of contact card information that can be used for data storage and exchange in address book or directory applications. It aims to be an alternative to the vCard data format and to be unambiguous, extendable and simple to process. In contrast to the JSON-based jCard format, it is not a direct mapping from the vCard data model and expands semantics where appropriate.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on June 17, 2021.

Table of Contents

1. Introduction

This document defines a data model for contact card data normally used in address book or directory applications and services. It aims to be an alternative to the vCard data format [RFC6350] and to provide a JSON-based standard representation of contact card data.

The key design considerations for this data model are as follows:

- Most of the initial set of attributes should be taken from the vCard data format [RFC6350] and extensions ([RFC6473], [RFC6474], [RFC6715], [RFC6869], [RFC8605]). The specification should add new attributes or value types, or not support existing ones, where appropriate. Conversion between the data formats need not fully preserve semantic meaning.

- The attributes of the cards data represented must be described as a simple key-value pair, reducing complexity of its representation.

- The data model should avoid all ambiguities and make it difficult to make mistakes during implementation.

- Extensions, such as new properties and components, MUST NOT lead to requiring an update to this document.

The representation of this data model is defined in the I-JSON format [RFC7493], which is a strict subset of the JavaScript Object Notation (JSON) Data Interchange Format [RFC8259]. Using JSON is mostly a pragmatic choice: its widespread use makes JSCard easier to adopt, and the availability of production-ready JSON implementations eliminates a whole category of parser-related interoperability issues.

1.1. Relation to the xCard and jCard formats

The xCard [RFC6351] and jCard [RFC7095] specifications define alternative representations for vCard data, in XML and JSON format respectively. Both explicitly aim to not change the underlying data model. Accordingly, they are regarded as equal to vCard in the context of this document.

1.2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

1.3. Vendor-specific Property Extensions and Values

Vendors MAY add additional properties to JSContact objects to support their custom features. The names of these properties MUST be prefixed with a domain name controlled by the vendor to avoid conflict, e.g. "example.com/customprop".

Some JSContact properties allow vendor-specific value extensions. If so, vendor-specific values MUST be prefixed with a domain name controlled by the vendor, e.g. "example.com/customrel".

Vendors are strongly encouraged to register any new property values or extensions that are useful to other systems as well, rather than using a vendor-specific prefix.

2. JSCard

MIME type: "application/jscontact+json;type=jscard"

A JSCard object stores information about a person, organization or company.

2.1. Metadata properties

2.1.1. uid

Type: "String" (mandatory).

An identifier, used to associate the object as the same across different systems, addressbooks and views. [RFC4122] describes a range of established algorithms to generate universally unique identifiers (UUID), and the random or pseudo-random version is recommended. For compatibility with [RFC6350] UIDs, implementations MUST accept both URI and free-form text.

2.1.2. prodId

Type: "String" (optional).

The identifier for the product that created the JSCard object.

2.1.3. updated

Type: "String" (optional).

The date and time when the data in this JSCard object was last modified. The timestamp MUST be formatted as specified in [RFC3339].

2.1.4. kind

Type: "String" (optional). The kind of the entity the Card represents.

The value MUST be either one of the following values, registered in a future RFC, or a vendor-specific value:

- "individual": a single person
- "org": an organization
- "location": a named location
- "device": a device, such as appliances, computers, or network elements
- "application": a software application

2.1.5. relatedTo

Type: "String[Relation]" (optional).

Relates the object to other JSCard objects. This is represented as a map of the URI (or single text value) of the related objects to a possibly empty set of relation types. The Relation object has the following properties:

- relation: "String[Boolean]" (optional, default: empty Object) Describes how the linked object is related to the linking object. The relation is defined as a set of relation types. If empty, the relationship between the two objects is unspecified. Keys in the set MUST be one of the RELATED property [RFC6350] type parameter values, or an IANA-registered value, or a vendor-specific value. The value for each key in the set MUST be true.

Note, the Relation object only has one property; it is specified as an object with a single property to allow for extension in the future.

2.2. Name and Organization properties

2.2.1. fullName

Type: "LocalizedString" (optional).

The full name (e.g. the personal name and surname of an individual, the name of an organization) of the entity represented by this card.

2.2.2. name

Type: "NameComponent[]" (optional).

The name components of the name of the entity represented by this JSCard. Name components SHOULD be ordered such that their values joined by whitespace produce a valid full name of this entity.

A NameComponent has the following properties:

- value: "String" (mandatory). The value of this name component.
- type: "String" (mandatory). The type of this name component. The value MUST be either one of the following values, registered in a future RFC, or a vendor-specific value:
  - "prefix". The value is a honorific title(s), e.g. "Mr", "Ms", "Dr".
  - "personal". The value is a personal name(s), also known as "first name", "given name".
  - "surname". The value is a surname, also known as "last name", "family name".
  - "additional". The value is an additional name, also known as "middle name".
  - "suffix". The value is a honorific suffix, e.g. "B.A.", "Esq.".
  - "nickname". The value is a nickname.

2.2.3. organization

Type: "LocalizedString[]" (optional).

The company or organization name and units associated with this card. The first entry in the list names the organization, and any following entries name organizational units.

2.2.4. jobTitle

Type : "LocalizedString[]" (optional).

The job title(s) or functional position(s) of the entity represented by this card.

2.2.5. role

Type : "LocalizedString[]" (optional).

The role(s), function(s) or part(s) played in a particular situation by the entity represented by this card. In contrast to a job title, the roles might differ for example in project contexts.

2.3. Contact and Resource properties

2.3.1. emails

Type: "Resource[]" (optional).

An array of Resource objects where the values are URLs in the "mailto" scheme [RFC6068] or free-text email addresses. The default value of the "type" property is "email". If set, the type MUST be "email" or "other".

2.3.2. phones

Type: "Resource[]" (optional).

An array of Resource objects where the values are URIs scheme or free-text phone numbers. Typical URI schemes are the [RFC3966] "tel" or [RFC3261] "sip" schemes, but any URI scheme is allowed. Types are:

- "voice" The number is for calling by voice.
- "fax" The number is for sending faxes.
- "pager" The number is for a pager or beeper.
- "other" The number is for some other purpose. A label property MAY be included to display next to the number to help the user identify its purpose.

2.3.3. online

Type: "Resource[]" (optional).

An array of Resource objects where the values are URIs or usernames associated with the card for online services. Types are:

- "uri" The value is a URI, e.g. a website link.
- "username" The value is a username associated with the entity represented by this card (e.g. for social media, or an IM client). A label property SHOULD be included to identify what service this is for. For compatibility between clients, this label SHOULD be the canonical service name, including capitalisation. e.g. "Twitter", "Facebook", "Skype", "GitHub", "XMPP".
- "other" The value is something else not covered by the above categories. A label property MAY be included to display next to the number to help the user identify its purpose.

2.3.4. preferredContactMethod

Type : "String" (optional)

Defines the preferred contact method or resource with additional information about this card. The value MUST be the property name of one of the Resource lists: "emails", "phones", "online", "other".

2.3.5. preferredContactLanguages

Type : "String[ContactLanguage[]]" (optional)

Defines the preferred languages for contacting the entity associated with this card. The keys in the object MUST be [RFC5646] language tags. The values are a (possibly empty) list of contact language preferences for this language. Also see the definition of the VCARD LANG property (Section 6.4.4., [RFC6350]).

A ContactLanguage object has the following properties:

- type: "String" (optional). Defines the context of this preference. This could be "work", "home" or another value.
- preference: "Number" (optional). Defines the preference order of this language for the context defined in the type property. If set, the property value MUST be between 1 and 100 (inclusive). Lower values correspond to a higher level of preference, with 1 being most preferred. If not set, the default MUST be to

interpret the language as the least preferred in its context. Preference orders SHOULD be unique across language for a specific type.

A valid ContactLanguage object MUST have at least one of its properties set.

2.4. Address and Location properties

2.4.1. addresses

Type: Address[] (optional).

An array of Address objects, containing physical locations. An Address object has the following properties:

- context: "String" (optional, default "other"). Specifies the context of the address information. The value MUST be either one of the following values, registered in a future RFC, or a vendor-specific value:
  - "private" An address of a residence.
  - "work" An address of a workplace.
  - "billing" An address to be used for billing.
  - "postal" An address to be used for delivering physical items.
  - "other" An address not covered by the above categories.
- label: "String" (optional). A label describing the value in more detail.
- fullAddress: "LocalizedString" (optional). The complete address, excluding type and label. This property is mainly useful to represent addresses of which the individual address components are unknown, or to provide localized representations.
- street: "String" (optional). The street address. This MAY be multiple lines; newlines MUST be preserved.
- extension: "String" (optional) The extended address, such as an apartment or suite number, or care-of address.
- locality: "String" (optional). The city, town, village, post town, or other locality within which the street address may be found.

- region: "String" (optional). The province, such as a state, county, or canton within which the locality may be found.
- country: "String" (optional). The country name.
- postOfficeBox: "String" (optional) The post office box.
- postcode: "String" (optional). The postal code, post code, ZIP code or other short code associated with the address by the relevant country's postal system.
- countryCode: "String" (optional). The ISO-3166-1 country code.
- coordinates: "String" (optional) A [RFC5870] "geo:" URI for the address.
- timeZone: "String" (optional) Identifies the time zone this address is located in. This SHOULD be a time zone name registered in the IANA Time Zone Database [1]. Unknown time zone identifiers MAY be ignored by implementations.
- isPreferred: Boolean (optional, default: false). Whether this Address is the preferred for its type. This SHOULD only be one per type.

2.5. Additional properties

2.5.1. anniversaries

Type : Anniversary[] (optional).

Memorable dates and events for the entity represented by this card. An Anniversary object has the following properties:

- type: "String" (mandatory). Specifies the type of the anniversary. This RFC predefines the following types, but implementations MAY use additional values:
  - "birth": a birth day anniversary
  - "death": a death day anniversary
  - "other": an anniversary not covered by any of the known types.
- label: "String" (optional). A label describing the value in more detail, especially if the type property has value "other" (but MAY be included with any type).

- date: "String" (mandatory). The date of this anniversary, in the form "YYYY-MM-DD" (any part may be all 0s for unknown) or a [RFC3339] timestamp.

- place: Address (optional). An address associated with this anniversary, e.g. the place of birth or death.

2.5.2. personalInfo

Type: PersonalInformation[] (optional).

A list of personal information about the entity represented by this card. A PersonalInformation object has the following properties:

- type: "String" (mandatory). Specifies the type for this personal information. Allowed values are:

  * "expertise": a field of expertise or credential

  * "hobby": a hobby

  * "interest": an interest

  * "other": an information not covered by the above categories

- value: "String" (mandatory). The actual information. This generally is free-text, but future specifications MAY restrict allowed values depending on the type of this PersonalInformation.

- level: "String" (optional) Indicates the level of expertise, or engagement in hobby or interest. Allowed values are: "high", "medium" and "low".

2.5.3. notes

Type: "LocalizedString[]" (optional).

Arbitrary notes about the entity represented by this card.

2.5.4. categories

Type: "String[]" (optional). A list of free-text or URI categories that relate to the card.

2.6. Common JSCard types

2.6.1. LocalizedString

A LocalizedString object has the following properties:

- value: "String" (mandatory). The property value.
- language: "String" (optional). The [RFC5646] language tag of this value, if any.
- localizations: "String[String]" (optional). A map from [RFC5646] language tags to the value localized in that language.

2.6.2. Resource

A Resource object has the following properties:

- context: "String" (optional) Specifies the context in which to use this resource. Pre-defined values are:
  - "private": The resource may be used to contact the card holder in a private context.
  - "work": The resource may be used to contact the card holder in a professional context.
  - "other": The resource may be used to contact the card holder in some other context. A label property MAY be help to identify its purpose.
- type: "String" (optional). Specifies the property-specific variant of the resource. This MUST be taken from the set of allowed types specified in the respective contact method property.
- labels: "String[Boolean]" (optional). A set of labels that describe the value in more detail, especially if the type property has value "other" (but MAY be included with any type). The keys in the map define the label, the values MUST be "true".
- value: "String" (mandatory). The actual resource value, e.g. an email address or phone number.
- mediaType: "String" (optional). Used for properties with URI values. Provides the media type [RFC2046] of the resource identified by the URI.

- isPreferred: Boolean (optional, default: false). Whether this resource is the preferred for its type. This SHOULD only be one per type.

3. JSCardGroup

MIME type: "application/jscontact+json;type=jscardgroup"

A JSCardGroup object represents a named set of JSCards.

3.1. Properties

3.1.1. uid

Type : "String" (mandatory).

A globally unique identifier. The same requirements as for the JSCard uid property apply.

3.1.2. name

Type: "String" (optional).

The user-visible name for the group, e.g. "Friends". This may be any UTF-8 string of at least 1 character in length and maximum 255 octets in size. The same name may be used by two different groups.

3.1.3. cards

Type : "JSCard[]" (mandatory). The cards in the group. Implementations MUST preserve the order of list entries.

4. Implementation Status

NOTE: Please remove this section and the reference to [RFC7942] prior to publication as an RFC. This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [RFC7942]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist. According to [RFC7942], "this will allow reviewers and working groups to assign

due consideration to documents that have the benefit of running code, which may serve as evidence of valuable experimentation and feedback that have made the implemented protocols more mature. It is up to the individual working groups to use this information as they see fit".

4.1. IIT-CNR/Registro.it

- Responsible Organization: Institute of Informatics and Telematics of National Research Council (IIT-CNR)/Registro.it
- Location: https://rdap.pubtest.nic.it/ [2]
- Description: This implementation includes support for RDAP queries using data from the public test environment of .it ccTLD. The RDAP server does not implement any security policy because data returned by this server are only for experimental testing purposes. The RDAP server returns responses including JSCard in place of jCard when queries contain the parameter jscard=1.
- Level of Maturity: This is a "proof of concept" research implementation.
- Coverage: This implementation includes all of the features described in this specification.
- Contact Information: Mario Loffredo, mario.loffredo@iit.cnr.it

5. IANA Considerations

TBD

6. Security Considerations

TBD

7. References

7.1. Normative References

[RFC2046] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, DOI 10.17487/RFC2046, November 1996, <https://www.rfc-editor.org/info/rfc2046>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, DOI 10.17487/RFC4122, July 2005, <https://www.rfc-editor.org/info/rfc4122>.

[RFC5646] Phillips, A., Ed. and M. Davis, Ed., "Tags for Identifying Languages", BCP 47, RFC 5646, DOI 10.17487/RFC5646, September 2009, <https://www.rfc-editor.org/info/rfc5646>.

[RFC5870] Mayrhofer, A. and C. Spanring, "A Uniform Resource Identifier for Geographic Locations (’geo’ URI)", RFC 5870, DOI 10.17487/RFC5870, June 2010, <https://www.rfc-editor.org/info/rfc5870>.

[RFC6350] Perreault, S., "vCard Format Specification", RFC 6350, DOI 10.17487/RFC6350, August 2011, <https://www.rfc-editor.org/info/rfc6350>.

[RFC6351] Perreault, S., "xCard: vCard XML Representation", RFC 6351, DOI 10.17487/RFC6351, August 2011, <https://www.rfc-editor.org/info/rfc6351>.

[RFC7095] Kewisch, P., "jCard: The JSON Format for vCard", RFC 7095, DOI 10.17487/RFC7095, January 2014, <https://www.rfc-editor.org/info/rfc7095>.

[RFC7493] Bray, T., Ed., "The I-JSON Message Format", RFC 7493, DOI 10.17487/RFC7493, March 2015, <https://www.rfc-editor.org/info/rfc7493>.

[RFC7942] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", BCP 205, RFC 7942, DOI 10.17487/RFC7942, July 2016, <https://www.rfc-editor.org/info/rfc7942>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

7.2. Informative References

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, DOI 10.17487/RFC3261, June 2002, <https://www.rfc-editor.org/info/rfc3261>.

[RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002, <https://www.rfc-editor.org/info/rfc3339>.

[RFC3966] Schulzrinne, H., "The tel URI for Telephone Numbers", RFC 3966, DOI 10.17487/RFC3966, December 2004, <https://www.rfc-editor.org/info/rfc3966>.

[RFC6068] Duerst, M., Masinter, L., and J. Zawinski, "The 'mailto' URI Scheme", RFC 6068, DOI 10.17487/RFC6068, October 2010, <https://www.rfc-editor.org/info/rfc6068>.

[RFC6473] Saint-Andre, P., "vCard KIND:application", RFC 6473, DOI 10.17487/RFC6473, December 2011, <https://www.rfc-editor.org/info/rfc6473>.

[RFC6474] Li, K. and B. Leiba, "vCard Format Extensions: Place of Birth, Place and Date of Death", RFC 6474, DOI 10.17487/RFC6474, December 2011, <https://www.rfc-editor.org/info/rfc6474>.

[RFC6715] Cauchie, D., Leiba, B., and K. Li, "vCard Format Extensions: Representing vCard Extensions Defined by the Open Mobile Alliance (OMA) Converged Address Book (CAB) Group", RFC 6715, DOI 10.17487/RFC6715, August 2012, <https://www.rfc-editor.org/info/rfc6715>.

[RFC6869] Salgueiro, G., Clarke, J., and P. Saint-Andre, "vCard KIND:device", RFC 6869, DOI 10.17487/RFC6869, February 2013, <https://www.rfc-editor.org/info/rfc6869>.

[RFC8605] Hollenbeck, S. and R. Carney, "vCard Format Extensions: ICANN Extensions for the Registration Data Access Protocol (RDAP)", RFC 8605, DOI 10.17487/RFC8605, May 2019, <https://www.rfc-editor.org/info/rfc8605>.

7.3. URIs

[1] https://www.iana.org/time-zones

[2] https://rdap.pubtest.nic.it/

Authors' Addresses

Robert Stepanek
FastMail
PO Box 234, Collins St West
Melbourne, VIC 8007
Australia

Email: rsto@fastmailteam.com

Mario Loffredo
IIT-CNR
Via Moruzzi,1
Pisa, 56124
Italy

Email: mario.loffredo@iit.cnr.it

JMAP
Internet-Draft
Intended status: Standards Track
Expires: September 5, 2020

R. Cordier, Ed.
M. Bailly, Ed.
Linagora

March 4, 2020

JMAP for Quotas
draft-ietf-jmap-quotas-01

Abstract

This document specifies a data model for handling quotas on accounts with a server using JMAP.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 5, 2020.

Table of Contents

1. Introduction

JMAP ([RFC8620] - JSON Meta Application Protocol) is a generic protocol for synchronising data, such as mails, calendars or contacts, between a client and a server. It is optimised for mobile and web environments, and aims to provide a consistent interface to different data types.

This specification defines a data model for handling mail quotas over JMAP, allowing you to read and explain quota information.

This specification does not address quota administration, which should be handled by other means.

1.1. Notational conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

Type signatures, examples and property descriptions in this document follow the conventions established in section 1.1 of [RFC8620]. Data types defined in the core specification are also used in this document.

Servers MUST support all properties specified for the new data types defined in this document.

1.2. Terminology

1.2.1. Quota

A quota is a numeric upper limit that the server is enforcing. Quotas are applied to accounts.

1.3. Addition to the capabilities object

The capabilities object is returned as part of the JMAP Session object; see [RFC8620], section 2.

This document defines one additional capability URI.

1.3.1. urn::ietf::params::jmap::quota

This represents support for the Quota data type and associated API methods. The value of this property in the JMAP session capabilities property is an empty object.

The value of this property in an account's accountCapabilities property is an object that MUST contain the following information on server capabilities and permissions for that account:

- *quotaIds:* "Id[]" (default: "[]") A list of quota ids bound to that account, or "[]" if that account has no quota restrictions.

1.4. Push

Servers MUST support the JMAP push mechanisms, as specified in [RFC8620] Section 7, to receive notifications when the state changes for the Quota type defined in this specification.

2. Quota definition

The quota is an object that displays the limit set to an account usage as well as the current usage in regard to that limit.

2.1. The Scope Data Type

The *Scope* is a "String" from an enumeration defined list of values, handled by the server.

It explains the entities this value applies to. Some custom specifications might provide some additional values. If the client does not specify custom scope specifications in the "using" parameter of the request, the server should respond the JSON value "null", instead of answering a scope value that the client does not support. Standard values are:

- "account": Applies for this account
- "domain": All users of this domain share this part of the quota
- "global": All users of this mail server share this part of the quota

2.2. The ResourceType Data Type

The *ResourceType* is a "String" from an enumeration defined list of values, handled by the server.

A resource type is like an unit of measure for the quota usage. Some custom specifications might provide some additional values. If the client does not specify custom resource type specifications in the "using" parameter of the request, the server should respond the JSON value "null", instead of answering a resource type value that the client does not support. Standard values are:

- "count": The quota is measured in number of data type objects. For example, a quota can have a limit of 50 "Mail" objects.
- "size": The quota is measured in size. The default unit is in "bytes", but a server can decide of the unit it wants to use (like in "octets"). For example, a quota can have a limit of 25000 "bytes"

2.3. The Quota Object

The quota object MUST contain the following fields:

- *id*: "Id" The unique identifier for this object. It should respect the JMAP ID datatype defined in section 1.2 of [RFC8620]
- *resourceType*: "ResourceType" The resource type of the quota.

- *used*: "UnsignedInt" The current usage of the mailbox. Computation of this value is handled by the server.

- *limit*: "UnsignedInt" The hard limit set by this quota object. No more outgoing and ingoing messages should be allowed if we reach this limit. It should higher than the "warnLimit" and the "softLimit".

- *scope*: "Scope" The "Scope" of this quota.

- *name*: "String" The name of the quota object. Useful for managing quotas and use queries for searching.

- *datatypes*: "String[]" A list of all the data types values that are applying to this quota. This allows to assign quotas to separated or shared data types. This MAY include data types the client does not recognise. Clients MUST ignore any unknown data type in the list.

The quota object MAY contain the following field:

- *warnLimit*: "UnsignedInt|null" The warn limit set by this quota object. It can be used to send a warning to an user that he is going to reach the hard limit soon, but no action is taken. If set, it should be lower than the "softLimit" and the "limit".

- *softLimit*: "UnsignedInt|null" The soft limit set by this quota object. It can be used to block outgoing messages, but still allowing incoming messages. If set, it should be higher than the "warnLimit" but lower than the "limit".

- *description*: "String|null" Arbitrary free, human readable, description of this quota. Might be used to explain where the limit comes from and explain the entities this quota applies to.

2.4. Example

```
{
  "id": "2a06df0d-9865-4e74-a92f-74dcc814270e",
  "resourceType": "count",
  "used": 1056,
  "warnLimit": 1600,
  "softLimit": 1800,
  "limit": 2000,
  "scope": "account",
  "name": "bob@example.com",
  "description": "Personal account usage",
  "datatypes" : [ "Mail", "Calendar", "Contact" ]
}
```

2.5. Quota/get

Standard "/get" method as described in [RFC8620] section 5.1. The ids argument may be "null" to fetch all at once.

2.5.1. Example

Request fetching all quotas related to an account :

```
[[ "Quota/get", {
  "accountId": "u33084183",
  "ids": null
}, "0" ]]
```

With response :

```
            [[ "Quota/get", {
              "accountId": "u33084183",
              "state": "78540",
              "list": [{
                "id": "2a06df0d-9865-4e74-a92f-74dcc814270e",
                "resourceType": "count",
                "used": 1056,
                "warnLimit": 1600,
                "softLimit": 1800,
                "limit": 2000,
                "scope": "account",
                "name": "bob@example.com",
                "description": "Personal account usage",
                "datatypes" : [ "Mail", "Calendar", "Contact" ]
              }, {
                "id": "3b06df0e-3761-4s74-a92f-74dcc963501x",
                "resourceType": "size",
                ...
              }, ...],
              "notFound": []
            }, "0" ]]
```

2.6. Quota/changes

Standard "/changes" method as described in [RFC8620] section 5.2 but with one extra argument to the response:

- *updatedProperties*: "String[]|null" If only the "used" Quota properties has changed since the old state, this will be the list of properties that may have changed. If the server is unable to tell if only "used" has changed, it MUST just be null.

Since "used" frequently changes but other properties are generally only changed rarely, the server can help the client optimise data transfer by keeping track of changes to Quota usage separate from other state changes. The updatedProperties array may be used directly via a back-reference in a subsequent Quota/get call in the same request, so only these properties are returned if nothing else has changed.

Servers MAY decide to add other properties to the list that they judge changing frequently.

2.6.1. Example

Request:

```
                    [[ "Quota/changes", {
                      "accountId": "u33084183",
                      "sinceState": "10824",
                      "maxChanges": 20
                    }, "0" ],
                    [ "Quota/get", {
                      "accountId": "u33084183",
                      "#ids": {
                        "resultOf": "0",
                        "name": "Quota/changes",
                        "path": "/updated"
                      },
                      "#properties": {
                        "resultOf": "0",
                        "name": "Quota/changes",
                        "path": "/updatedProperties"
                      }
                    }, "1" ]]
```

Response:

```
        [[ "Quota/changes", {
          "accountId": "u33084183",
          "oldState": "10824",
          "newState": "10826",
          "hasMoreChanges": false,
          "created": [],
          "updated": ["2a06df0d-9865-4e74-a92f-74dcc814270e"],
          "destroyed": []
        }, "0" ],
        [ "Quota/get", {
          "accountId": "u33084183",
          "state": "10826",
          "list": [{
            "id": "2a06df0d-9865-4e74-a92f-74dcc814270e",
            "used": 1246
          }],
          "notFound": []
        }, "1" ]]
```

2.7. Quota/query

This is a standard "/query" method as described in [RFC8620], Section 5.5.

A *FilterCondition* object has the following properties, any of which may be omitted:

- *name*: "String" The Quota _name_ property contains the given string.
- *scopes*: "Scope[]" The Quota _scope_ property must be in this list to match the condition.
- *resourceTypes*: "ResourceType[]" The Quota _resourceType_ property must be in this list to match the condition.
- *datatypes*: "String[]" The Quota _datatypes_ property must contain the elements in this list to match the condition.

A Quota object matches the FilterCondition if and only if all of the given conditions match. If zero properties are specified, it is automatically true for all objects.

The following Quota properties MUST be supported for sorting:

- *name*
- *used*

2.8. Quota/queryChanges

This is a standard "/queryChanges" method as described in [RFC8620], Section 5.6.

3. Security considerations

All security considerations of JMAP ([RFC8620]) apply to this specification.

4. IANA Considerations

4.1. JMAP Capability Registration for "quota"

IANA will register the "quota" JMAP Capability as follows:

Capability Name: "urn:ietf:params:jmap:quota"

Specification document: this document

Intended use: common

Change Controller: IETF

Security and privacy considerations: this document, section 4.

5. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8620] Jenkins, N. and C. Newman, "The JSON Meta Application Protocol (JMAP)", RFC 8620, DOI 10.17487/RFC8620, July 2019, <https://www.rfc-editor.org/info/rfc8620>.

Authors' Addresses

Rene Cordier (editor)
Linagora
100 Terrasse Boieldieu - Tour Franklin
Paris - La Defense CEDEX 92042
France

Email: rcordier@linagora.com
URI: https://www.linagora.com

Michael Bailly (editor)
Linagora
100 Terrasse Boieldieu - Tour Franklin
Paris - La Defense CEDEX 92042
France

Email: mbailly@linagora.com
URI: https://www.linagora.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: January 3, 2021

A. Melnikov
Isode Ltd

July 2, 2020

S/MIME signature verification extension to JMAP
draft-ietf-jmap-smime-02

Abstract

This document specifies an extension to JMAP for returning S/MIME signature verification status.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 3, 2021.

Copyright Notice

Table of Contents

1. Introduction

[RFC8621] is a JSON based application protocol for synchronising email data between a client and a server.

This document describes an extension to JMAP for returning S/MIME [RFC8551] signature verification status, without requiring a JMAP client to download the signature body part and all signed body parts (when multipart/signed media type is used) or to download and decode CMS (when application/pkcs7-mime media type is used).

2. Conventions Used in This Document

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 [RFC2119].

3. Addition to the capabilities object

The capabilities object is returned as part of the standard JMAP Session object; see the JMAP spec. Servers supporting _this_ specification MUST add a property called "urn:ietf:params:jmap:smime" to the capabilities object.

The value of this property is an empty object in both the JMAP session _capabilities_ property and an account's _accountCapabilities_ property.

4. Extension to Email/get for S/MIME signature verification

[RFC8621] defines Email/get method for retrieving message specific information. This document defines the following pseudo values in the _properties_ argument:

o *smimeStatus*: If "smimeStatus" is included in the list of requested properties, it MUST be interpreted by the server as a request to return "smimeStatus" response property.
o *smimeErrors*: If "smimeErrors" is included in the list of requested properties, it MUST be interpreted by the server as a request to return "smimeErrors" response property.
o *smimeVerifiedAt*: If "smimeVerifiedAt" is included in the list of requested properties, it MUST be interpreted by the server as a request to return "smimeVerifiedAt" response property.

The "smimeStatus" response property is defined as follows:

smimeStatus: "String|null". null signifies that the message doesn't contain any signature. This property contains the S/MIME signature and certificate verification status calculated according to [RFC8551] and [RFC8550]. Possible string values of the property are listed below. Servers MAY return other values not defined below. Client MUST treat unrecognized values as "unknown" or "signed/failed". Note that the value of this property might change over time.

unknown S/MIME message, but it is neither signed, nor encrypted. This can also be returned for a multipart/signed message which contains unrecognized signing protocol (for example OpenPGP).

signed S/MIME signed message, but the signature was not yet verified. Some servers might not attempt to verify signature until a particular message is requested by the client. JMAP servers compliant with this document SHOULD return "signed/verified" or "signed/failed" instead of this signature status.

signed/verified S/MIME signed message and the sender's signature was successfully verified, sender matches the From header field and the sender's certificate (and the certificate chain) is trusted for signing.

signed/failed S/MIME signed message, but the signature failed to verify. This might be a policy related decision (message signer doesn't match the From header field), message was modified, the signer's certificate has expired or was revoked, etc.

The "smimeErrors" response property is defined as follows:

smimeErrors: "String[]|null". null signifies that the message doesn't contain any signature or that there were no errors when verifying S/MIME signature. (I.e. this property is non null only when the corresponding "smimeStatus" response property value is "signed/failed".) Each string in the array is a human readable description (in the language specified in Content-Language header field, if any)

of a problem with the signature or the signing certificate. (See Section 3.8 of [RFC8620] in regards to how this is affected by the language selection.) For example, the signing certificate might be expired and the message From email address might not correspond to any of the email addresses in the signing certificate. Or the certificate might be expired and the JMAP server might be unable to retrieve CRL for the certificate. In both of these cases there would be 2 elements in the array.

The "smimeVerifiedAt" response property is defined as follows:

smimeVerifiedAt: "UTCDate|null" (server-set). null signifies that the message doesn't contain any S/MIME signature or that there is a signature, but there was no attempt to verify it. In all other cases it is set to the date and time of when S/MIME signature was verified the last time.

"smimeStatus" and "smimeErrors" values are calculated at the time the corresponding JMAP request was processed, not at the time when the message was generated (according to it's Date header field value). It MAY be calculated at the time the message was delivered to the mailbox. In all cases "smimeVerifiedAt" is set to time when "smimeStatus" and "smimeErrors" were last updated. As recalculating these values is expensive for the server they MAY be cached for up to 10 minutes from the moment when they were calculated.

```
["Email/get", {
"ids": [ "f123u987" ],
"properties": [ "mailboxIds", "from", "subject", "date",
 "smimeStatus" ]
}, "#1"]
```

This will result in the following response:

```
[["Email/get", {
   "accountId": "abc",
   "state": "41234123231",
   "list": [
     {
       id: "f123u457",
       mailboxIds: { "f123": true },
       from: [{name: "Joe Bloggs", email: "joe@bloggs.example.net"}],
       subject: "Dinner tonight?",
       date: "2020-07-07T14:12:00Z",
       smimeStatus: "signed/verified"
     }
   ]
}, "#1"]]
```

Example 1:

```
        ["Email/get", {
        "ids": [ "af123u123" ],
        "properties": [ "mailboxIds", "from", "subject", "date",
         "smimeStatus", "smimeErrors", "smimeVerifiedAt" ]
        }, "#1"]
```

This will result in the following response:

```
        [["Email/get", {
           "accountId": "abc",
           "state": "41234123231",
           "list": [
             {
               id: "af123u123",
               mailboxIds: { "f123": true },
               from: [{name: "Jane Doe",
                       email: "jdoe@example.com"}],
               subject: "Company takeover",
               date: "2020-01-31T23:00:00Z",
               smimeStatus: "signed/failed",
               smimeErrors: [
                 "From email address doesn't match the certificate",
                 "Can't retrieve CRL from the CRL URL"],
               "smimeVerifiedAt": "2020-03-01T12:11:19Z"
             }
           ]
        }, "#1"]]
```

Example 2:

5. Open Issues

[[This section should be empty before publication]]

1. Should a new property be added on requests to allow signature verification "at specified time"?

6. IANA Considerations

6.1. JMAP capability registration for "smime"

IANA is requested to register the "smime" JMAP Capability as follows:

Capability Name: "urn:ietf:params:jmap:smime"

Specification document: this document

Intended use: common

Change Controller: IETF

Security and privacy considerations: this document, Section 7

7. Security Considerations

Server side S/MIME signature verification requires the client to trust server verification code and configuration to perform S/MIME signature verification. For example, if the server is not configured with some Trust Anchors, some messages will have "signed/failed" status instead of "signed/verified".

Constant recalculation of S/MIME signature status can result in Denial-of-Service condition. For that reason it is RECOMMENDED to cache results of signature verification for 10 minutes.

8. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC8550] Schaad, J., Ramsdell, B., and S. Turner, "Secure/ Multipurpose Internet Mail Extensions (S/MIME) Version 4.0 Certificate Handling", RFC 8550, DOI 10.17487/RFC8550, April 2019, <https://www.rfc-editor.org/info/rfc8550>.

[RFC8551] Schaad, J., Ramsdell, B., and S. Turner, "Secure/ Multipurpose Internet Mail Extensions (S/MIME) Version 4.0 Message Specification", RFC 8551, DOI 10.17487/RFC8551, April 2019, <https://www.rfc-editor.org/info/rfc8551>.

[RFC8620] Jenkins, N. and C. Newman, "The JSON Meta Application Protocol (JMAP)", RFC 8620, DOI 10.17487/RFC8620, July 2019, <https://www.rfc-editor.org/info/rfc8620>.

[RFC8621] Jenkins, N. and C. Newman, "The JSON Meta Application Protocol (JMAP) for Mail", RFC 8621, DOI 10.17487/RFC8621, August 2019, <https://www.rfc-editor.org/info/rfc8621>.

Author's Address

Alexey Melnikov
Isode Ltd
14 Castle Mews
Hampton, Middlesex TW12 2NP
UK

EMail: Alexey.Melnikov@isode.com

Independent Submission
Internet-Draft
Intended status: Standards Track
Expires: September 6, 2020

K. Murchison
FastMail

March 5, 2020

JMAP for Sieve Scripts
draft-murchison-jmap-sieve-01

Abstract

This document specifies a data model for managing Sieve scripts on a server using JMAP.

Open Issues

- How should doing /set{create} with an existing script name be handled? Should it fail or overwrite the existing script? Should the /set request include an 'overwrite' boolean argument?
- Should setting isActive==true on a script automatically deactivate any other existing active script, or should the client have to do so itself (as is currently documented)?
- Do we want/need a SieveScript/copy method?
- Do we want to leverage draft-ietf-jmap-quotas to query Sieve script storage quotas?

Status of This Memo

Table of Contents

1. Introduction

JMAP ([RFC8620] - JSON Meta Application Protocol) is a generic protocol for synchronizing data, such as mail, calendars or contacts, between a client and a server. It is optimized for mobile and web environments, and aims to provide a consistent interface to different data types.

This specification defines a data model for managing Sieve [RFC5228] scripts on a server using JMAP. The data model is designed to allow a server to provide consistent access to the same scripts via ManageSieve [RFC5804] as well as JMAP, however the functionality offered over the two protocols may differ.

1.1. Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

Type signatures, examples, and property descriptions in this document follow the conventions established in Section 1.1 of [RFC8620]. Data types defined in the core specification are also used in this document.

1.2. Terminology

The same terminology is used in this document as in the core JMAP specification, see [RFC8620], Section 1.6.

The term SieveScript (with this specific capitalization) is used to refer to the data type defined in this document and instances of those data types.

1.3. Addition to the Capabilities Object

The capabilities object is returned as part of the JMAP Session object; see [RFC8620], Section 2. This document defines one additional capability URI.

1.3.1. urn:ietf:params:jmap:sieve

This represents support for the SieveScript data type and associated API methods. The value of this property in the JMAP Session capabilities property is an empty object.

The value of this property in an account’s accountCapabilities property is an object that MUST contain the following information on server capabilities:

- *maxNumberRedirects*: "UnsignedInt|null" The maximum number of Sieve "redirect" actions a script can perform during a single evaluation (see [RFC5804], Section 1.7), or "null" for no limit.

- *maxNumberScripts*: "UnsignedInt|null" The maximum number of Sieve scripts the server is willing to store for the user, or "null" for no limit.

- *maxSizeScript*: "UnsignedInt|null" The maximum size (in octets) of a Sieve script the server is willing to store for the user, or "null" for no limit.

- *sieveExtensions*: "String[]" A list of Sieve extensions (as listed in Sieve "require" action [RFC5228], Section 3.2) supported by the Sieve engine.

- *notificationMethods*: "String[]|null" A list of URI schema parts [RFC3986] for notification methods supported by the Sieve "enotify" extension [RFC5435], or "null" if the extension is not supported by the Sieve engine.

- *externalLists*: "String[]|null" A list of URI schema parts [RFC3986] for externally stored list types supported by the Sieve "extlists" extension [RFC6134], or "null" if the extension is not supported by the Sieve engine.

2. Sieve Scripts

A *SieveScript* object represents a single script on the server and has the following properties:

- *id*: "Id" (immutable; server-set) The id of the script.

- *name*: "String" The user-visible name for the script, subject to the requirements in [RFC5804], Section 1.6.

- *content*: "String" The Sieve code in the script. Note that any double (") quote or backslash (\) characters appearing in the script content MUST be escaped by prefixing them with a backslash (\).

- *isActive*: "Boolean" (default: false) Is this the user's active script?

Example (using the Imap4Flags [RFC5232] Extension):

```
{
  "id": "665c423a-6991-4733-8c7c-52b299572c66",
  "name": "example.siv",
  "content":
     "require [ \"imap4flags\" ];\r\nkeep :flags \"\\\\flagged\";",
  "isActive": false
}
```

2.1. SieveScript/get

This is a standard "/get" method as described in [RFC8620], Section 5.1. The _ids_ argument may be "null" to fetch all at once.

This method provides similar functionality to the GETSCRIPT and LISTSCRIPTS commands in [RFC5804].

2.2. SieveScript/set

This is a standard "/set" method as described in [RFC8620], Section 5.3.

This method provides similar functionality to the PUTSCRIPT, DELETESCRIPT, RENAMESCRIPT, and SETACTIVE commands in [RFC5804].

Per [RFC5804], Section 1.4, a user may have multiple Sieve scripts on the server, yet only one script may be active. Therefore, when changing the active script, the call to this method MUST both set the _isActive_ argument on the currently active script to "false" and set it to "true" on the script to be activated.

The following extra SetError type is defined:

For "create" and "update":

- *scriptIsActive*: The "isActive" argument was true and the user already has another active script. The SetError object SHOULD also include the *id* property of the currently active script.

2.3. SieveScript/validate

This method is used by the client to verify Sieve script validity without storing the script on the server.

The method provides similar functionality to the CHECKSCRIPT command in [RFC5804].

The server MUST check the submitted script for syntactic validity, which includes checking that all Sieve extensions mentioned in Sieve script "require" statement(s) are supported by the Sieve interpreter. (Note that if the Sieve interpreter supports the Sieve "ihave" extension [RFC5463], any unrecognized/unsupported extension mentioned in the "ihave" test MUST NOT cause the syntactic validation failure.)

The *SieveScript/validate* method takes the following arguments:

- *accountId*: "Id" The id of the account to use.
- *content*: "String" The Sieve code to validate. Note that any double (") quote or backslash (\) characters appearing in the script content MUST be escaped by prefixing them with a backslash (\).

The response has the following arguments:

- *accountId*: "Id" The id of the account used for this call.
- *isValid*: "Boolean" Is the Sieve code valid?
- *errorDescription*: "String" A description of the error to show to the user, or an empty string if the Sieve code is valid.

3. Security Considerations

All security considerations of JMAP [RFC8620] apply to this specification.

4. IANA Considerations

4.1. JMAP Capability Registration for "sieve"

IANA will register the "sieve" JMAP Capability as follows:

Capability Name: "urn:ietf:params:jmap:sieve"

Specification document: this document

Intended use: common

Change Controller: IETF

Security and privacy considerations: this document, Section 3

4.2. JMAP Error Codes Registry

The following sub-section registers a new error code in the JMAP Error Codes registry, as defined in [RFC8620].

4.2.1. scriptIsActive

JMAP Error Code: scriptIsActive

Intended use: common

Change controller: IETF

Reference: This document, section 2.5

Description: The client tried to activate a Sieve script, but another acript is already active.

5. Acknowledgments

The concepts in this document are based largely on those in [RFC5804]. The author would like to thank the authors of that document for providing both inspiration and some borrowed text for this document.

6. References

6.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[RFC5228] Guenther, P., Ed. and T. Showalter, Ed., "Sieve: An Email Filtering Language", RFC 5228, DOI 10.17487/RFC5228, January 2008, <https://www.rfc-editor.org/info/rfc5228>.

[RFC5435] Melnikov, A., Ed., Leiba, B., Ed., Segmuller, W., and T. Martin, "Sieve Email Filtering: Extension for Notifications", RFC 5435, DOI 10.17487/RFC5435, January 2009, <https://www.rfc-editor.org/info/rfc5435>.

[RFC5804] Melnikov, A., Ed. and T. Martin, "A Protocol for Remotely Managing Sieve Scripts", RFC 5804, DOI 10.17487/RFC5804, July 2010, <https://www.rfc-editor.org/info/rfc5804>.

[RFC6134] Melnikov, A. and B. Leiba, "Sieve Extension: Externally Stored Lists", RFC 6134, DOI 10.17487/RFC6134, July 2011, <https://www.rfc-editor.org/info/rfc6134>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8620] Jenkins, N. and C. Newman, "The JSON Meta Application Protocol (JMAP)", RFC 8620, DOI 10.17487/RFC8620, July 2019, <https://www.rfc-editor.org/info/rfc8620>.

6.2. Informative References

[RFC5232] Melnikov, A., "Sieve Email Filtering: Imap4flags Extension", RFC 5232, DOI 10.17487/RFC5232, January 2008, <https://www.rfc-editor.org/info/rfc5232>.

[RFC5463] Freed, N., "Sieve Email Filtering: Ihave Extension", RFC 5463, DOI 10.17487/RFC5463, March 2009, <https://www.rfc-editor.org/info/rfc5463>.

Appendix A. Change History (To be removed by RFC Editor before publication)

Changes since -00:

- Added IANA registration for "scriptIsActive" JMAP error code.
- Added open issue about /set{create} with an existing script name.

Author's Address

Kenneth Murchison
Fastmail US LLC
1429 Walnut Street - Suite 1201
Philadelphia, PA 19102
USA

Email: murch@fastmailteam.com