Abstract

This document specifies a data model for synchronising email 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 May 3, 2018.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

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

JMAP is a generic protocol for synchronising data, such as mail, 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 synchronising mail between a client and a server using JMAP.

1.1. Notational Conventions

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].

The underlying format used for this specification is I-JSON ([RFC7493]). Consequently, the terms "object" and "array" as well as the four primitive types (strings, numbers, booleans, and null) are to be interpreted as described in Section 1 of [RFC7159]. Unless otherwise noted, all the property names and values are case sensitive.

Some examples in this document contain "partial" JSON documents used for illustrative purposes. In these examples, three periods "..." are used to indicate a portion of the document that has been removed for compactness.

Types signatures are given for all JSON objects in this document. The following conventions are used:

- "Boolean|String" - The value is either a JSON "Boolean" value, or a JSON "String" value.
- "Foo" - Any name that is not a native JSON type means an object for which the properties (and their types) are defined elsewhere within this document.
- "Foo[]" - An array of objects of type "Foo".
- "String[Foo]" - A JSON "Object" being used as a map (associative array), where all the values are of type "Foo".

Object properties may also have a set of attributes defined along with the type signature. These have the following meanings:
*sever-set*: Only the server can set the value for this property. The client MUST NOT send this property when creating a new object of this type.

*immutable*: The value MUST NOT change after the object is created.

*default*: (This is followed by a JSON value). The value that will be used for this property if it is omitted when creating a new object of this type.

1.2. Terminology

The same terminology is used in this document as in the core JMAP specification.

1.3. 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 "{TODO: URI for this spec}" to the capabilities object. The value of this property is an object which SHOULD contain the following information on server capabilities:

*maxMailboxesPerMessage*: "Number|null" The maximum number of mailboxes that can be assigned to a single message. This MUST be an integer >= 1, or "null" for no limit (or rather, the limit is always the number of mailboxes in the account).

*maxSizeMessageAttachments*: "Number" The maximum total size of attachments, in bytes, allowed for messages. A server MAY still reject messages with a lower attachment size total (for example, if the body includes several megabytes of text, causing the size of the encoded MIME structure to be over some server-defined limit).

*maxDelayedSend*: "Number" The number in seconds of the maximum delay the server supports in sending (see the MessageSubmission object). This is "0" if the server does not support delayed send.

*messageListSortOptions*: "String[]" A list of all the message properties the server supports for sorting by. This MAY include properties the client does not recognise (for example custom properties specified in a vendor extension). Clients MUST ignore any unknown properties in the list.
A JMAP implementation that talks to a Submission [RFC6409] server SHOULD have a configuration setting that allows an administrator to expose a new submission EHLO capability in this field. This allows a JMAP server to gain access to a new submission extension without code changes. By default, the JMAP server should show only known safe-to-expose EHLO capabilities in this field, and hide EHLO capabilities that are only relevant to the JMAP server. Each key in the object is the _ehlo-name_, and the value is a list of _ehlo-args_. Examples of safe-to-expose Submission extensions include:

* FUTURERELEASE ([RFC4865])
* SIZE ([RFC1870])
* DSN ([RFC3461])
* DELIVERYBY ([RFC2852])
* MT-PRIORITY ([RFC6710])

A JMAP server MAY advertise an extension and implement the semantics of that extension locally on the JMAP server even if a submission server used by JMAP doesn't implement it. The full IANA registry of submission extensions can be found at <https://www.iana.org/assignments/mail-parameters/mail-parameters.xhtml#mail-parameters-2>

2. Mailboxes

A mailbox represents a named set of emails. This is the primary mechanism for organising messages within an account. It is analogous to a folder or a label in other systems. A mailbox may perform a certain role in the system; see below for more details.

For compatibility with IMAP, a message MUST belong to one or more mailboxes. The message id does not change if the message changes mailboxes.

A *Mailbox* object has the following properties:

* *id*: "String" (immutable; server-set) The id of the mailbox.

* *name*: "String" User-visible name for the mailbox, e.g. "Inbox". This may be any UTF-8 string ([RFC3629]) of at least 1 character in length and maximum 256 bytes in size. Servers SHOULD forbid sibling Mailboxes with the same name.
- **parentId**: "String|null" (default: "null") The mailbox id for the parent of this mailbox, or "null" if this mailbox is at the top level. Mailboxes form acyclic graphs (forests) directed by the child-to-parent relationship. There MUST NOT be a loop.

- **role**: "String|null" (default: "null") Identifies system mailboxes. This property can only be set on create. After the record has been created, this property is immutable. The following values MUST be used for the relevant mailboxes:
  
  * "inbox" - the mailbox to which new mail is delivered by default, unless diverted by a rule or spam filter etc.
  
  * "archive" - messages the user does not need right now, but does not wish to delete.
  
  * "drafts" - messages the user is currently writing and are not yet sent.
  
  * "sent" - messages the user has sent.
  
  * "trash" - messages the user has deleted.
  
  * "spam" - messages considered spam by the server.
  
  * "templates" - drafts which should be used as templates (i.e. used as the basis for creating new drafts).

No two mailboxes may have the same role. Mailboxes without a known purpose MUST have a role of "null". An account is not required to have mailboxes with any of the above roles. A client MAY create new mailboxes with a role property to help them keep track of a use-case not covered by the above list. To avoid potential conflict with any special behaviour a server might apply to mailboxes with certain roles in the future, any roles not in the above list created by the client must begin with "x-". The client MAY attempt to create mailboxes with the standard roles if not already present, but the server MAY reject these.

- **sortOrder**: "Number" (default: "0") Defines the sort order of mailboxes 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 mailbox with a lower order should be displayed before a mailbox with a higher order (that has the same parent) in any mailbox listing in the client's UI. Mailboxes with equal order SHOULD be sorted in alphabetical order by name. The sorting SHOULD take into account locale-specific character order convention.
*mayReadItems*: "Boolean" (server-set) If true, may use this mailbox as part of a filter in a _getMessageList_ call. If a submailbox is shared but not the parent mailbox, this may be "false".

*mayAddItems*: "Boolean" (server-set) The user may add messages to this mailbox (by either creating a new message or moving an existing one).

*mayRemoveItems*: "Boolean" (server-set) The user may remove messages from this mailbox (by either changing the mailboxes of a message or deleting it).

*mayCreateChild*: "Boolean" (server-set) The user may create a mailbox with this mailbox as its parent.

*mayRename*: "Boolean" (server-set) The user may rename the mailbox or make it a child of another mailbox.

*mayDelete*: "Boolean" (server-set) The user may delete the mailbox itself.

*totalMessages*: "Number" (server-set) The number of messages in this mailbox.

*unreadMessages*: "Number" (server-set) The number of messages in this mailbox that have neither the "$Seen" keyword nor the "$Draft" keyword.

*totalThreads*: "Number" (server-set) The number of threads where at least one message in the thread is in this mailbox.

*unreadThreads*: "Number" (server-set) The number of threads where at least one message in the thread has neither the "$Seen" keyword nor the "$Draft" keyword AND at least one message in the thread is in this mailbox (but see below for special case handling of Trash). Note, the unread message does not need to be the one in this mailbox.

The Trash mailbox (that is a mailbox with "role == "trash"")) MUST be treated specially for the purpose of unread counts:

1. Messages that are *only* in the Trash (and no other mailbox) are ignored when calculating the "unreadThreads" count of other mailboxes.

2. Messages that are *not* in the Trash are ignored when calculating the "unreadThreads" count for the Trash mailbox.
The result of this is that messages in the Trash are treated as though they are in a separate thread for the purposes of unread counts. It is expected that clients will hide messages in the Trash when viewing a thread in another mailbox and vice versa. This allows you to delete a single message to the Trash out of a thread.

So for example, suppose you have an account where the entire contents is a single conversation with 2 messages: an unread message in the Trash and a read message in the Inbox. The "unreadThreads" count would be "1" for the Trash and "0" for the Inbox.

For IMAP compatibility, a message in both the Trash and another mailbox SHOULD be treated by the client as existing in both places (i.e. when emptying the trash, the client SHOULD just remove the Trash mailbox and leave it in the other mailbox).

The following JMAP methods are supported:

2.1. getMailboxes

Standard _getFoos_ method. The _ids_ argument may be "null" to fetch all at once.

2.2. getMailboxUpdates

Standard _getFooUpdates_ method, but with one extra argument to the _mailboxUpdates_ response:

o *changedProperties*: "String[]|null" If only the mailbox counts (unread/total messages/threads) have changed since the old state, this will be the list of properties that may have changed, i.e. ":["totalMessages", "unreadMessages", "totalThreads", "unreadThreads"]". If the server is unable to tell if only counts have changed, it MUST just be "null".

Since counts frequently change but the rest of the mailboxes state for most use cases changes rarely, the server can help the client optimise data transfer by keeping track of changes to message counts separately to other state changes. The _changedProperties_ array may be used directly via a result reference in a subsequent getMailboxes call in a single request.

2.3. getMailboxList

Standard _getFooList_ method.

The *FilterCondition* object (optionally passed as the _filter_ argument) has the following properties, any of which may be omitted:
o  *parentId*: "String|null" The Mailbox _parentId_ property must match the given value exactly.

o  *hasRole*: "Boolean" If this is "true", a Mailbox matches if it has a non-"null" value for its _role_ property.

A Mailbox object matches the filter if and only if all of the given conditions given match. If zero properties are specified, it is automatically "true" for all objects.

The following properties MUST be supported for sorting:

  o  "sortOrder"

  o  "name"

2.4.  getMailboxListUpdates

Standard _getFooListUpdates_ method.

2.5.  setMailboxes

Standard _setFoos_ method. The following extra _SetError_ types are defined:

For *create*:

  o  "maxQuotaReached": The user has reached a server-defined limit on the number of mailboxes.

For *update*:

  o  "forbidden": The update would violate a mayXXX property.

For *destroy*:

  o  "forbidden": The update would violate a mayXXX property.

  o  "mailboxHasChild": The mailbox still has at least one child mailbox. The client MUST remove these before it can delete the parent mailbox.

  o  "mailboxHasMessage": The mailbox has at least one message assigned to it. The client MUST remove these before it can delete the mailbox.
3. Threads

Replies are grouped together with the original message to form a thread. In JMAP, a thread is simply a flat list of messages, ordered by date. Every message MUST belong to a thread, even if it is the only message in the thread.

The JMAP spec does not require the server to use any particular algorithm for determining whether two messages belong to the same thread, however there is a recommended algorithm in the implementation guide [1].

If messages are delivered out of order for some reason, a user may receive two messages in the same thread but without headers that associate them with each other. The arrival of a third message in the thread may provide the missing references to join them all together into a single thread. Since the "threadId" of a message is immutable, if the server wishes to merge the threads, it MUST handle this by deleting and reinserting (with a new message id) the messages that change threadId.

A *Thread* object has the following properties:

- *id*: "String" (immutable) The id of the thread.
- *messageIds*: "String[]" The ids of the messages in the thread, sorted such that:
  
  * Any message with the "$Draft" keyword that has an "In-Reply-To" header is sorted after the _first_ non-draft message in the thread with the corresponding "Message-Id" header, but before any subsequent non-draft messages.
  
  * Other than that, everything is sorted in _date_ order (as determined by the date property on the _Message_ object), oldest first.
  
  * If two messages are identical under the above two conditions, the sort is server-dependent but MUST be stable (sorting by id is recommended).

The following JMAP methods are supported:

3.1. getThreads

Standard _getFoos_ method.
3.1.1. Example

Request:

```json
[ "getThreads", {
    "ids": ["f123u4", "f41u44"],
}, "#1" ]
```

with response:

```json
[ "threads", {
    "accountId": "acme",
    "state": "f6a7e214",
    "list": [
        {
            "id": "f123u4",
            "messageIds": [ "eaa623", "f782cbb"]
        },
        {
            "id": "f41u44",
            "messageIds": [ "82cf7bb"]
        }
    ],
    "notFound": null
}, "#1" ]
```

3.2. getThreadUpdates

Standard _getFooUpdates_ method.

4. Messages

A *Message* object is a JSON representation of an [RFC5322] message that hides the complexities of MIME. All special encodings of either headers or textual body parts, such as Base64 ([RFC4648]), or [RFC2047] encoding of non-ASCII characters, MUST be fully decoded into UTF-8. It has the following properties:

- **id**: "String" (immutable; server-set) The id of the message. This is the JMAP id, NOT the [RFC5322] Message-Id header.

- **blobId**: "String" (immutable; server-set) The id representing the raw [RFC5322] message. This may be used to download the original message or to attach it directly to another message etc.

- **threadId**: "String" (immutable; server-set) The id of the thread to which this message belongs.
*mailboxIds*: "String[Boolean]" The set of mailbox ids this message is in. A message MUST belong to one or more mailboxes at all times (until it is deleted). The set is represented as an object, with each key being a _Mailbox id_. The value for each key in the object MUST be "true".

*keywords*: "String[Boolean]" (default: "{}") A set of keywords that apply to the message. The set is represented as an object, with the keys being the _keywords_. The value for each key in the object MUST be "true". Keywords are shared with IMAP. The six system keywords from IMAP are treated specially. The following four keywords have their first character changed from "\" in IMAP to "$" in JMAP and have particular semantic meaning:

* "$Draft": The message is a draft the user is composing.
* "$Seen": The message has been read.
* "$Flagged": The message has been flagged for urgent/special attention.
* "$Answered": The message has been replied to.

The IMAP "\Recent" keyword is not exposed via JMAP. The IMAP "\Deleted" keyword is also not present: IMAP uses a delete+expunge model, which JMAP does not. Any message with the "\Deleted" keyword MUST NOT be visible via JMAP. Users may add arbitrary keywords to a message. For compatibility with IMAP, a keyword is a (case-sensitive) string of 1-255 characters in the ASCII subset %x21-%x7e (excludes control chars and space), and MUST NOT include any of these characters: "( ) { ] % * " ". The IANA Keyword Registry [2] as established in [RFC5788] assigns semantic meaning to some other keywords in common use. New keywords may be established here in the future. In particular, note:

* "$Forwarded": The message has been forwarded.
* "$Phishing": The message is highly likely to be phishing. Clients SHOULD warn users to take care when viewing this message and disable links and attachments.
* "$Junk": The message is definitely spam. Clients SHOULD set this flag when users report spam to help train automated spam-detection systems.
* "$NotJunk": The message is definitely not spam. Clients SHOULD set this flag when users indicate a message is legitimate, to help train automated spam-detection systems.
o  *hasAttachment*: "Boolean" (immutable; server-set) This is "true"
   if and only if the _attachments_ property for the Message contains
   at least one entry where _isInline_ is "false".

o  *headers*: "String[String]" (immutable; default: "{}") A map of
   lower-cased header name to (decoded) header value for all headers
   in the message. For headers that occur multiple times (e.g.
   "Received"), the values are concatenated with a single new line
   ("\n") character in between each one.

o  *sender*: "Emailer|null" (immutable; default: "null") An Emailer
   object (see below) containing the name/email from the parsed
   "Sender" header of the email. If the email doesn't have a
   "Sender" header, this is "null".

o  *from*: "Emailer[]|null" (immutable; default: "null") An array of
   name/email objects (see below) representing the parsed "From"
   header of the email, in the same order as they appear in the
   header. If the email doesn't have a "From" header, this is
   "null". If the header exists but does not have any content, the
   response is an array of zero length.

o  *to*: "Emailer[]|null" (immutable; default: "null") An array of
   name/email objects (see below) representing the parsed "To"
   header of the email, in the same order as they appear in the
   header. If the email doesn't have a "To" header, this is
   "null". If the header exists but does not have any content, the
   response is an array of zero length.

o  *cc*: "Emailer[]|null" (immutable; default: "null") An array of
   name/email objects (see below) representing the parsed "Cc"
   header of the email, in the same order as they appear in the
   header. If the email doesn't have a "Cc" header, this is
   "null". If the header exists but does not have any content, the
   response is an array of zero length.

o  *bcc*: "Emailer[]|null" (immutable; default: "null") An array of
   name/email objects (see below) representing the parsed "Bcc"
   header of the email. If the email doesn't have a "Bcc" header
   (which will be true for most emails outside of the Sent mailbox),
   this is "null". If the header exists but does not have any
   content, the response is an array of zero length.

o  *replyTo*: "Emailer[]|null" (immutable; default: "null") An array
   of name/email objects (see below) representing the parsed "Reply-
   To" header of the email, in the same order as they appear in the
   header. If the email doesn't have a "Reply-To" header, this is
"null". If the header exists but does not have any content, the response is an array of zero length.

- **subject**: "String" (immutable; default: """") The subject of the message. If none, defaults to the empty string, not "null".

- **date**: "Date" (immutable; default: time of creation on server) The date the message was sent (or saved, if the message is a draft).

- **size**: "Number" (immutable; server-set) The size in bytes of the whole message as counted by the server towards the user's quota.

- **preview**: "String" (immutable; server-set) Up to 256 characters of the beginning of a plain text version of the message body. This is intended to be shown as a preview line on a mailbox listing, and the server may choose to skip quoted sections or salutations to return a more useful preview.

- **textBody**: "String" (immutable; default: """") The plain text body part for the message. If there is only an HTML version of the body, a plain text version MUST be generated from this; the exact method of conversion in this case is not defined and is server-specific. If there is neither a "text/plain" nor a "text/html" body part, this MUST be the empty string.

- **htmlBody**: "String|null" (immutable; default: "null") The HTML body part for the message if present.

- **attachments**: "Attachment[]|null" (default: "null") An array of attachment objects (see below) detailing all the attachments to the message.

- **attachedMessages**: "String[Message]|null" (immutable; server-set) An object mapping attachment id (as found in the "attachments" property) to a *Message* object with the following properties, for each [RFC5322] message attached to this one:
  - headers
  - from
  - to
  - cc
  - bcc
* replyTo
* subject
* date
* textBody
* htmlBody
* attachments
* attachedMessages

This property is set by the server based on the _attachments_ property.

An *Emailer* object has the following properties:

- *name*: "String" The name of the sender/recipient. If a name cannot be extracted for an email, this property SHOULD be the empty string.

- *email*: "String" The email address of the sender/recipient. This MUST be of the form "<mailbox>@<host>" If a "host" or even "mailbox" cannot be extracted for an email, the empty string SHOULD be used for this part (so the result MUST always still contain an "@" character).

Group information and comments from the RFC 5322 header MUST be discarded when converting into an Emailer object.

Example array of Emailer objects:

```
[  
    {name:"Joe Bloggs", email:"joeb@example.com"},
    {name:"", email:"john@example.com"},
    {name:"John Smith", email: "john@"}
]
```

An *Attachment* object has the following properties:

- *blobId*: "String" The id of the binary data.

- *type*: "String" The content-type of the attachment.

- *name*: "String|null" The full file name, e.g. "myworddocument.doc", if available.
*size*: "Number" The size, in bytes, of the attachment when fully decoded (i.e. the number of bytes in the file the user would download).

*cid*: "String|null" The id used within the message body to reference this attachment. This is only unique when paired with the message id, and has no meaning without reference to that.

*isInline*: "Boolean" True if the attachment is referenced by a "cid:" link from within the HTML body of the message.

*width*: "Number|null" (optional, server MAY omit if not supported) The width (in px) of the image, if the attachment is an image.

*height*: "Number|null" (optional, server MAY omit if not supported) The height (in px) of the image, if the attachment is an image.

To add an attachment, the file must first be uploaded using the standard upload mechanism; this will give the client a blobId that may be used to identify the file. The "cid" property may be assigned by the client, and is solely used for matching up with "cid:<id>" links inside the "htmlBody".

The following JMAP methods are supported:

4.1. getMessages

Standard _getFoos_ method, except the client may use the following pseudo values in the _properties_ argument:

*body*: If ""body"" is included in the list of requested properties, it MUST be interpreted by the server as a request for ""htmlBody"" if the message has an HTML part, or ""textBody"" otherwise.

*headers.property*: Instead of requesting all the headers (by requesting the ""headers"" property, the client may specify the particular headers it wants using the "headers.property-name" syntax, e.g. ""headers.x-spam-score", "headers.x-spam-hits""). The server MUST return a _headers_ property but with just the requested headers in the object rather than all headers. If ""headers"" is requested, the server MUST ignore the individual header requests and just return all headers. If a requested header is not present in the message, it MUST NOT be present in the _headers_ object. Header names are case-insensitive.
4.1.1. Example

Request:

["getMessages", {
  "ids": [ "f123u456", "f123u457" ],
  "properties": [ "threadId", "mailboxIds", "from", "subject", "date" ]
}, "#1"]

and response:

["messages", {
  "accountId": "abc",
  "state": "41234123231",
  "list": [
    {
      id: "f123u457",
      threadId: "ef1314a",
      mailboxIds: { "f123": true },
      from: [{name: "Joe Bloggs", email: "joe@bloggs.com"}],
      subject: "Dinner on Thursday?",
      date: "2013-10-13T14:12:00Z"
    }
  ],
  notFound: [ "f123u456" ]
}, "#1"]

4.2. getMessageUpdates

Standard _getFooUpdates_ method.

4.3. getMessageList

Standard _getFooList_ method, but with the following additional arguments:

  - *collapseThreads*: "Boolean" (default: "false") If "true", messages in the same thread as a previous message in the list (given the filter and sort order) will be removed from the list. This means at most only one message will be included in the list for any given thread.

4.3.1. Filtering

A *FilterOperator* object has the following properties:

  - *operator*: "String" This MUST be one of the following strings: "AND"/"OR"/"NOT":

* "AND": all of the conditions must match for the filter to match.

* "OR": at least one of the conditions must match for the filter to match.

* "NOT": none of the conditions must match for the filter to match.

- *conditions*: "(FilterCondition|FilterOperator)[]" The conditions to evaluate against each message.

A *FilterCondition* object has the following properties, any of which may be omitted:

- *inMailbox*: "String" A mailbox id. A message must be in this mailbox to match the condition.

- *inMailboxOtherThan*: "String" A mailbox id. A message be in any mailbox other than this one to match the condition. This is to allow messages solely in trash/spam to be easily excluded from a search.

- *before*: "Date" The date of the message (as returned on the Message object) must be before this date to match the condition.

- *after*: "Date" The date of the message (as returned on the Message object) must be on or after this date to match the condition.

- *minSize*: "Number" The size of the message in bytes (as returned on the Message object) must be equal to or greater than this number to match the condition.

- *maxSize*: "Number" The size of the message in bytes (as returned on the Message object) must be less than this number to match the condition.

- *allInThreadHaveKeyword*: "String" All messages (including this one) in the same thread as this message must have the given keyword to match the condition.

- *someInThreadHaveKeyword*: "String" At least one message (possibly this one) in the same thread as this message must have the given keyword to match the condition.
o  *noneInThreadHaveKeyword*: "String" All messages (including this one) in the same thread as this message must *not* have the given keyword to match the condition.

o  *hasKeyword*: "String" This message must have the given keyword to match the condition.

o  *notKeyword*: "String" This message must not have the given keyword to match the condition.

o  *hasAttachment*: "Boolean" The "hasAttachment" property of the message must be identical to the value given to match the condition.

o  *text*: "String" Looks for the text in messages. The server SHOULD look up text in the _from_, _to_, _cc_, _bcc_, _subject_, _textBody_, _htmlBody_ or _attachments_ properties of the message. The server MAY extend the search to any additional textual property.

o  *from*: "String" Looks for the text in the _from_ property of the message.

o  *to*: "String" Looks for the text in the _to_ property of the message.

o  *cc*: "String" Looks for the text in the _cc_ property of the message.

o  *bcc*: "String" Looks for the text in the _bcc_ property of the message.

o  *subject*: "String" Looks for the text in the _subject_ property of the message.

o  *body*: "String" Looks for the text in the _textBody_ or _htmlBody_ property of the message.

o  *attachments*: "String" Looks for the text in the attachments of the message. Server MAY handle text extraction when possible for the different kinds of media.

o  *header*: "String[]" The array MUST contain either one or two elements. The first element is the name of the header to match against. The second (optional) element is the text to look for in the header. If not supplied, the message matches simply if it _has_ a header of the given name.
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 word or 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 message, but MUST all be present for the message 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").
- When searching inside the _htmlBody_ property, HTML tags and attributes SHOULD be ignored.

### 4.3.2 Sorting

The following properties MUST be supported for sorting:

- *date* - The date as returned in the Message object.

The following properties SHOULD be supported for sorting:

- *size* - The size as returned in the Message object.
- *from* - This is taken to be either the "name" part of the Eemailer object, or if none then the "email" part of the Eemailer object (see the definition of the from property in the Message object). If still none, consider the value to be the empty string.
- *to* - This is taken to be either the "name" part of the *first* Eemailer object, or if none then the "email" part of the *first* Eemailer object (see the definition of the to property in the
Message object). If still none, consider the value to be the empty string.

- **subject** - This is taken to be the subject of the Message with any ignoring any leading "Fwd:"s or "Re:"s (case-insensitive match).

- **keyword**: "$keyword" - This value MUST be considered "true" if the message has the keyword, or "false" otherwise.

- **allThreadKeyword**: "$keyword" - This value MUST be considered "true" for the message if *all* of the messages in the same thread (regardless of mailbox) have the keyword.

- **someThreadKeyword**: "$keyword" - This value MUST be considered "true" for the message if *any* of the messages in the same thread (regardless of mailbox) have the keyword.

The server MAY support sorting based on other properties as well. A client can discover which properties are supported by inspecting the server's _capabilities_ object (see section 1).

Example sort:

```
[ "someThreadKeyword:$Flagged desc", "date desc" ]
```

This would sort messages in flagged threads first (the thread is considered flagged if any message within it is flagged), and then in date order, newest first. If two messages have both identical flagged status and date, the order is server-dependent but must be stable.

### 4.3.3 Thread collapsing

When "collapseThreads == true", then after filtering and sorting the message list, the list is further winnowed by removing any messages for a thread id that has already been seen (when passing through the list sequentially). A thread will therefore only appear "once" in the "threadIds" list of the result, at the position of the first message in the list that belongs to the thread.

### 4.3.4 Response

The _messageList_ response has the following additional argument:

- **collapseThreads**: "Boolean" The _collapseThreads_ value that was used when calculating the message list for this call.
4.4.  getMessageListUpdates

Standard _getFooListUpdates_ method, with the following additional arguments:

- *collapseThreads*: "Boolean" (default: "false") The _collapseThreads_ argument that was used with _getMessageList_.

The _messageListUpdates_ response has the following additional arguments:

- *collapseThreads*: "Boolean" The _collapseThreads_ value that was used when calculating the message list for this call.

4.5.  setMessages

Standard _setFoos_ method. The _setMessages_ method encompasses:

- Creating a draft message
- Changing the flags of a message (unread/flagged status)
- Adding/removing a message to/from mailboxes (moving a message)
- Deleting messages

When creating a message, the _headers_ property specifies extra headers to add in addition to any based off the parsed properties (like _from_/_to_/_subject_). The keys MUST only contain the characters a-z (lower-case only), 0-9 and hyphens. If a header is included that conflicts with one of the other properties on the Message object (e.g. _from_, _date_), the value in the _headers_ object MUST be ignored.

The server MAY also choose to set additional headers. If not included, the server MUST generate and set a "Message-Id" header in conformance with [RFC5322] section 3.6.4.

Other than making sure it conforms to the correct type, the server MUST NOT attempt to validate _from_/_to_/_cc_/_bcc_ (e.g. checking if an email address is valid) when creating a message. This is to ensure draft messages can be saved at any point.

Destroying a message removes it from all mailboxes to which it belonged. To just delete a message to trash, simply change the "mailboxIds" property so it is now in the mailbox with "role == "trash"", and remove all other mailbox ids.
When emptying the trash, clients SHOULD NOT destroy messages which are also in a mailbox other than trash. For those messages, they SHOULD just remove the Trash mailbox from the message.

The following extra _setError_ types are defined:

For *create*:

- "attachmentNotFound": At least one blob id given in an attachment doesn't exist. An extra _notFound_ property of type "String[]" MUST be included in the error object containing every _blobId_ referenced in _attachments_ that could not be found on the server.

- "maxQuotaReached": The user has reached a server-defined limit on their message storage quota.

For *update*:

- "tooManyKeywords": The change to the message's keywords would exceed a server-defined maximum.

### 4.6. importMessages

The _importMessages_ method adds [RFC5322] messages to a user's set of messages. The messages must first be uploaded as a file using the standard upload mechanism. It takes the following arguments:

- *accountId*: "String|null" The id of the account to use for this call. If "null", defaults to the primary account.

- *messages*: "String[MessageImport]" A map of creation id (client specified) to MessageImport objects

A *MessageImport* object has the following properties:

- *blobId*: "String" The id representing the raw [RFC5322] message (see the file upload section).

- *mailboxIds* "String[Boolean]" The ids of the mailbox(es) to assign this message to. At least one mailbox MUST be given.

- *keywords*: "String[Boolean]" (default: "{}") The keywords to apply to the message.

Each message to import is considered an atomic unit which may succeed or fail individually. Importing successfully creates a new message object from the data reference by the blobId and applies the given mailboxes and keywords.
The server MAY forbid two messages with the same exact [RFC5322] content, or even just with the same [RFC5322] Message-Id, to coexist within an account. In this case, it should reject attempts to import a message considered a duplicate with an "alreadyExists" SetError. A _messageId_ property of type "String" MUST be included on the error object with the id of the existing message.

If the _blobId_, _mailboxIds_, or _keywords_ properties are invalid (e.g. missing, wrong type, id not found), the server MUST reject the import with an "invalidProperties" SetError.

If the message cannot be imported because it would take the account over quota, the import should be rejected with a "maxQuotaReached" SetError.

If the blob referenced cannot be parsed as an [RFC5322] message, the server MUST reject the import with an "invalidMessage" SetError.

The response to _importMessages_ is called _messagesImported_. It has the following arguments:

- *accountId*: "String" The id of the account used for this call.
- *created*: "String[Message]" A map of the creation id to an object containing the _id_, _blobId_, _threadId_ and _size_ properties for each successfully imported Message.
- *notCreated*: "String[SetError]" A map of creation id to a SetError object for each Message that failed to be created. The possible errors are defined above.

The following errors may be returned instead of the _messagesImported_ response:

"accountNotFound": Returned if an _accountId_ was explicitly included with the request, but it does not correspond to a valid account.

"accountNotSupportedByMethod": Returned if the _accountId_ given corresponds to a valid account, but the account does not support this data type.

"accountReadOnly": Returned if the account has "isReadOnly == true".

"invalidArguments": Returned if one of the arguments is of the wrong type, or otherwise invalid. A "description" property MAY be present on the response object to help debug with an explanation of what the problem was.
4.7. copyMessages

The only way to move messages between two different accounts is to copy them using the _copyMessages_ method, then once the copy has succeeded, delete the original. It takes the following arguments:

- *fromAccountId*: "String|null" The id of the account to copy messages from. If "null", defaults to the primary account.
- *toAccountId*: "String|null" The id of the account to copy messages to. If "null", defaults to the primary account.
- *messages*: "String[MessageCopy]" A map of _creation id_ to a MessageCopy object.

A *MessageCopy* object has the following properties:

- *messageId*: "String" The id of the message to be copied in the "from" account.
- *mailboxIds*: "String[Boolean]" The ids of the mailboxes (in the "to" account) to add the copied message to. At least one mailbox MUST be given.
- *keywords*: "String[Boolean]" (default: "{}") The _keywords_ property for the copy.

The server MAY forbid two messages with the same exact [RFC5322] content, or even just with the same [RFC5322] Message-Id, to coexist within an account. If duplicates are allowed though, the "from" account may be the same as the "to" account to copy messages within an account.

Each message copy is considered an atomic unit which may succeed or fail individually. Copying successfully MUST create a new message object, with separate ids and mutable properties (e.g. mailboxes and keywords) to the original message.

The response to _copyMessages_ is called _messagesCopied_. It has the following arguments:

- *fromAccountId*: "String" The id of the account messages were copied from.
- *toAccountId*: "String" The id of the account messages were copied to.
o  *created*: "String[Message]|null" A map of the creation id to an object containing the _id_, _blobId_, _threadId_ and _size_ properties for each successfully copied Message.

o  *notCreated*: "String[SetError]|null" A map of creation id to a SetError object for each Message that failed to be copied, "null" if none.

The *SetError* may be one of the following types:

"alreadyExists": Returned if the server forbids duplicates and the message already exists in the target account. A _messageId_ property of type "String" MUST be included on the error object with the id of the existing message.

"notFound": Returned if the _messageId_ given can't be found.

"invalidProperties": Returned if the _mailboxIds_ or _keywords_ properties are invalid (e.g. missing, wrong type, id not found).

"maxQuotaReached": Returned if the user has reached their mail quota so the message cannot be copied.

The following errors may be returned instead of the _messagesCopied_ response:

"fromAccountNotFound": Returned if a _fromAccountId_ was explicitly included with the request, but it does not correspond to a valid account.

"toAccountNotFound": Returned if a _toAccountId_ was explicitly included with the request, but it does not correspond to a valid account.

"fromAccountNoMail": Returned if the _fromAccountId_ given corresponds to a valid account, but does not contain any mail data.

"toAccountNoMail": Returned if the _toAccountId_ given corresponds to a valid account, but does not contain any mail data.

"accountReadOnly": Returned if the "to" account has "isReadOnly == true".

"invalidArguments": Returned if one of the arguments is of the wrong type, or otherwise invalid. A "description" property MAY be present on the response object to help debug with an explanation of what the problem was.
5.  MessageSubmission

The MessageSubmission object represents the submission of a message for delivery to one or more recipients. A *MessageSubmission* object has the following properties:

- *id*: "String" (immutable; server-set) The id of the message submission.

- *identityId*: "String" (immutable) The id of the identity to associate with this submission.

- *messageId*: "String" (immutable) The id of the message to send. The message being sent does not have to be a draft, for example when "redirecting" an existing message to a different email address.

- *threadId*: "String" (immutable; server-set) The thread id of the message to send. This is set by the server to the _threadId_ property of the message referenced by the _messageId_.

- *envelope*: "Envelope|null" (immutable; default: "null") Information for use when sending via SMTP. An *Envelope* object has the following properties:
  - *mailFrom*: "Address" The email address to use as the return address in the SMTP submission, plus any parameters to pass with the MAIL FROM address. The JMAP server MAY allow the email to be the empty string. When a JMAP server performs a message submission, it MAY use the same id string for the [RFC3461] ENVID parameter and the MessageSubmission object id. Servers that do this MAY replace a client-provided value for ENVID with a server-provided value.
  - *rcptTo*: "Address[]" The email addresses to send the message to, and any RCPT TO parameters to pass with the recipient.

An *Address* object has the following properties:

- *email*: "String" The email address being represented by the object. This as a "Mailbox" as used in the Reverse-path or Forward-path of the MAIL FROM or RCPT TO command in [@!RFC5321]

- *parameters*: "Object|null" Any parameters to send with the email (either mail-parameter or rcpt-parameter as appropriate, as specified in [RFC5321]). If supplied, each key in the object is a parameter name, and the value either the parameter value (type "String") or if the parameter does not take a value
then "null". For both name and value, any xtext or unitext encodings are removed ([RFC3461], [RFC6533]) and JSON string encoding applied.

If the _envelope_ property is "null" or omitted on creation, the server MUST generate this from the referenced message as follows:

* **mailFrom**: The email in the _Sender_ header, if present, otherwise the _From_ header, if present, and no parameters. If multiple addresses are present in one of these headers, or there is more than one _Sender_/ _From_ header, the server SHOULD reject the message as invalid but otherwise MUST take the first email address in the last _Sender_/ _From_ header in the [RFC5322] version of the message. If the address found from this is not allowed by the identity associated with this submission, the _email_ property from the identity MUST be used instead.

* **rcptTo**: The deduplicated set of email addresses from the _To_, _Cc_ and _Bcc_ headers, if present, with no parameters for any of them.

* **sendAt**: "Date" (immutable; server-set) The date the message was/will be released for delivery. If the client successfully used [RFC4865] FUTURERELEASE with the message, this MUST be the time when the server will release the message; otherwise it MUST be the time the MessageSubmission was created.

* **undoStatus**: "String" (server-set) This represents whether the submission may be canceled. This is server set and MUST be one of the following values:

  * "pending": It MAY be possible to cancel this submission.

  * "final": The message has been relayed to at least one recipient in a manner that cannot be recalled. It is no longer possible to cancel this submission.

  * "canceled": The message submission was canceled and will not be delivered to any recipient.

On systems that do not support unsending, the value of this property will always be "final". On systems that do support canceling submission, it will start as "pending", and MAY transition to "final" when the server knows it definitely cannot recall the message, but MAY just remain "pending". If in pending state, a client can attempt to cancel the submission by setting this property to "canceled"; if the update succeeds, the
submission was successfully canceled and the message has not been
delivered to any of the original recipients.

- *deliveryStatus*: "String[DeliveryStatus]|null" (server-set) This
  represents the delivery status for each of the message recipients,
  if known. This property MAY not be supported by all servers, in
  which case it will remain "null". Servers that support it SHOULD
  update the MessageSubmission object each time the status of any of
  the recipients changes, even if some recipients are still being
  retried. This value is a map from the email address of each
  recipient to a _DeliveryStatus_ object. A *DeliveryStatus* object
  has the following properties:

  * *smtpReply*: "String" The SMTP reply string returned for this
    recipient when the server last tried to relay the message, or
    in a later DSN response for the message. This SHOULD be the
    response to the RCPT TO stage, unless this was accepted and the
    message as a whole rejected at the end of the DATA stage, in
    which case the DATA stage reply SHOULD be used instead. Multi-
    line SMTP responses should be concatenated to a single string
    as follows:

    + The hyphen following the SMTP code on all but the last line
      is replaced with a space.

    + Any prefix in common with the first line is stripped from
      lines after the first.

    + CRLF is replaced by a space.

    For example:

    550-5.7.1 Our system has detected that this message is
    550 5.7.1 likely spam, sorry.

    would become:

    **550** 5.7.1 Our system has detected that this message is likely spam, sorry.

    For messages relayed via an alternative to SMTP, the server MAY
    generate a synthetic string representing the status instead.
    If it does this, the string MUST be of the following form:

    + A 3-digit SMTP reply code, as defined in [RFC5321], section
      4.2.3.

    + Then a single space character.
+ Then an SMTP Enhanced Mail System Status Code as defined in [RFC3463], with a registry defined in [RFC5248].

+ Then a single space character.

+ Then an implementation-specific information string with a human readable explanation of the response.

* *delivered*: "String" Represents whether the message has been successfully delivered to the recipient. This MUST be one of the following values:

+ "queued": The message is in a local mail queue and status will change once it exits the local mail queues. The _smtpReply_ property may still change.

+ "yes": The message was successfully delivered to the mailbox of the recipient. The _smtpReply_ property is final.

+ "no": Message delivery to the recipient permanently failed. The _smtpReply_ property is final.

+ "unknown": The final delivery status is unknown, (e.g. it was relayed to an external machine and no further information is available). The _smtpReply_ property may still change if a DSN arrives.

Note, successful relaying to an external SMTP server SHOULD NOT be taken as an indication that the message has successfully reached the final mailbox. In this case though, the server MAY receive a DSN response, if requested. If a DSN is received for the recipient with Action equal to "delivered", as per [RFC3464] section 2.3.3, then the _delivered_ property SHOULD be set to "yes"; if the Action equals "failed", the property SHOULD be set to "no". Receipt of any other DSN SHOULD NOT affect this property. The server MAY also set this property based on other feedback channels.

* *displayed*: "String" Represents whether the message has been displayed to the recipient. This MUST be one of the following values:

+ "unknown": The display status is unknown. This is the initial value.

+ "yes": The receiveipient's system claims the message content has been displayed to the recipient. Note, there is no
guarantee that the recipient has noticed, read, or understood the content.

If an MDN is received for this recipient with Disposition-Type (as per [RFC3798] section 3.2.6.2) equal to "displayed", this property SHOULD be set to "yes". The server MAY also set this property based on other feedback channels.

- *dsnBlobIds*: "String[]" (server-set) A list of blob ids for DSNs received for this submission, in order of receipt, oldest first.

- *mdnBlobIds*: "String[]" (server-set) A list of blob ids for MDNs received for this submission, in order of receipt, oldest first.

JMAP servers MAY choose not to expose DSN and MDN responses as Message objects if they correlate to a MessageSubmission object. It SHOULD only do this if it exposes them in the _dsnBlobIds_ and _mdnBlobIds_ fields instead, and expects the user to be using clients capable of fetching and displaying delivery status via the MessageSubmission object.

For efficiency, a server MAY destroy MessageSubmission objects a certain amount of time after the message is successfully sent or it has finished retrying sending the message. For very basic SMTP proxies, this MAY be immediately after creation, as it has no way to assign a real id and return the information again if fetched later.

The following JMAP methods are supported:

5.1. getMessageSubmissions

Standard _getFoos_ method.

5.2. getMessageSubmissionUpdates

Standard _getFooUpdates_ method.

5.3. getMessageSubmissionList

Standard _getFooList_ method.

The *FilterCondition* object (optionally passed as the _filter_ argument) has the following properties, any of which may be omitted:

- *messageIds*: "String[]" The MessageSubmission _messageId_ property must be in this list to match the condition.
*threadIds*: "String[]" The MessageSubmission _threadId_ property must be in this list to match the condition.

*undoStatus*: "String" The MessageSubmission _undoStatus_ property must be identical to the value given to match the condition.

*before*: "Date" The _sendAt_ property of the MessageSubmission object must be before this date to match the condition.

*after*: "Date" The _sendAt_ property of the MessageSubmission object must be after this date to match the condition.

A MessageSubmission object matches the filter if and only if all of the given conditions given match. If zero properties are specified, it is automatically "true" for all objects.

The following properties MUST be supported for sorting:

- "messageId"
- "threadId"
- "sentAt"

**5.4. getMessageSubmissionListUpdates**

Standard _getFooListUpdates_ method.

**5.5. setMessageSubmissions**

Standard _setFoos_ method, with the following two extra arguments:

- *onSuccessUpdateMessage*: "String[Message]|null" A map of _MessageSubmission id_ to an object containing properties to update on the Message object referenced by the MessageSubmission if the create/update/destroy succeeds. (For references to MessageSubmission creations, this is equivalent to a back reference so the id will be the creation id prefixed with a "#".)

- *onSuccessDestroyMessage*: "String[]|null" A list of _MessageSubmission ids_ for which the message with the corresponding messageId should be destroyed if the create/update/destroy succeeds. (For references to MessageSubmission creations, this is equivalent to a back reference so the id will be the creation id prefixed with a "#".)

A single implicit _setMessages_ call MUST be made after all MessageSubmission create/update/destroy requests have been processed
to perform any changes requested in these two arguments. The _messagesSet_ response MUST be returned after the _messageSubmissionsSet_ response.

A message is sent by creating a MessageSubmission object. When processing each create, the server must check that the message is valid, and the user has sufficient authorization to send it. If the creation succeeds, the message will be sent to the recipients given in the envelope _rcptTo_ parameter. The server MUST remove any _Bcc_ header present on the message during delivery. The server MAY add or remove other headers from the submitted message, or make further alterations in accordance with the server's policy during delivery.

If the referenced message is destroyed at any point after the MessageSubmission object is created, this MUST NOT change the behaviour of the message submission (i.e. it does not cancel a future send).

Similarly, destroying a MessageSubmission object MUST NOT affect the deliveries it represents. It purely removes the record of the message submission. The server MAY automatically destroy MessageSubmission objects after a certain time or in response to other triggers, and MAY forbid the client from manually destroying MessageSubmission objects.

The following extra _SetError_ types are defined:

For *create*:

- "tooLarge" - The message size is larger than the server supports. A _maxSize_ "Number" property MUST be present on the SetError specifying the maximum size of a message that may be sent, in bytes.

- "tooManyRecipients" - The envelope (supplied or generated) has more recipients than the server allows. A _maxRecipients_ "Number" property MUST be present on the SetError specifying the maximum number of allowed recipients.

- "noRecipients" - The envelope (supplied or generated) does not have any rcptTo emails.

- "invalidRecipients" - The _rcptTo_ property of the envelope (supplied or generated) contains at least one rcptTo value which is not a valid email for sending to. An _invalidEmails_ "String[]" property MUST be present on the SetError, which is a list of the invalid emails.
o "notPermittedFrom" - The server does not permit the user to send a message with the From header of the message to be sent.

o "notPermittedToSend" - The user does not have permission to send at all right now for some reason. A _description_ "String" property MAY be present on the SetError object to display to the user why they are not permitted.

o "messageNotFound" - The _messageId_ is not a valid id for a message in the account.

o "invalidMessage" - The message to be sent is invalid in some way. The SetError SHOULD contain a property called _properties_ of type "String[]" that lists *all* the properties of the Message that were invalid.

For *update*:

o "cannotUnsend": The client attempted to update the _undoStatus_ of a valid MessageSubmission object from "pending" to "canceled", but the message cannot be unsent.

For *destroy*:

o "forbidden": The server does not allow clients to destroy MessageSubmission objects.

6. Identities

An *Identity* object stores information about an email address (or domain) the user may send from. It has the following properties:

o *id*: "String" (immutable; server-set) The id of the identity.

o *name*: "String" (default: """") The "From" _name_ the client SHOULD use when creating a new message from this identity.

o *email*: "String" The "From" email address the client MUST use when creating a new message from this identity. This property is immutable. The "email" property MAY alternatively be of the form "*@example.com", in which case the client may use any valid email address ending in "@example.com".

o *replyTo*: "Emailer[]|null" (default: "null") The Reply-To value the client SHOULD set when creating a new message from this identity.
o  *bcc*: "Emailer[]|null" (default: "null") The Bcc value the client SHOULD set when creating a new message from this identity.

o  *textSignature*: "String" (default: """") Signature the client SHOULD insert into new rich-text messages that will be sending from this identity. Clients MAY ignore this and/or combine this with a client-specific signature preference.

o  *htmlSignature*: "String" (default: """") Signature the client SHOULD insert into new HTML messages that will be sending from this identity. This text MUST be an HTML snippet to be inserted into the "<body></body>" section of the new email. Clients MAY ignore this and/or combine this with a client-specific signature preference.

o  *mayDelete*: "Boolean" (server-set) Is the user allowed to delete this identity? Servers may wish to set this to "false" for the user's username or other default address.

Multiple identities with the same email address MAY exist, to allow for different settings the user wants to pick between (for example with different names/signatures).

The following JMAP methods are supported:

6.1. getIdenties

Standard _getFoos_ method. The _ids_ argument may be "null" to fetch all at once.

6.2. getIdentityUpdates

Standard _getFooUpdates_ method.

6.3. setIdenties

Standard _setFoos_ method. The following extra _SetError_ types are defined:

For *create*:

-  "maxQuotaReached": The user has reached a server-defined limit on the number of identities.

-  "emailNotAllowed": The user is not allowed to send from the address given as the _email_ property of the identity.

For *destroy*:
7. SearchSnippets

When doing a search on a "String" property, the client may wish to show the relevant section of the body that matches the search as a preview instead of the beginning of the message, and to highlight any matching terms in both this and the subject of the message. Search snippets represent this data.

A *SearchSnippet* object has the following properties:

- *messageId*: "String" The message id the snippet applies to.
- *subject*: "String|null" If text from the filter matches the subject, this is the subject of the message HTML-escaped, with matching words/phrases wrapped in "<mark></mark>" tags. If it does not match, this is "null".
- *preview*: "String|null" If text from the filter matches the plain-text or HTML body, this is the relevant section of the body (converted to plain text if originally HTML), HTML-escaped, with matching words/phrases wrapped in "<mark></mark>" tags, up to 256 characters long. If it does not match, this is "null".
- *attachments*: "String|null" If text from the filter matches the text extracted from an attachment, this is the relevant section of the attachment (converted to plain text), with matching words/phrases wrapped in "<mark></mark>" tags, up to 256 characters long. If it does not match, this is "null".

It is server-defined what is a relevant section of the body for preview. If the server is unable to determine search snippets, it MUST return "null" for both the _subject_, _preview_ and _attachments_ properties.

Note, unlike most data types, a SearchSnippet DOES NOT have a property called "id".

The following JMAP method is supported:

7.1. getSearchSnippets

To fetch search snippets, make a call to "getSearchSnippets". It takes the following arguments:
o  *accountId*: "String|null" The id of the account to use for this call. If "null", defaults to the primary account.

o  *messageIds*: "String[]" The list of ids of messages to fetch the snippets for.

o  *filter*: "FilterCondition|FilterOperator|null" The same filter as passed to getMessageList; see the description of this method for details.

The response to "getSearchSnippets" is called "searchSnippets". It has the following arguments:

o  *accountId*: "String" The id of the account used for the call.

o  *filter*: "FilterCondition|FilterOperator|null" Echoed back from the call.

o  *list*: "SearchSnippet[]" An array of SearchSnippet objects for the requested message ids. This may not be in the same order as the ids that were in the request.

o  *notFound*: "String[]|null" An array of message ids requested which could not be found, or "null" if all ids were found.

Since snippets are only based on immutable properties, there is no state string or update mechanism needed.

The following errors may be returned instead of the _searchSnippets_ response:

"accountNotFound": Returned if an _accountId_ was explicitly included with the request, but it does not correspond to a valid account.

"accountNotSupportedByMethod": Returned if the _accountId_ given corresponds to a valid account, but the account does not support this data type.

"requestTooLarge": Returned if the number of _messageIds_ requested by the client exceeds the maximum number the server is willing to process in a single method call.

"cannotDoFilter": Returned if the server is unable to process the given _filter_ for any reason.

"invalidArguments": Returned if the request does not include one of the required arguments, or one of the arguments is of the wrong type, or otherwise invalid. A "description" property MAY be present on the
response object to help debug with an explanation of what the problem was.

8. Vacation Response

The *VacationResponse* object represents the state of vacation-response related settings for an account. It has the following properties:

- **id**: "String" (immutable) The id of the object. There is only ever one vacation response object, and its id is "singleton".

- **isEnabled**: "Boolean" Should a vacation response be sent if a message arrives between the _fromDate_ and _toDate_?

- **fromDate**: "Date|null" If _isEnabled_ is "true", the date/time after which messages that arrive should receive the user's vacation response, in UTC. If "null", the vacation response is effective immediately.

- **toDate**: "Date|null" If _isEnabled_ is "true", the date/time after which messages that arrive should no longer receive the user's vacation response, in UTC. If "null", the vacation response is effective indefinitely.

- **subject**: "String|null" The subject that will be used by the mail sent in response to messages when the vacation response is enabled. If null, an appropriate subject SHOULD be set by the server.

- **textBody**: "String|null" The plain text part of the message to send in response to messages when the vacation response is enabled. If this is "null", when the vacation message is sent a plain-text body part SHOULD be generated from the _htmlBody_ but the server MAY choose to send the response as HTML only.

- **htmlBody**: "String|null" The HTML message to send in response to messages when the vacation response is enabled. If this is "null", when the vacation message is sent an HTML body part MAY be generated from the _textBody_, or the server MAY choose to send the response as plain-text only.

The following JMAP methods are supported:
8.1. getVacationResponse

Standard _getFoos_ method.

There MUST only be exactly one VacationResponse object in an account. It MUST have the id "singleton".

8.2. setVacationResponse

Standard _setFoos_ method. The following extra _SetError_ types are defined:

For *create* or *destroy*:

- "singleton": This is a singleton object, so you cannot create another one or destroy the existing one.

9. Security considerations

All security considerations of JMAP {TODO: insert RFC ref} apply to this specification.

10. References

10.1. Normative References


Internet-Draft                  JMAP Mail                   October 2017

[RFC3461]  Moore, K., "Simple Mail Transfer Protocol (SMTP) Service
            Extension for Delivery Status Notifications (DSNs)",
            RFC 3461, DOI 10.17487/RFC3461, January 2003,

[RFC3463]  Vaudreuil, G., "Enhanced Mail System Status Codes",
            RFC 3463, DOI 10.17487/RFC3463, January 2003,

            for Delivery Status Notifications", RFC 3464,
            DOI 10.17487/RFC3464, January 2003,

[RFC3629]  Yergeau, F., "UTF-8, a transformation format of ISO
            10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November

            Disposition Notification", RFC 3798, DOI 10.17487/RFC3798,

[RFC4648]  Josefsson, S., "The Base16, Base32, and Base64 Data
            Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006,

            Extension for Future Message Release", RFC 4865,
            DOI 10.17487/RFC4865, May 2007,

            Mail System Status Codes", BCP 138, RFC 5248,
            DOI 10.17487/RFC5248, June 2008,

            DOI 10.17487/RFC5321, October 2008,

            DOI 10.17487/RFC5322, October 2008,

            RFC 5788, DOI 10.17487/RFC5788, March 2010,


10.2. URIs

[1] server.html


Author's Address

Neil Jenkins
FastMail
Level 2, 114 William St
Melbourne VIC 3000
Australia

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