Abstract

This document specifies a data model for synchronizing task 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 2 August 2021.

Copyright Notice

Copyright (c) 2021 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.

Table of Contents

1. Introduction .............................................. 3
### Table of Contents

1. **Notational Conventions** ............................................. 3
2. **Terminology** .................................................... 3
3. **Data Model Overview** ............................................ 4
4. **Addition to the Capabilities Object** .......................... 4
   4.1. urn:ietf:params:jmap:tasks .................................... 4
2. **Principals** ..................................................... 5
   2.1. Principal Capability urn:ietf:params:jmap:tasks .............. 5
3. **Assignee Identities** ............................................ 6
   3.1. AssigneeIdentity/get ......................................... 6
   3.2. AssigneeIdentity/changes .................................... 6
   3.3. AssigneeIdentity/set ......................................... 6
4. **TaskLists** ...................................................... 6
   4.1. TaskList/get .................................................. 9
   4.2. TaskList/changes ............................................ 9
   4.3. TaskList/set ................................................ 9
5. **Tasks** .......................................................... 9
   5.1. Additional JSCalendar properties ............................ 10
      5.1.1. mayInviteSelf ........................................... 10
      5.1.2. mayInviteOthers ......................................... 11
      5.1.3. hideAttendees ........................................... 11
      5.1.4. relatedTo ............................................... 11
   5.2. Properties similar in JMAP for Calendar ................. 11
   5.3. Task/get .................................................... 11
   5.4. Task/changes ............................................... 12
   5.5. Task/set .................................................. 12
   5.6. Task/copy ................................................ 12
   5.7. Task/query ................................................ 12
   5.8. Task/queryChanges ......................................... 12
6. **Task Notifications** ............................................. 12
   6.1. Object Properties .......................................... 12
   6.2. TaskNotification/get ....................................... 13
   6.3. TaskNotification/changes ................................... 14
   6.4. TaskNotification/set ....................................... 14
   6.5. TaskNotification/query .................................... 14
      6.5.1. Filtering ............................................... 14
      6.5.2. Sorting ................................................ 14
   6.6. TaskNotification/queryChanges ............................. 14
7. **Security Considerations** ..................................... 14
8. **IANA Considerations** .......................................... 15
   8.1. JMAP Capability Registration for "tasks" ..................... 15
   8.2. JSCalendar Property Registrations ........................ 15
9. **Normative References** ......................................... 15
10. **Informative References** ...................................... 16

Authors' Addresses .................................................. 16
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.

JMAP for Calendars ([I-D.ietf-jmap-calendars]) 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.

While CalDAV defines access to tasks, JMAP for Calendars does not. This specification fills this gap and defines a data model for synchronizing task data between a client and a server using JMAP. It is built upon JMAP for Calendars and reuses most of its definitions. For better readability this document only outlines differences between this specification and JMAP for Calendars. If not stated otherwise, the same specifics that apply to Calendar, CalendarEvent and CalendarEventNotification objects as defined in the aforementioned specification also apply to similar data types introduced in this specification.

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 terms ParticipantIdentity, TaskList, Task and TaskNotification are used to refer to the data types defined in this document and instances of those data types.
1.3. Data Model Overview

Similar to JMAP for Calendar, an Account (see [RFC8620], Section 1.6.2) contains zero or more TaskList objects, which is a named collection of Tasks belonging to a Principal (see [I-D.jenkins-jmap-sharing] Section XXX). Task lists can also provide defaults, such as alerts and a color to apply to tasks in the calendar. Clients commonly let users toggle visibility of tasks belonging to a particular task list on/off. Servers may allow a task to belong to multiple TaskLists within an account.

A Task is a representation of a single task or recurring series of Tasks in JSTask [I-D.ietf-calext-jscalendar] format. Recurrence rules and alerts as defined in JMAP for Calendars (see [I-D.ietf-jmap-calendars] Section XXX) apply.

Just like the CalendarEventNotification objects (see [I-D.ietf-jmap-calendars] Section XXX), TaskNotification objects keep track of the history of changes made to a task by other users. Similarly, the ShareNotification type (see [I-D.jenkins-jmap-sharing] Section XXX) notifies the user when their access to another user’s calendar is granted or revoked.

1.4. 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.

1.4.1. urn:ietf:params:jmap:tasks

This represents support for the TaskList, Task and TaskNotification 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:

* *sharesActAs*: "String" This MUST be one of:
  - "self" - sharees act as themselves when using tasks in this account.
  - "secretary"- sharees act as the principal to which this account belongs.
* *maxTaskListsPerTask*: "UnsignedInt|null" The maximum number of TaskLists (see Section XXX) that can be assigned to a single Task object (see Section XXX). This MUST be an integer \( \geq 1 \), or null for no limit (or rather, the limit is always the number of TaskLists in the account).

* *minDateTime*: "LocalDate" The earliest date-time the server is willing to accept for any date stored in a Task.

* *maxDateTime*: "LocalDate" The latest date-time the server is willing to accept for any date stored in a Task.

* *maxExpandedQueryDuration*: "Duration" The maximum duration the user may query over when asking the server to expand recurrences.

* *maxAssigneesPerTask*: "Number|null" The maximum number of assignees a single task may have, or null for no limit.

* *mayCreateTaskList*: "Boolean" If true, the user may create a task list in this account.

2. Principals

For systems that also support JMAP Sharing [RFC XXX], the tasks capability is used to indicate that this principal may be used with tasks.

2.1. Principal Capability urn:ietf:params:jmap:tasks

A "urn:ietf:params:jmap:tasks" 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:tasks" capability that contains the task 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.

* *account*: "Account|null" The JMAP Account object corresponding to the accountId, null if none.

* *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 JSTask (see [I-D.ietf-calext-jscalendaret], Section 4.4.5).
3. Assignee Identities

An AssigneeIdentity stores information about a URI that represents the user within that account in an event’s assignees. It has the following properties:

* `*id*`: "Id" (immutable; server-set) The id of the AssigneeIdentity.
* `*name*`: "String" (default: "") The display name of the assignee to use when adding this assignee to a task, e.g. "Jane 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.

An assignee in an task corresponds to an AssigneeIdentity 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. AssigneeIdentity/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. AssigneeIdentity/changes

This is a standard "/changes" method as described in [RFC8620], Section 5.2.

3.3. AssigneeIdentity/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. TaskLists

A TaskList is a named collection of tasks. All tasks are associated with at least one TaskList.
A *TaskList* object has the following properties:

* *id*: "Id" (immutable; server-set) The id of the task list.
* *role*: "String|null" (default: null) Denotes the task list has a special purpose. This MUST be one of the following:
  - "inbox": This is the principal’s default task list;
  - "trash": This task list holds messages the user has discarded;
* *name*: "String" The user-visible name of the task list. 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 task list, 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 task list.

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 background.

* *sortOrder*: "UnsignedInt" (default: 0) Defines the sort order of task lists 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 task list with a lower order should be displayed before a list with a higher order in any list of task lists in the client’s UI. Task lists 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 task list in their client? This SHOULD default to false for task lists in shared accounts the user has access to and true for any new task list created by the user themself.
If false, the task list 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 task list’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 task lists, it should be displayed if isVisible is true for any of those task lists.

* *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 task lists; 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 task lists; 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 task list 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 task list 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 task list 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.

The user is an *owner* for a task if the Task object has an "assignee" property, and one of the Participant objects both:

1. Has the "chair" role.
2. Corresponds to one of the user’s AssigneeIdentity objects in the account.

A task has no owner if its assignee property is null or omitted.

TODO currently disregarding "myRights"

4.1. TaskList/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.

TODO add part about rights properties.

4.2. TaskList/changes

This is a standard "/changes" method as described in [RFC8620], Section 5.2.

4.3. TaskList/set

This is the "Calendar/set" method as described in [I-D.ietf-calext-jscalendar], Section XXX.

TODO copy+paste from "Calendar/set" and replace "onDestroyRemoveEvents" by "onDestroyRemoveTasks" (and "calendarHasEvent").

5. Tasks

A *Task* object contains information about a task, or recurring series of tasks. It is a JSTask object, as defined in [I-D.ietf-calext-jscalendar], with the following additional properties:

* *id*: "Id" The id of the Task. This property is immutable. The id uniquely identifies a JSTask with a particular "uid" and "recurrenceId" within a particular account.

* *taskListIds*: "Id[Boolean]" The set of TaskList ids this task belongs to. An task MUST belong to one or more TaskLists at all times (until it is destroyed). The set is represented as an object, with each key being a _TaskList id_. The value for each key in the object MUST be "true".
* `isDraft`: "Boolean" If true, this task is to be considered a draft. The server will not send any 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 task.

If set, server will convert to the task’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 tasks (tasks without a time zone) will be interpreted as per the time zone given as a Task/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 task. 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 tasks will be interpreted as per the time zone given as a Task/get argument.

5.1. Additional JSCalendar properties

This document defines four 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.

5.1.4. relatedTo

Type: "Id[String]|null" (default: null)

A map of task ids to relations. Relation SHOULD be one of: - "blockedBy": Blocked by task with id. - "clonedBy": Task with id was cloned from this issue. - "duplicatedBy": Task with id is a duplicate of this issue. - "causedBy": Task with id was the cause for this task. - "relatesTo": Task with id is related. - "childOf": Task with id is parent.

5.2. Properties similar in JMAP for Calendar

Attachments, per-user properties, recurrences and updates to recurrences are described in [I-D.ietf-jmap-calendars], Section XXX.

5.3. Task/get

This is the "CalendarEvent/get" method as described in [I-D.ietf-jmap-calendars], Section XXX.

TODO redefine this here. Similar to "TaskList/get" we only need to replace a few definitions. For example, replace "reduceParticipants" with "reduceAssignees". Copy+Paste most of the stuff.
5.4. Task/changes

This is a standard "/changes" method as described in [RFC8620], Section 5.2.

5.5. Task/set

This is the "CalendarEvent/set" method as described in [I-D.ietf-jmap-calendars], Section XXX.

TODO copy+paste most stuff from "CalendarEvent/set". It should be fine to just reference patching.

5.6. Task/copy

This is a standard "/copy" method as described in [RFC8620], Section 5.4.

5.7. Task/query

This is the "CalendarEvent/query" method as described in [I-D.ietf-jmap-calendars], Section XXX.

TODO copy+paste most stuff from "CalendarEvent/query". Mainly filtering should be different.

5.8. Task/queryChanges

This is a standard "/queryChanges" method as described in [RFC8620], Section 5.6.

6. Task Notifications

The TaskNotification data type records changes made by external entities to tasks in calendars the user is subscribed to. Notifications are stored in the same Account as the Task that was changed.

This is the same specification as the CalendarEventNotification object from [I-D.ietf-jmap-calendars], Section XXX. Only the object properties differ slightly and are therefore fully described in this document.

6.1. Object Properties

The *TaskNotification* object has the following properties:

* *id*: "String" The id of the TaskNotification.
* *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 principal corresponding to the person who made the change, if any. This will be null if the change was due to receiving 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

* *taskId*: "String" The id of the Task that this notification is about.

* *isDraft*: "Boolean" (created/updated only) Is this event a draft?

* *event*: "JSTask" 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.

6.2. TaskNotification/get

This is a standard "/get" method as described in [RFC8620], Section 5.1.
6.3. TaskNotification/changes

This is a standard "/changes" method as described in [RFC8620], Section 5.2.

6.4. TaskNotification/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.

6.5. TaskNotification/query

This is a standard "/query" method as described in [RFC8620], Section 5.5.

6.5.1. 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.

* *taskId*: "Id[]|null" A list of task ids. The taskId property of the notification must be in this list to match the condition.

6.5.2. Sorting

The "created" property MUST be supported for sorting.

6.6. TaskNotification/queryChanges

This is a standard "/queryChanges" method as described in [RFC8620], Section 5.6.

7. Security Considerations

All security considerations of JMAP for Calendars [I-D.ietf-jmap-calendars] apply to this specification.
8. IANA Considerations

8.1. JMAP Capability Registration for "tasks"

TODO Actually register

IANA will register the "tasks" JMAP Capability as follows:

Capability Name: "urn:ietf:params:jmap:tasks"

Specification document: this document

Intended use: common

Change Controller: IETF

Security and privacy considerations: this document, Section XXX

8.2. JSCalendar Property Registrations

All IANA registrations for JSTask are described in JMAP for Calendars
[I-D.ietf-jmap-calendars].

9. Normative References

[I-D.ietf-calext-jscalendar]

[I-D.ietf-jmap-calendars]

[I-D.jenkins-jmap-sharing]


10. Informative References


Authors’ Addresses

Joris Baum (editor)
audriga
Durlacher Allee 47
76131 Karlsruhe
Germany

Email: joris@audriga.com
URI:   https://www.audriga.com

Hans-Joerg (editor)
audriga
Durlacher Allee 47
76131 Karlsruhe
Germany

Email: hans-joerg@audriga.com
URI:   https://www.audriga.com
Abstract

The PARTIAL extension of the Internet Message Access Protocol (RFC 3501/RFC 9051) allows clients to limit the number of search results returned, as well as to perform incremental (paged) searches. This also helps servers to optimize resource usage when performing searches.

This document extends PARTIAL SEARCH return option originally specified in RFC 5267. It also clarifies some interactions between RFC 5267 and RFC 4731/RFC 9051.

This document also describes the MESSAGELIMIT extension for announcing a limit on the number of messages that can be processed in a single FETCH/SEARCH/STORE/COPY/MOVE command.

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 11 November 2022.
Copyright Notice

Copyright (c) 2022 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 Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

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

Table of Contents

1. Introduction and Overview .................................. 3
2. Document Conventions ....................................... 3
3. The PARTIAL extension ...................................... 3
   3.1. Incremental SEARCH and partial results .................. 4
   3.2. Interaction between PARTIAL, MIN, MAX and SAVE SEARCH return options .................................. 5
   3.3. Extension to UID FETCH ................................... 6
   3.4. Use of PARTIAL and CONDSTORE IMAP extensions together ..... 7
4. The MESSAGELIMIT extension ................................. 7
   4.1. Returning limits on the number of messages processed in a single SEARCH/FETCH/STORE/COPY/MOVE command .......... 8
   4.2. Interaction with SORT and THREAD extensions ............. 10
5. Formal syntax ................................................ 10
6. Security Considerations .................................... 11
7. IANA Considerations ........................................ 12
   7.1. Changes/additions to the IMAP4 capabilities registry ... 12
8. Acknowledgments ............................................. 12
9. References .................................................. 12
   9.1. Normative References .................................... 12
   9.2. Informative References ................................. 13
1. Introduction and Overview

This document defines an extension to the Internet Message Access Protocol [RFC3501] for performing incremental searches and fetches. This extension is compatible with both IMAP4rev1 [RFC3501] and IMAP4rev2 [RFC9051].

The PARTIAL extension of the Internet Message Access Protocol (RFC 3501/RFC 9051) allows clients to limit the number of search results returned, as well as to perform incremental (paged) searches. This also helps servers to optimize resource usage when performing searches.

This document extends PARTIAL SEARCH return option originally specified in RFC 5267. It also clarifies some interactions between RFC 5267 and RFC 4731/RFC 9051.

This document also describes the MESSAGELIMIT extension for announcing a limit on the number of messages that can be processed in a single FETCH/SEARCH/STORE/COPY/MOVE command.

2. 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.

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.

Other capitalised words are IMAP keywords [RFC3501][RFC9051] or keywords from this document.

3. The PARTIAL extension

An IMAP server advertises support for the PARTIAL extension by including "PARTIAL" capability in the CAPABILITY response/response code.
Clients that implement support for PARTIAL extension MUST also support the MESSAGELIMIT response code (see Section 4).

3.1. Incremental SEARCH and partial results

The PARTIAL search return option causes the server to provide in an ESEARCH response a subset of the results denoted by the sequence range given as the mandatory argument. The first result (message with the lowest matching UID) is 1; thus, the first 500 results would be obtained by a return option of "PARTIAL 1:500", and the second 500 by "PARTIAL 501:1000". This intentionally mirrors message sequence numbers.

It is also possible to direct the server to start SEARCH from the latest matching (with the highest UID) message. This can be done by prepending "-" to the index. For example -1 is the last message, -2 is next to the last and so on. Using this syntax helps server implementations to optimize their SEARCHes.

A single command MUST NOT contain more than one PARTIAL or ALL search return option -- that is, either one PARTIAL, one ALL, or neither PARTIAL nor ALL is allowed.

For SEARCH results, the entire result list MUST be ordered in mailbox order, that is, in UID or message sequence number order.

Where a PARTIAL search return option references results that do not exist, by using a range which starts or ends higher (or lower) than the current number of results, then the server returns the results that are in the set. This yields a PARTIAL return data item that has, as payload, the original range and a potentially missing set of results that may be shorter than the extent of the range. If the whole range references results that do not exist, a special value "NIL" is returned by the server instead of the sequence set.

Clients need not request PARTIAL results in any particular order. Because mailboxes may change, clients might wish to use PARTIAL in combination with UPDATE (see [RFC5267] if the server also advertises CONTEXT=SEARCH capability, especially if the intent is to walk a large set of results; however, these return options do not interact -- the UPDATE will provide notifications for all matching results.
// Let’s assume that the A01 SEARCH without PARTIAL would return
// 23764 results.
C: A01 UID SEARCH RETURN (PARTIAL -1:-100) UNDELETED
   UNKEYWORD $Junk
S: * ESEARCH (TAG "A01") UID PARTIAL (-1:-100 ...) 
   // 100 most recent results in set syntax elided.
S: A01 OK Completed.

// Let’s assume that the A02 SEARCH without PARTIAL would return
// 23764 results.
C: A02 UID SEARCH RETURN (PARTIAL 23500:24000) UNDELETED
   UNKEYWORD $Junk
C: A03 UID SEARCH RETURN (PARTIAL 1:500) UNDELETED
   UNKEYWORD $Junk
C: A04 UID SEARCH RETURN (PARTIAL 24000:24500) UNDELETED
   UNKEYWORD $Junk
S: * ESEARCH (TAG "A02") UID PARTIAL (23500:24000 ...) 
   // 264 results in set syntax elided, 
   // this spans the end of the results.
S: A02 OK Completed.
S: * ESEARCH (TAG "A03") UID PARTIAL (1:500 ...) 
   // 500 results in set syntax elided.
S: A03 OK Completed.
S: * ESEARCH (TAG "A04") UID PARTIAL (24000:24500 NIL)
   // No results are present, this is beyond the end of the results.
S: A04 OK Completed.

3.2. Interaction between PARTIAL, MIN, MAX and SAVE SEARCH return
     options

This section only applies if the server advertises PARTIAL IMAP
capability or CONTEXT=SEARCH [RFC5267], together with ESEARCH
[RFC4731] and/or IMAP4rev2"[RFC9051].

The SAVE result option doesn’t change whether the server would return
items corresponding to PARTIAL SEARCH result options.

As specified in Section 3.1, it is an error to specify both PARTIAL
and ALL result options in the same SEARCH command.

When the SAVE result option is combined with the PARTIAL result
option, and none of MIN/MAX/COUNT result options is present, the
corresponding PARTIAL is returned, and the "$" marker would contain
all messages returned by the PARTIAL result option.

When the SAVE + PARTIAL result options are combined with the MIN or
the MAX result option, and the COUNT result option is absent, the
corresponding PARTIAL result and MIN/MAX is returned (if the search
result is not empty), and the "$" marker would contain all messages returned by the PARTIAL result option + the corresponding MIN/MAX message.

If the SAVE + PARTIAL result options are combined with both MIN and MAX result options, and the COUNT result options is absent, the PARTIAL, MIN and MAX are returned (if the search result is not empty), and the "$" marker would contain all messages returned by the PARTIAL result option plus MIN and MAX messages.

If the SAVE + PARTIAL result options are combined with the COUNT result option, the PARTIAL and COUNT are returned, and 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.

<table>
<thead>
<tr>
<th>Combination of Result option</th>
<th>&quot;$&quot; marker value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAVE PARTIAL</td>
<td>PARTIAL</td>
</tr>
<tr>
<td>SAVE PARTIAL MIN</td>
<td>PARTIAL &amp; MIN</td>
</tr>
<tr>
<td>SAVE PARTIAL MAX</td>
<td>PARTIAL &amp; MAX</td>
</tr>
<tr>
<td>SAVE PARTIAL MIN MAX</td>
<td>PARTIAL &amp; MIN &amp; MAX</td>
</tr>
<tr>
<td>SAVE PARTIAL COUNT [m]</td>
<td>all found messages</td>
</tr>
</tbody>
</table>

Table 1

where ' [m]' means optional "MIN" and/or "MAX"

3.3. Extension to UID FETCH

The PARTIAL extension also extends the UID FETCH command with a PARTIAL FETCH modifier. The PARTIAL FETCH modifier has the same syntax as the PARTIAL SEARCH result option. Presence of the PARTIAL FETCH modifier instructs the server to only return FETCH results for messages in the specified range. It is useful when the sequence-set (first) parameter to the UID FETCH command includes unknown number of messages.
// Returning information for the last 3 messages in the UID range
C: 10 UID FETCH 25900:26600 (UID FLAGS) (PARTIAL -1:-3)
S: * 12888 FETCH (FLAGS (\Seen) UID 25996)
S: * 12889 FETCH (FLAGS (\Flagged \Answered) UID 25997)
S: * 12890 FETCH (FLAGS () UID 26600)
S: 10 OK FETCH completed

// Returning information for the first 5 messages in the UID range
C: 11 UID FETCH 25900:26600 (UID FLAGS) (PARTIAL 1:5)
S: * 12591 FETCH (FLAGS (\Seen) UID 25900)
S: * 12592 FETCH (FLAGS (\Flagged) UID 25902)
S: * 12593 FETCH (FLAGS (\Answered) UID 26310)
S: * 12594 FETCH (FLAGS () UID 26311)
S: * 12595 FETCH (FLAGS (\Answered) UID 26498)
S: 11 OK FETCH completed

3.4. Use of PARTIAL and CONDSTORE IMAP extensions together

This section is informative.

The PARTIAL FETCH modifier can be combined with the CHANGEDSINCE FETCH modifier.

// Returning information for the last 30 messages in the UID range
// that have any flag/keyword modified since modseq 98305
C: 101 UID FETCH 25900:26600 (UID FLAGS) (PARTIAL -1:-30 CHANGEDSINCE 98305)
S: * 12888 FETCH (FLAGS (\Flagged \Answered) MODSEQ (98306) UID 25997)
S: * 12890 FETCH (FLAGS () MODSEQ (98312) UID 26600)
S: 101 OK FETCH completed

The above example causes the server to first select the last 30 messages and then only return flag changes for subset of these messages which have MODSEQ higher than 98305.

Note that the order of PARTIAL and CHANGEDSINCE FETCH modifiers in the UID FETCH command is not important, i.e. the above example can also use "UID FETCH 25900:26600 (UID FLAGS) (CHANGEDSINCE 98305 PARTIAL -1:-30)" command and it would result in the same responses.

4. The MESSAGELIMIT extension

An IMAP server advertises support for the MESSAGELIMIT extension by including "MESSAGELIMIT=<limit>" capability in the CAPABILITY response/response code, where "<limit>" is a positive integer that conveys the maximum number of messages that can be processed in a single SEARCH/FETCH/STORE/COPY/MOVE command.
4.1. Returning limits on the number of messages processed in a single
SEARCH/FETCH/STORE/COPY/MOVE command

// Do we need a way to specify SEARCH criterion for "all UIDs after"
// or "all UIDs before" a specific UID?

If a server implementation doesn’t allow more than <N> messages to be
operated on by a single SEARCH/FETCH/STORE/COPY/MOVE command, it MUST
return the MESSAGELIMIT response code defined below:

MESSAGELIMIT The server doesn’t allow more than <N> messages to be operated
on by a single SEARCH/FETCH/STORE/COPY/MOVE command. The
lowest processed UID is <LastUID>. The client needs to repeat
the operation for remaining messages, if required.

In the following example the <N> value is 1000 and the lowest
processed UID <LastUID> is 23221.

C: 03 FETCH 10000:14589 (UID FLAGS)
S: * 14589 FETCH (FLAGS (\Seen) UID 25000)
S: * 14588 FETCH (FLAGS (\Answered) UID 24998)
S: ... further 997 fetch responses
S: * 13590 FETCH (FLAGS () UID 23221)
S: 03 OK [MESSAGELIMIT 1000 23221] FETCH completed with 1000 partial
results

In the following example the client searches for UNDELETED UIDs
between 22000:25000. The total number of matching messages
exceeds the server’s published 1000 messages limit.

C: 04 UID SEARCH UID 22000:25000 UNDELETED
S: * SEARCH 25000 24998 (... 997 UIDs ...) 23221
S: 04 OK [MESSAGELIMIT 1000 23221] SEARCH completed with 1000 partial
results

The following example demonstrates copy of messages with UIDs
between 18000:21000. The total message count exceeds the
server’s published 1000 messages limit.

C: 05 UID COPY 18000:21000 "Trash"
S: * NO [MESSAGELIMIT 1000 20001] Too many messages to copy
S: 05 OK [COPYUID 1397597919 20001:21000 21363:22362] COPY completed f
or the last 1000 messages

Open Issue: Note that the above example shows a UID COPY that
partially fails. This is assumed to be better for clients that don't understand the MESSAGELIMIT response code. However this might cause naive clients to believe that the COPY fully completed and that all messages were copied. (An alternative would be to return MESSAGELIMIT in the tagged NO response, meaning that no messages could be copied. However this wouldn’t work well with clients that don’t support MESSAGELIMIT response code.)

The following example shows MOVE of messages with UIDs between 18000:21000. The total message count exceeds the server’s published 1000 messages limit. The client that wants to move all messages in the range and observes a MESSAGELIMIT response code, can repeat the command by updating the UID set parameter specified in the command. The client needs to keep doing this until MESSAGELIMIT response is not returned (or until a tagged NO/BAD is returned).

C: 06 UID MOVE 18000:21000 "Archive/2021/2021-12"
S: * OK [COPYUID 1397597919 20001:21000 22363:23362] Some messages were not moved
  S: * 12336 EXPUNGE
  S: * 12335 EXPUNGE
  ...
  S: * 11335 EXPUNGE
  S: 06 OK [MESSAGELIMIT 1000 20001] MOVE completed for the last 1000 messages

The following example shows update of flags for messages with UIDs between 18000:20000. The total message count exceeds the server’s published 1000 messages limit. The client that wants to change flags for all messages in the range and observes a MESSAGELIMIT response code, can repeat the command by updating the UID set parameter specified in the command. The client needs to keep doing this until MESSAGELIMIT response is not returned (or until a tagged NO/BAD is returned).

C: 07 UID STORE 18000:20000 +FLAGS (\Seen)
S: * 11215 FETCH (FLAGS (\Seen \Deleted) UID 20000)
S: * 11214 FETCH (FLAGS (\Seen \Answered \Deleted) UID 19998)
  ...
  S: * 10216 FETCH (FLAGS (\Seen) UID 19578)
  S: 07 OK [MESSAGELIMIT 1000 19578] STORE completed for the last 1000 messages

The following example shows use of MESSAGELIMIT response code together with the PARTIAL extension. The total message count exceeds the server’s published 1000 messages limit.
Note that when the server needs to return both EXPUNGEISSUED ([RFC9051]) and MESSAGELIMIT response codes, the former MUST be returned in the tagged OK response, while the latter MUST be returned in an untagged NO response. The following example demonstrates that:

C: 08 UID FETCH 22000:25000 (UID FLAGS MODSEQ) (PARTIAL -1:-1500)
S: 08 NO [MESSAGELIMIT 1000] FETCH exceeds the maximum 1000 message limit

C: 031 FETCH 10000:14589 (UID FLAGS)
S: * 14589 FETCH (FLAGS (\Seen) UID 25000)
S: * 14588 FETCH (FLAGS (\Answered) UID 24998)
S: ... further 997 fetch responses
S: * 13590 FETCH (FLAGS () UID 23221)
S: * NO [MESSAGELIMIT 1000 23221] FETCH completed with 1000 partial results
S: 031 OK [EXPUNGEISSUED] Some messages were also expunged

4.2. Interaction with SORT and THREAD extensions

Servers that advertise MESSAGELIMIT N will be unable to execute a THREAD command in a mailbox with more than N messages.

Servers that advertise MESSAGELIMIT N might be unable to execute a SORT command in a mailbox with more than N messages, unless they maintain indices for different SORT orders they support.

5. 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.

```
SP                  = <Defined in RFC 5234>
MINUS               = "-"
capability          =/ "PARTIAL"
                    ;; <capability> from [RFC3501]
modifier-partial    = "PARTIAL" SP partial-range
```
partial-range-first = nz-number ":" nz-number
    ;; Request to search from oldest (lowest UIDs) to
    ;; more recent messages.
    ;; A range 500:400 is the same as 400:500.
    ;; This is similar to <seq-range> from [RFC3501],
    ;; but cannot contain "*".

partial-range-last = MINUS nz-number ":" MINUS nz-number
    ;; Request to search from newest (highest UIDs) to
    ;; oldest messages.
    ;; A range -500:-400 is the same as -400:-500.

partial-range = partial-range-first / partial-range-last

search-return-opt =/ modifier-partial
    ;; All conform to <search-return-opt>, from [IMAP-ABNF]/[RFC9051]

search-return-data =/ ret-data-partial

ret-data-partial = "PARTIAL"
    ;; <partial-range> is the requested range.
    SP "(" partial-range SP partial-results ")"

partial-results = sequence-set / "NIL"
    ;; <sequence-set> from [RFC3501].
    ;; NIL indicates no results correspond to the requested range.

tagged-ext-simple =/ partial-range-last

fetch-modifier =/ modifier-partial

capability =/ "MESSAGELIMIT=" message-limit
    ;; <capability> from [RFC3501]

message-limit = nz-number

resp-text-code =/ "MESSAGELIMIT" SP message-limit [SP uniqueid]
    ;; No more than nz-number messages can be processed
    ;; by any command at a time. The last (lowest) processed
    ;; UID is uniqueid.
    ;; The last parameter is omitted, when not known.

6. Security Considerations

    TBD.
7. IANA Considerations

7.1. Changes/additions to the 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

IANA is requested to add definition of the PARTIAL extension to point to this document.

8. Acknowledgments

This document was motivated by Yahoo! team and their questions about best client practices for dealing with large mailboxes.

Editor of this document would like to thank the following people who provided useful comments or participated in discussions of this document: Timo Sirainen.

This document uses lots of text from RFC 5267. Thus work of the RFC 5267 authors Dave Cridland and Curtis King is appreciated.

9. References

9.1. Normative References


9.2. Informative References


Index

M

MESSAGELIMIT (response code)
Section 4.1, Paragraph 3.2.1

Authors’ Addresses

Alexey Melnikov
Isode Limited
Email: alexey.melnikov@isode.com
URI: https://www.isode.com

Arun Prakash Achuthan
Yahoo!
Email: arunprakash@myyahoo.com

Abstract

This document defines a QUOTA extension of the Internet Message Access Protocol (RFC 3501/RFC 9051) that permits administrative limits on resource usage (quotas) to be manipulated through the IMAP protocol.

This document 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 22 May 2022.

Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.
Melnikov                   Expires 22 May 2022                  [Page 2]
8. Security Considerations ........................................ 16
9. IANA Considerations ........................................... 17
   9.1. Changes/additions to the IMAP4 capabilities registry ... 17
   9.2. IMAP quota registry ....................................... 17
   9.3. Registrations of IMAP Quota Resource Types .......... 18
10. Contributors .................................................. 19
11. Acknowledgments ............................................... 19
12. Changes since RFC 2087 ......................................... 20
13. References ..................................................... 20
   13.1. Normative References .................................... 20
   13.2. Informative References .................................. 21
Author’s Address .................................................. 21

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 before such lines unless specifically mentioned.

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.

Other capitalised words are IMAP keywords [RFC3501][RFC9051] or
keywords from this document.

2. Introduction and Overview

This document defines a couple of extensions to the Internet Message
Access Protocol [RFC3501] for querying and manipulating
administrative limits on resource usage (quotas). This extension is
compatible with both IMAP4rev1 [RFC3501] and IMAP4rev2 [RFC9051].

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 12 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, informational 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-dependent, 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 (see Section 3.2), 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.

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 63 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 response/response code consisting 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

This document introduces a concept of a "quota root", as resource limits can apply across multiple IMAP mailboxes.

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 document 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 arbitrary mailboxes or mailbox hierarchies.

Not all resources may be limitable or calculable 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-dependent 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. (Names of quota roots applicable to a particular mailbox can be discovered by issuing the GETQUOTAROOT command, see Section 4.1.2.) Note that the server is not required to support any specific resource type (as advertised in the CAPABILITY response, i.e. all capability items with the "QUOTA=RES-" prefix) 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, even resource limits not explicitly listed in the SETQUOTA command. (For example, if the quota root had both STORAGE and MESSAGE limits assigned to the quota root before the SETQUOTA is called and the SETQUOTA only includes the STORAGE limit, then the MESSAGE limit is removed from the quota root.)

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.

The DELETED status data item requests the server to return the number of messages with \Deleted flag set. The DELETED status data item is only required to be implemented when the server advertises QUOTA=RES-MESSAGE capability.

The 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. The 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, a GETQUOTAROOT or a SETQUOTA 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.

Examples:

S: * QUOTAROOT INBOX ""

// The INBOX mailbox is covered by a single quota root with name "".

S: * QUOTAROOT comp.mail.mime

// The comp.mail.mime mailbox has no quota root associated with it, but one can be created.

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 1:  C: A003 APPEND saved-messages ({\Seen} {326})
             S: + Ready for literal data
             C: Date: Mon, 7 Feb 1994 21:52:25 -0800 (PST)
             C: From: Fred Foobar <foobar@Blurdybloop.example>
             C: Subject: afternoon meeting
             C: To: mooch@owatagu.siam.edu.example
             C: Message-Id: <B27397-01000000@Blurdybloop.example>
             C: MIME-Version: 1.0
             C: Content-Type: TEXT/PLAIN; CHARSET=US-ASCII
             C:
             C: Hello Joe, do you think we can meet at 3:30 tomorrow?
             C:
             S: A003 NO [OVERQUOTA] APPEND Failed

The OVERQUOTA response code MAY also be returned in an untagged NO response in the authenticated or the selected state, when a mailbox exceeds soft quota. For example, such OVERQUOTA response code might be sent as a result of an external event (e.g. LMTP delivery or COPY/MOVE/APPEND in another IMAP connection) that causes the currently selected mailbox to exceed soft quota. Note that such OVERQUOTA response code might be ambiguous, because it might relate to the target mailbox (as specified in COPY/MOVE/APPEND) or to the currently selected mailbox. (The WG chose not to address this deficiency due to syntactic limitations of IMAP response codes and because such events are likely to be rare.) This form of the OVERQUOTA response codes MUST NOT be returned if there is no mailbox selected and no command in progress that adds a message to a mailbox (e.g. APPEND).
Example 2:  
C: A003 APPEND saved-messages (\Seen) {326}  
S: + Ready for literal data  
C: Date: Mon, 7 Feb 1994 21:52:25 -0800 (PST)  
C: From: Fred Foobar <foobar@Blurdybloop.example>  
C: Subject: afternoon meeting  
C: To: mooch@owatagu.siam.edu.example  
C: Message-Id: <B27397-0100000@Blurdybloop.example>  
C: MIME-Version: 1.0  
C: Content-Type: TEXT/PLAIN; CHARSET=US-ASCII  
C:  
C: Hello Joe, do you think we can meet at 3:30 tomorrow?  
C:  
S: * NO [OVERQUOTA] Soft quota has been exceeded  
S: A003 OK [APPENDUID 38505 3955] APPEND completed  

Example 3:  
C: A004 COPY 2:4 MEETING  
S: * NO [OVERQUOTA] Soft quota has been exceeded  
S: A004 OK [COPYUID 38505 304,319:320 3956:3958] COPY command 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 multiple 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 the DELETED-STOREAGE status data item.

Support for this resource MUST be indicated by the server by advertising the CAPABILITY "QUOTA=RES-STOREAGE".

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 the 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-STOREAGE

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-STOREAGE".
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.

<table>
<thead>
<tr>
<th>Operations</th>
<th>Rights</th>
<th>l</th>
<th>r</th>
<th>s</th>
<th>w</th>
<th>i</th>
<th>c</th>
<th>x</th>
<th>t</th>
<th>e</th>
<th>a</th>
<th>Any</th>
<th>Non</th>
</tr>
</thead>
<tbody>
<tr>
<td>GETQUOTA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>+</td>
</tr>
<tr>
<td>GETQUOTAROOT</td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*</td>
</tr>
<tr>
<td>SETQUOTA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>+</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1

See Section 4 of RFC 4314 for conventions used in this table.

Legend:

+ - The right is required

* - Only one of the rights marked with * is required

"Any" - at least one of the "l", "r", "i", "c", "x", "a" rights is required

"Non" - no rights required to perform the command

Note that which permissions are needed in order to perform GETQUOTAROOT command depends on the quota resource type being requested. For example, a quota on number of messages (MESSAGE resource type) or total size of messages (STORAGE resource type) requires "r" right on the mailbox in question, since the quota involved would reveal information about the number (or total size) of messages in the mailbox. By comparison, the MAILBOX resource type doesn’t require any right.

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-ext
resource-name-ext = atom
;; Future resource registrations
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"

number64 = <Defined in RFC 9051>

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. In particular, no quota information should be disclosed to anonymous users.
As for any resource shared across users (for example a quota root attached to a set of shared mailboxes), a user that can consume or render unusable the resource can affect the resources available to the other users; this might occur, for example, by a user with permission to execute SETQUOTA setting an artificially small value.

Note that computing resource usage might incur a heavy load on the server. Server implementers should consider implementation techniques that lower load on servers, such as caching of resource usage information or usage of less precise computations when under heavy load.

9. IANA Considerations

9.1. Changes/additions to the 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

IANA is requested to update definition of the QUOTA extension to point to this document. IANA is also requested to add the "QUOTASET" capability to the IMAP4 capabilities registry, with this document as the reference.

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 (see Section 9.2).

IANA is requested to reserve all other capabilities starting with "QUOTA=" prefix for future IETF Stream extensions to this document.

9.2. IMAP quota resource type registry

IANA is 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.

Designated Experts should check that provided references are correct, that they describe the quota resource type being registered in sufficient details to be implementable, that syntax of any optional
commands/responses is correct (e.g. ABNF validates), and their syntax/description complies with rules and limitations imposed by IMAP [RFC3501][RFC9051]. Designated Experts should avoid registering multiple identical quota resource types under different names and should provide advice to requestors about other possible quota resource types to use.

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 Section 9.3 for the registration templates.

9.3. 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: ANNOTATION-STORAGE

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.

Extra required IMAP commands/responses: N/A

Extra optional IMAP commands/responses: N/A

Reference: Section 5.4 of RFCXXXX

10. Contributors

Dave Cridland wrote lots of text in an earlier draft that became the basis for this document.

11. Acknowledgments

Editor 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, Benjamin Kaduk, Roman Danyliw, Eric Vyncke.

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.
12. Changes since RFC 2087

This document is a revision of RFC 2087. It tries to clarify the 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 4 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"). The DELETED STATUS data item must be implemented if the QUOTA=RES-MESSAGE capability is advertised. The DELETED-STORAGE STATUS data item must be implemented if the QUOTA=RES-STORAGE capability is advertised. For extensibility quota usage and quota limits are now 63 bit unsigned integers.

13. References

13.1. Normative References


13.2. Informative References


Author’s Address

Alexey Melnikov
Isode Limited

Email: alexey.melnikov@isode.com
URI: https://www.isode.com
IANA registry for Sieve actions
draft-ietf-extra-sieve-action-registry-02

Abstract

This document creates a registry of Sieve (RFC 5228) actions in order to help developers and Sieve extension writers track interactions between different extensions.

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 8 September 2022.

Copyright Notice

Copyright (c) 2022 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 Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.
1. Introduction

Sieve Email Filtering Language [RFC5228] is a popular email filtering language used upon final mail delivery. Popularity of Sieve resulted in a myriad of Sieve extensions that can interact with each other in wonderful and complex ways. There is currently no easy way to find out all actions defined by Sieve extensions published in RFCs, which make it quite difficult for Sieve extension writers and Sieve implementation developers to foresee interactions between Sieve actions.

This document creates a registry of Sieve [RFC5228] actions in order to help developers and Sieve extension writers track interactions between different extensions.

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. IANA Considerations

3.1. Sieve Actions Registration Template and Procedure

IANA is requested to create a new registry for Sieve actions (see Section 2.9 of [RFC5228] for details on Sieve actions). Registration of both actions specified in IETF Stream RFCs and vendor specific actions is allowed and encouraged. The registration template contains:

1. name of the action;
2. short description;

3. references: one or more documents describing the action and any significant updates to its definition (this field is REQUIRED for actions described in RFCs and OPTIONAL otherwise);

4. name(s) of Sieve capability(ies) associated with the Sieve action being registered;

5. interactions with other Sieve actions, if any;

6. flag specifying whether the action cancels the implicit keep (see Section 2.10.2 of [RFC5228]);

7. whether or not this action can be used with IMAP events in Sieve ([RFC6785]), and

8. optional comment.

Registration procedure for this registry is Expert Review. The Designated Expert only checks that the name of the action being registered matches documentation, that the description field is accurate, that the correct documents are referenced and that the list of relevant documents is as complete as possible. The Designated Expert can’t reject a registration based on personal dislike of the document defining an action and should always err on the side of registering, even if documentation is not complete.

Addition of a new reference to an existing registration or change to the description field goes through the same registration procedure as a new registration.

3.2. Initial Sieve Action Registry

The following table is used to initialize the actions registry.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>References</th>
<th>Capabilities</th>
<th>Interactions</th>
<th>Cancels</th>
<th>Use</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>addheader</td>
<td>Add a header</td>
<td>[RFC5293]</td>
<td>&quot;editheader&quot;</td>
<td>All</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>addflag</td>
<td>Add flags to a list of IMAP flags</td>
<td>[RFC5232], [RFC5229], [RFC5229]</td>
<td>&quot;imap4flags&quot;, &quot;variables&quot;</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>----------------------------------</td>
<td>---------------------------------</td>
<td>--------------------------</td>
<td>---</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>convert</th>
<th>Convert body parts from one MIME type to another</th>
<th>[RFC6558]</th>
<th>&quot;convert&quot;</th>
<th>All</th>
<th>N</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>deleteheader</th>
<th>Remove a header field from the existing message</th>
<th>[RFC5293]</th>
<th>&quot;editheader&quot;</th>
<th>All</th>
<th>N</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>discard</th>
<th>Silently throw away the message</th>
<th>[RFC5228]</th>
<th></th>
<th>Compatible</th>
<th>Y</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>enclose</th>
<th>Enclose a message as an</th>
<th>[RFC5703]</th>
<th>&quot;enclose&quot;</th>
<th>All</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>tests and actions, except &quot;redirect&quot; apply to the altered message</td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------------------</td>
<td>----------------------</td>
<td>----------------------</td>
<td>---------------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>attachment to a new message</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| ereject               | Refuse               | [RFC5429] "ereject" | Y       | N         |
|                      | delivery of          |                      |         |           |
|                      | the message          |                      |         |           |

| extracttext          | Store text of [RFC5703], "extracttext", [RFC5229] "variables" | N       |         |
|                      | into a variable      |                      |         |           |

<p>| fileinto             | Deliver the [RFC5228], &quot;fileinto&quot;, Use of :copy | Y       |         |
|                      | message into [RFC3894], &quot;copy&quot;, suppresses      |         |         |</p>
<table>
<thead>
<tr>
<th>keep</th>
<th>File message</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>notify</td>
<td>Send a</td>
<td>N</td>
</tr>
<tr>
<td>redirect</td>
<td>Send</td>
<td>Y</td>
</tr>
<tr>
<td>reject</td>
<td>Refuse</td>
<td>Y</td>
</tr>
<tr>
<td>removeflag</td>
<td>Remove flags</td>
<td>N</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>keep</th>
<th>File message</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>notify</td>
<td>Send a</td>
<td>N</td>
</tr>
<tr>
<td>redirect</td>
<td>Send</td>
<td>Y</td>
</tr>
<tr>
<td>reject</td>
<td>Refuse</td>
<td>Y</td>
</tr>
<tr>
<td>removeflag</td>
<td>Remove flags</td>
<td>N</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>keep</th>
<th>File message</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>notify</td>
<td>Send a</td>
<td>N</td>
</tr>
<tr>
<td>redirect</td>
<td>Send</td>
<td>Y</td>
</tr>
<tr>
<td>reject</td>
<td>Refuse</td>
<td>Y</td>
</tr>
<tr>
<td>removeflag</td>
<td>Remove flags</td>
<td>N</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>keep</th>
<th>File message</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>notify</td>
<td>Send a</td>
<td>N</td>
</tr>
<tr>
<td>redirect</td>
<td>Send</td>
<td>Y</td>
</tr>
<tr>
<td>reject</td>
<td>Refuse</td>
<td>Y</td>
</tr>
<tr>
<td>removeflag</td>
<td>Remove flags</td>
<td>N</td>
</tr>
<tr>
<td>replace</td>
<td>Replace a MIME part</td>
<td>[RFC5703]</td>
</tr>
<tr>
<td>---------</td>
<td>---------------------</td>
<td>----------</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>set</th>
<th>Store a value in a variable</th>
<th>[RFC5229]</th>
<th>&quot;variables&quot;</th>
<th>N</th>
</tr>
</thead>
</table>

---

<table>
<thead>
<tr>
<th>setflag</th>
<th>Set IMAP system flags or keywords</th>
<th>[RFC5232], &quot;imap4flags&quot;, [RFC5229]</th>
<th>&quot;variables&quot;</th>
<th>N</th>
</tr>
</thead>
</table>

---

<table>
<thead>
<tr>
<th>vacation</th>
<th>Vacation autoresponder</th>
<th>[RFC5230], &quot;vacation&quot;, [RFC6131], &quot;vacation-&quot;</th>
<th>N</th>
<th>N</th>
</tr>
</thead>
</table>
4. Security Considerations

The sole purpose of this document is to create a new IANA registry, so it doesn’t create new security considerations for Sieve implementations.

The new registry should help Sieve extension writers and Sieve implementors track interactions between different Sieve actions, so it might improve quality of specifications and implementations, including security aspects.

5. References

5.1. Normative References


5.2. Informative References


Appendix A. Acknowledgements

TBD.

Authors’ Addresses

Alexey Melnikov
Isode Ltd
14 Castle Mews
Hampton
TW12 2NP
United Kingdom
Email: Alexey.Melnikov@isode.com

Kenneth Murchison
Fastmail US LLC
1429 Walnut Street - Suite 1201
Philadelphia, PA 19102
United States of America
Email: murch@fastmailteam.com
Sieve Email Filtering: Snooze Extension
draft-ietf-extra-sieve-snooze-04

Abstract

This document describes the "snooze" extension to the Sieve email filtering language. The "snooze" extension gives Sieve the ability to postpone the delivery of an incoming email message into a target mailbox until a later point in time.

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 8 September 2022.

Copyright Notice

Copyright (c) 2022 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 Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.
1. Introduction

Users are not always ready, willing, or able to read and respond to email messages at the time of their arrival. Sometimes it is desirable to have messages appear in a mailbox at a more convenient time for the user to act upon them.

This document defines an extension to the Sieve language [RFC5228] that enables scripts to postpone the delivery of a message into a target mailbox until a later point in time.

2. Conventions Used in This Document

Conventions for notations are as in Section 1.1 of [RFC5228], including use of the "Usage:" label for the definition of action and tagged arguments syntax.
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. Capability Identifier

Sieve implementations that implement this extension have an identifier of "snooze" for use with the capability mechanism.

4. Snooze Action

Usage: snooze *AWAKEN-OPTIONS <times: string-list>

The AWAKEN-OPTIONS argument is defined here in ABNF [RFC4234] syntax so that it can be modified by other extensions.

AWAKEN-OPTIONS = MAILBOX / WEEKDAYS / TZID
; each option MUST NOT appear more than once
; however, per Section 2.6.2 of RFC 5228,
; the tagged arguments in AWAKEN-OPTIONS
; may appear in any order

MAILBOX = ":mailbox" string
WEEKDAYS = ":weekdays" string-list
TZID = ":tzid" string

The "snooze" action cancels the implicit keep and postpones delivery of the message into the specified mailbox at a later point in time.

The snooze action is semantically equivalent to a delayed fileinto action (see Section 4.1 of [RFC5228]). The arguments of the snooze action specify when, where, and how the awakened message will be filed.

A Sieve interpreter MUST implement the snooze action by delivering the message to a special "snoozed" mailbox within its mailstore. IMAP [RFC3501] and JMAP [RFC8621] servers MUST apply the "Snoozed" (Section 8.3) attribute to this mailbox. The message will reside in this special mailbox until the prescribed awaken time at which it will be moved into the specified target mailbox.
4.1. Mailbox Argument

The optional :mailbox argument is used to specify the target mailbox that the message will be filed into when it is awakened. It is equivalent to the mailbox argument of the fileinto action (see Section 4.1 of [RFC5228]).

If :mailbox is omitted, or if the specified mailbox doesn’t exist at the time of awakening, the message will be filed into the user’s main mailbox. For instance, in an implementation where an IMAP server is running scripts on behalf of the user at time of delivery, the user’s "INBOX" would be the implicit target for awakening messages.

4.2. Weekdays Argument

The optional :weekdays argument specifies the set of days on which the specified set of awakening times apply. Each day of the week is expressed as an integer between "0" and "6". "0" is Sunday, "1" is Monday, etc. This syntax matches that of the "weekday" date-part argument to the date test extension (see Section 4.2 of [RFC5260]).

If :weekdays is omitted, the set of awakening times applies to every day of the week.

4.3. Times and TZID Arguments

The required times argument, along with the optional :tzid argument, are used to specify when a snoozed message will be awakened. Each time is specified in "hh:mm:ss" format and is interpreted as the local time in the time zone specified by the :tzid argument.

The value of the :tzid argument MUST be a time zone identifier from the IANA Time Zone Database [tzdb]. If :tzid is omitted, the time zone of the Sieve interpreter is used.

The combination of the weekdays and times form a chronological list of awaken times. When a message is snoozed, it is assigned the next future awaken time in the list. If a message is snoozed on a day with no awaken times, or after the last awaken time on a given day, the first awaken time on the next available day is used.

If the local time in the specified time zone occurs more than once (daylight saving to standard time transition), the first occurrence of the specified time value is used. If the local time in the specified time zone does not occur (standard to daylight saving time transition), the specified time value is interpreted using the UTC offset prior to the transition.
4.3.1. Awaken Times Examples

The following examples show, given the specified snooze action and a set of message arrival times, the corresponding times at which the message would be awakened and filed.

The following example shows awaken times rolling into the next day or week. Note that 2020-07-30 falls on a Thursday.

```
require "snooze";
require "snooze";
snooze :weekdays ["1", "3", "5", "2", "4"]
  :tzid "Australia/Melbourne" ["12:00:00", "08:00:00", "16:00:00"];
```

<table>
<thead>
<tr>
<th>Arrival (UTC)</th>
<th>Arrival (Melbourne)</th>
<th>Awaken (Melbourne)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2020-07-30T00:00:00Z</td>
<td>--07-30T10:00:00+10</td>
<td>--07-30T12:00:00+10</td>
</tr>
<tr>
<td>2020-07-30T04:00:00Z</td>
<td>--07-30T14:00:00+10</td>
<td>--07-30T16:00:00+10</td>
</tr>
<tr>
<td>2020-07-30T08:00:00Z</td>
<td>--07-30T18:00:00+10</td>
<td>--07-31T08:00:00+10</td>
</tr>
<tr>
<td>2020-07-31T12:00:00Z</td>
<td>--07-31T22:00:00+10</td>
<td>--08-03T08:00:00+10</td>
</tr>
<tr>
<td>2020-08-01T16:00:00Z</td>
<td>--08-02T02:00:00+10</td>
<td>--08-03T08:00:00+10</td>
</tr>
</tbody>
</table>

Table 1

The following example shows awaken times falling before, during, and after a daylight saving to standard time transition. Note that the transition occurs at 2020-11-01T02:00:00-04.

```
require "snooze";
snooze :tzid "America/New_York" "01:30:00";
```

<table>
<thead>
<tr>
<th>Arrival (UTC)</th>
<th>Arrival (New York)</th>
<th>Awaken (New York)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2020-11-01T05:00:00Z</td>
<td>--11-01T01:00:00-04</td>
<td>--11-01T01:30:00-04</td>
</tr>
<tr>
<td>2020-11-01T06:00:00Z</td>
<td>--11-01T01:00:00-05</td>
<td>--11-02T01:30:00-05</td>
</tr>
<tr>
<td>2020-11-01T07:00:00Z</td>
<td>--11-01T02:00:00-05</td>
<td>--11-02T01:30:00-05</td>
</tr>
</tbody>
</table>

Table 2
The following example shows awaken times falling before, during, and after a standard to daylight saving time transition. Note that the transition occurs at 2021-03-14T02:00:00-05.

```
require "snooze";
snooze :tzid "America/New_York" "02:30:00";
```

```
+---------------------------------------------+-----------------+-----------------+
<table>
<thead>
<tr>
<th>Arrival (UTC)</th>
<th>Arrival (New York)</th>
<th>Awaken (New York)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2021-03-13T06:30:00Z</td>
<td>--03-13T01:30:00-05</td>
<td>--03-13T02:30:00-05</td>
</tr>
<tr>
<td>2021-03-14T06:30:00Z</td>
<td>--03-14T01:30:00-05</td>
<td>--03-14T03:30:00-04</td>
</tr>
<tr>
<td>2021-03-14T07:30:00Z</td>
<td>--03-14T03:30:00-04</td>
<td>--03-15T02:30:00-04</td>
</tr>
</tbody>
</table>
```

Table 3

4.4. Interaction with Extensions to the Fileinto Action

Some tagged arguments defined in extensions to the fileinto action can be used together with the snooze action. The sections below describe these interactions. Tagged arguments in future extensions to the fileinto action need to describe their interaction with the snooze extension, if any.

When any fileinto extension arguments are used with the snooze extension, the corresponding extension MUST be enabled, and the arguments are defined to have the same syntax, semantics, and treatment as they do with the fileinto action.

4.4.1. Imap4flags Extension

When the "imap4flags" [RFC5232] extension is enabled in a script, two additional tagged arguments are added to "snooze" that allow manipulating the set of flags on a snoozed message.

```
AWAKEN-OPTIONS /= ADDFLAGS / REMOVEFLAGS
```

```
ADDFLAGS = ":addflags" string-list
REMOVEFLAGS = ":removeflags" string-list
```

The optional :addflags and :removeflags arguments are used to specify which IMAP [RFC3501] flags should be added to and/or removed from the set of IMAP flags present on the snoozed message at the time of awakening. Note the set of IMAP flags present at the time of awakening may be the empty set.
If the "setflag" and/or "addflag" actions have been used to store IMAP flags in the imap4flags internal variable, the Sieve interpreter MUST use the current value of the internal variable as the set of flags to associate with the message when storing it into the "snoozed" mailbox.

This document doesn’t dictate how the Sieve interpreter will set the IMAP flags. In particular, the Sieve interpreter may work as an IMAP client or may have direct access to the mailstore.

The general requirements for flag handling specified in Section 2 of [RFC5232] MUST be followed.

4.4.1.1. Example

The following example leverages the Date [RFC5260], Relational [RFC5231], and Imap4flags [RFC5232] extensions to snooze messages received after business hours until the following work day. Note that the message is marked as important when it is snoozed, and will be marked as unread when it is awakened.

```sieve
require ["snooze", "imap4flags", "date", "relational"];

if anyof(header :is "from" "boss@example.com",
         currentdate :is "weekday" "0",
         currentdate :is "weekday" "6",
         currentdate :value "ge" "hour" "17") {
  setflag "\Important";
  snooze :removeflags "\Seen"
         :weekdays ["1", "2", "3", "4", "5"]
         :tzid "American/New_York", "09:00";
}
```

4.4.2. Mailbox Extension

This document extends the definition of the ":create" [RFC5490] tagged argument so that it can be used with the snooze action.

AWAKEN-OPTIONS /= CREATE

CREATE = ":create"
  ; MUST NOT be appear unless MAILBOX also appears

If the optional ":create" argument is specified with snooze, it instructs the Sieve interpreter to create the target mailbox, if needed, before attempting to file the awakened message into the target mailbox.
4.4.3. Special-Use Extension

This document extends the definition of the "specialuse" [RFC8579] tagged argument so that it can be used with the snooze action.

AWAKEN-OPTIONS /= SPECIAL-USE

SPECIAL-USE = "specialuse" string

If the optional "specialuse" argument is specified with snooze, it instructs the Sieve interpreter to check whether a mailbox exists with the specific special-use flag assigned to it. If such a mailbox exists, the awakened message is filed into the special-use mailbox. Otherwise, the awakened message is filed into the target mailbox.

If both the optional "specialuse" and "create" arguments are specified with snooze, the Sieve interpreter is instructed to create the target mailbox per Section 4.1 of [RFC8579], if needed.

4.4.4. MailboxID Extension

This document extends the definition of the "mailboxid" [RFC9042] tagged argument so that it can be used with the snooze action.

AWAKEN-OPTIONS /= MAILBOXID

MAILBOXID = "mailboxid" string

If the optional "mailboxid" argument is specified with snooze, it instructs the Sieve interpreter to check whether a mailbox exists in the user’s personal namespace [RFC2342] with the specified MAILBOXID [RFC8474]. If such a mailbox exists, the awakened message is filed into that mailbox. Otherwise, the awakened message is filed into the target mailbox.

It is an error to specify both "mailboxid" and "specialuse" in the same snooze action.

5. Implementation Status

< RFC Editor: before publication please remove this section and the reference to [RFC7942] >

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".

5.1. Cyrus Server

The open source Cyrus Server (http://www.cyrusimap.org/) project is a highly scalable enterprise mail system which supports Sieve email filtering at the point of final delivery. This production level Sieve implementation supports all of the requirements described in this document. This implementation is freely distributable under a BSD style license from Computing Services at Carnegie Mellon University (http://www.cmu.edu/computing/).

6. Security Considerations

Security considerations are discussed in [RFC5228], [RFC5232], [RFC8579], and [RFC9042].

It is believed that this extension doesn’t introduce any additional security concerns.

7. Privacy Considerations

It is believed that this extension doesn’t introduce any privacy considerations beyond those in [RFC5228].

8. IANA Considerations

8.1. Registration of Sieve Extension

This document defines the following new Sieve extension to be added to the registry defined in Section 6.2 of [RFC5228] and located here: https://www.iana.org/assignments/sieve-extensions/sieve-extensions.xhtml#sieve-extensions
IANA are requested to add a capability to the Sieve Extensions registry:

To: iana@iana.org
Subject: Registration of new Sieve extension
Capability name: snooze
Description: Adds the "snooze" action command to postpone delivery of a message into a target mailbox until a later point in time.
RFC number: RFC XXXX
Contact address: The Sieve discussion list <sieve@ietf.org>

8.2. Registration of Sieve Action

This document defines the following new Sieve action to be added to the registry defined in Section 3.1 of [I-D.ietf-extra-sieve-action-registry].

IANA are requested to add an action to the Sieve Action registry:

Name: snooze
Description: Postpone delivery of a message into a target mailbox until a later point in time.
References: RFC XXXX, [RFC5232], [RFC5490], [RFC8579], [RFC9042]
Capabilities: "snooze", "imap4flags", "mailbox", "special-use", "mailboxid".
Interactions: Is not compatible with the reject or ereject actions.
Cancels Implicit Keep?: Y
Use with IMAP Events?: Y
Comments: Requires a special "snoozed" mailbox in the mailstore.
8.3. Registration of IMAP Mailbox Name Attribute

This document defines the following new IMAP mailbox name attribute to be added to the registry defined in Section 6.2 of [RFC8457] and located here: https://www.iana.org/assignments/imap-mailbox-name-attributes/imap-mailbox-name-attributes.xhtml#imap-mailbox-name-attributes

IANA are requested to add an attribute to the IMAP Mailbox Name Attribute registry:

To: iana@iana.org

Subject: Registration of new IMAP Mailbox Name Attribute

Attribute name: Snoozed

Description: Messages that have been snoozed.

Reference: RFC XXXX

9. Acknowledgments

The authors would like to thank the following individuals for contributing their ideas and support for writing this specification: Ned Freed, Barry Leiba, and Alexey Melnikov.

10. References
10.1. Normative References

[I-D.ietf-extra-sieve-action-registry]


10.2. Informative References


Appendix A. Change History (To be removed by RFC Editor before publication)

Changes since draft-ietf-extra-sieve-snooze-03:
* Added "snooze" to the Sieve Actions Registry.

Changes since draft-ietf-extra-sieve-snooze-02:
* Updated :mailboxid reference to RFC9042.
* Added an informative reference to RFC8621.
* Miscellaneous editorial changes.

Changes since draft-ietf-extra-sieve-snooze-01:
* Miscellaneous editorial changes.

Changes since draft-ietf-extra-sieve-snooze-00:
* Disallow both :mailboxid and :specialuse in the same snooze action.
* Updated :mailboxid reference to draft-ietf-extra-sieve-mailboxid
* Specified that snooze cancels implicit keep.
* Specified that implementations MUST use a "snoozed" mailbox.
* Added registration of \Snoozed Special-Use Attribute.
* Added example of manipulating IMAP flags at both snooze time and awaken time.

* Miscellaneous editorial changes.

Authors’ Addresses

Kenneth Murchison  
Fastmail US LLC  
1429 Walnut Street - Suite 1201  
Philadelphia, PA 19102  
United States of America  
Email: murch@fastmailteam.com

Ricardo Signes  
Fastmail US LLC  
1429 Walnut Street - Suite 1201  
Philadelphia, PA 19102  
United States of America  
Email: rjbs@fastmailteam.com

Neil Jenkins  
Fastmail Pty Ltd  
Level 2, 114 William Street  
Melbourne VIC 3000  
Australia  
Email: neilj@fastmailteam.com
Abstract

The JMAP base protocol (RFC8620) provides the ability to upload and download arbitrary binary data via HTTP POST and GET on defined endpoint. This binary data is called a "blob".

This extension adds additional ways to create and access blobs, by making inline method calls within a standard JMAP request.

This extension also adds a reverse lookup mechanism to discover where blobs are referenced within other data types.

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 4 December 2022.

Copyright Notice

Copyright (c) 2022 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
1. Introduction

Sometimes JMAP ([RFC8620]) interactions require creating a blob and then referencing it. In the same way that IMAP Literals were extended by [RFC7888], embedding small blobs directly into the JMAP method calls array can be an option for reducing roundtrips.

Likewise, when fetching an object, it can be useful to also fetch the raw content of that object without a separate roundtrip.

Since raw blobs may contain arbitrary binary data, this document defines a use of the base64 coding specified in [RFC4648] for both creating and fetching blob data.
Where JMAP is being proxied through a system which applies additional access restrictions, it can be useful to know which objects reference any particular blob, and this document defines a way to discover those references.

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.

The definitions of JSON keys and datatypes in the document follow the conventions described in the core JMAP specification [RFC8620].

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 an additional capability URI.

3.1. urn:ietf:params:jmap:blob

The capability urn:ietf:params:jmap:blob being present in the "accountCapabilities" property of an account represents support for additional API methods on the Blob datatype. Servers that include the capability in one or more "accountCapabilities" properties MUST also include the property in the "capabilities" property.

The value of this property in the JMAP session "capabilities" property MUST be an empty object.

The value of this property in an account’s "accountCapabilities" property is an object that MAY contain the following information on server capabilities and permissions for that account:

* maxSizeBlobSet: UnsignedInt|null

If present, this is the maximum size of blob (in octets) that the server will allow to be created (including blobs created by concatenating multiple data sources together).

Clients MUST NOT attempt to create blobs larger than this size.
If this value is not present or null, then clients are not required to limit the size of blob they try to create, though servers can always reject creation of blobs regardless of size; e.g. due to lack of disk space, or per-user rate limits.

* maxDataSources: UnsignedInt|null

If present, gives the maximum number of of DataSourceObjects allowed per creation in a Blob/upload. Servers MUST allow at least 64 items.

If this value is not present or null, then clients SHOULD assume a limit of 64 items.

* supportedTypeNames: String[]|null

An array of data type names that are supported for Blob/lookup. If the server does not support lookups then this could be the empty list, not present or null.

* supportedDigestAlgorithms String[]|null

An array of supported digest algorithms that are supported for Blob/get. If the server does not support calculating blob digests, then this could be the empty list, not present, or null. Algorithms on this list _MUST_ be present in the HTTP Digest Algorithms registry defined by [RFC3230], and are always lowercased.

3.1.1. Capability Example
4. Blob Methods

A blob is a sequence of zero or more octets.

The JMAP base spec [RFC8620] defines the Blob/copy method, which is unchanged by this specification, and is selected by the urn:ietf:params:jmap:core capability.

The following JMAP Methods are selected by the urn:ietf:params:jmap:blob capability.

4.1. Blob/upload

This is similar to a Foo/set from [RFC8620] in some ways, however blobs can’t be updated or deleted, so only create is allowed in the method call, and blobs don’t have state, so there is no state field present in the method response.

*Parameters*

  * accountId: Id
The id of the account in which the blobs will be created.

* create: Id[UploadObject]

A map of creation id to UploadObjects.

*Result*

The result is the same as for Foo/set in RFC8620, with created and notCreated objects mapping from the creationId.

The created objects contain:

* id: Id

  the blobId which was created

* type: String|null

  the media type as given in the creation (if any); or detected from content; or null

* size: UnsignedInt

  as per RFC8620 - the size of the created blob in octets

Plus any other properties identical to those that would be returned in the JSON response of the RFC8620 upload endpoint (which may be extended in the future - this document anticipates that implementations will extend both the upload endpoint and the Blob/upload responses in the same way)

Or if there is a problem with a creation, then the server will return a notCreated response with a map from the failed creationId to a SetError object.

For each successful upload, servers MUST add an entry to the creationIds map for the request. This allows the blob id to be used via back-reference in subsequent method calls.

*UploadObject*

* data: DataSourceObject[]

  an array of zero or more octet sources in order (zero to create an empty blob). The result of each of these sources is concatenated together in order to create the blob.
* type: String|null (default: null)
  hint for media type of the data

*DataSourceObject*

Exactly one of:

* data:asText: String|null (raw octets, must be UTF-8)
* data:asBase64: String|null (base64 representation of octets)

or a blobId source:

* blobId: Id
* offset: UnsignedInt|null (MAY be zero)
* length: UnsignedInt|null (MAY be zero)

If null then offset is assumed to be zero.

If null then length is the remaining octets in the blob.

If the range can not be fully satisfied (i.e. begins or extends past the end of the data in the blob) then the DataSourceObject is invalid and results in a notCreated response for this creation id.

If the data properties have any invalid references or invalid data contained in them, the server MUST NOT guess as to the user’s intent, and MUST reject the creation and return a notCreated response for that creation id.

Likewise, invalid characters in the base64 of data:asBase64, or invalid UTF-8 in data:asText MUST result in a nonCreated response.

It is envisaged that the definition for DataSourceObject might be extended in future, for example to fetch external content.

A server MUST accept at least 64 DataSourceObjects per create, as described in Section 3.1 of this document.

4.1.1. Blob/upload simple example

The data:asBase64 field is set over multiple lines for ease of publication here, however all data:asBase64 would be sent as a continuous string with no whitespace on the wire.
Method Call:

```json
[  
  "Blob/upload",
  {
    "accountId": "account1",
    "create": {
      "1": {
        "data": [
          {
            "data:asBase64": "iVBORw0KGgoAAAANSUhEUgAAAAEAAAABAQMAAAAl21bKA
            AAAA1BMVEX/AAAZ4gk3AAAAAXRSTlN/gFy0ywAAAAApJRE
            FUEJxjYgAAAAYAAzY3fKgAAAAASUVORK5CYII=",
          }
        ],
        "type": "image/png"
      }
    }
  },
  "R1"
]
```

Response:

```json
[  
  "Blob/upload",
  {
    "accountId": "account1",
    "created": {
      "1": {
        "id": "G4c6751edf9dd6903ff54b792e432fba781271beb",
        "type": "image/png",
        "size": 95
      }
    }
  },
  "R1"
]
```

4.1.2. Blob/upload complex example

Method Calls:

```json
[  
  [  
    "Blob/upload",
    {
      "create": {
      
```
"b4": {
  "data": [
    {
      "data:asText": "The quick brown fox jumped over the lazy dog."
    }
  ]
},
"S4"
],
[  "Blob/upload",
  {
    "create": {
      "cat": {
        "data": [
          {
            "data:asText": "How"
          },
          {
            "blobId": "#b4",
            "length": 7,
            "offset": 3
          },
          {
            "data:asText": "was t"
          },
          {
            "blobId": "#b4",
            "length": 1,
            "offset": 1
          },
          {
            "data:asBase64": "YXQ/"
          }
        ]
      }
    }
  }
],
"CAT"
],
[  "Blob/get",
  {
    "properties": [
      "data:asText",
      "size"
The image contains a snippet of JSON code that describes responses to a JMAP Blob operation. The snippet includes details about the actions taken in response to a blob upload and retrieval.

Responses:

```
[
  [
    "Blob/upload",
    {
      "oldState": null,
      "created": {
        "b4": {
          "id": "Gc0854fb9fb03c41cce3802cb0d220529e6eef94e",
          "size": 45,
          "type": "application/octet-stream"
        }
      },
      "notCreated": null,
      "accountId": "account1"
    },
    "G4"
  ],
  [
    "Blob/upload",
    {
      "oldState": null,
      "created": {
        "cat": {
          "id": "Gcc60576f036321ae6e8037ffc56bdee589bd3e23",
          "size": 19,
          "type": "application/octet-stream"
        }
      },
      "notCreated": null,
      "accountId": "account1"
    },
    "CAT"
  ],
  [
    "Blob/get",
    {
      "list": [
```
4.2. Blob/get

A standard JMAP get, with two additional optional parameters:

- offset: UnsignedInt|null
  
  start this many octets into the blob data. If null or unspecified, this defaults to zero.

- length: UnsignedInt|null
  
  return at most this many octets of the blob data. If null or unspecified, then all remaining octets in the blob are returned. This can be considered equivalent to an infinitely large length value, except that the isTruncated warning is not given unless the start offset is past the end of the blob.

*Request Properties:*

Any of

- data:asText
- data:asBase64
- data (returns data:asText if the selected octets are valid UTF-8, or data:asBase64)
- digest:<algorithm> (where <algorithm> is one of the named algorithms in the supportedDigestAlgorithms capability)
- size

If not given, properties defaults to data and size.

*Result Properties:*

- data:asText: String|null
the raw octets of the selected range if they are valid UTF-8, otherwise null

* data:asBase64: String
  the base64 encoding of the octets in the selected range

* digest:<algorithm> String
  the base64 encoding of the digest of the octets in the selected range, calculated using the named algorithm

* isEncodingProblem: Boolean (default: false)

* isTruncated: Boolean (default: false)

* size: UnsignedInt
  the number of octets in the entire blob

The size value MUST always be the number of octets in the underlying blob, regardless of offset and length.

The data fields contain a representation of the octets within the selected range that are present in the blob. If the octets selected are not valid UTF-8 (including truncating in the middle of a multi-octet sequence) and data or data:asText was requested, then the key isEncodingProblem MUST be set to true and the data:asText response value MUST be null. In the case where data was requested and the data is not valid UTF-8, then data:asBase64 MUST be returned.

If the selected range requests data outside the blob (i.e. the offset+length is larger than the blob) then the result is either just the octets from the offset to the end of the blob, or an empty string if the offset is past the end of the blob. Either way, the isTruncated property in the result MUST be set to true to tell the client that the requested range could not be fully satisfied. If digest was requested, any digest is calculated on the octets that would be returned for a data field.

Servers SHOULD store the size for blobs in a format which is efficient to read, and clients SHOULD limit their request to just the size parameter if that is all they need, as fetching blob content could be significantly more expensive and slower for the server.
4.2.1. Blob/get simple example

Where a blob containing the string "The quick brown fox jumped over the lazy dog!" has blobId G6ec94756e3e046be78fcb33953b85b944e70673e.

The first method call requests just the size for multiple blobs, and the second requests both size and a short range of the data for one of the blobs.

Method Calls:

```
[ [
  "Blob/get",
  { "ids" : [
    "G6ec94756e3e046be78fcb33953b85b944e70673e",
    "not-a-blob"
  ],
  "properties" : [ "size" ]
},
  "R1"
],
[ "Blob/get",
  { "accountId" : "account1",
    "ids" : [ "G6ec94756e3e046be78fcb33953b85b944e70673e" ],
    "properties" : [ "data:asText", "data:asBase64", "size" ],
    "offset" : 4,
    "length" : 9
  },
  "R2"
]
```

Responses:

```
[
  [ ]
]`
"Blob/get",
{
   "accountId": "account1",
   "list": [
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
* G4: since the requested range could be satisfied as text, both
blobs are returned as data:asText and there is no encoding
problem.

* G5: both blobs cannot satisfy the requested range, so isTruncated
is true for both.

Note: some values have been wrapped for line length - there would be
no whitespace in the data:asBase64 values on the wire

Method calls:

```json
[
  [
    "Blob/upload",
    {
      "create": {
        "b1": {
          "data": [
            {
              "data:asBase64": "VGhlIHF1aWNrIGJyb3duIGZveCBqdW1wZWQgb3ZlciB0aGUggerZG9nLg=="
            }
          ]
        },
        "b2": {
          "data": [
            {
              "data:asText": "hello world"
            }
          ],
          "type": "text/plain"
        }
      }
    },
    "S1"
  ],
  [
    "Blob/get",
    {
      "ids": [
        "#b1",
        "#b2"
      ]
    },
    "G1"
  ]
]
```
"Blob/get",
{
  "ids": [
    "#b1",
    "#b2"
  ],
  "properties": [
    "data:asText",
    "size"
  ]
},
"G2"
],
[
  "Blob/get",
  {
    "ids": [
      "#b1",
      "#b2"
    ],
    "properties": [
      "data:asBase64",
      "size"
    ]
  },
  "G3"
],
[
  "Blob/get",
  {
    "offset": 0,
    "length": 5,
    "ids": [ 
      "#b1",
      "#b2"
    ]
  },
  "G4"
],
[
  "Blob/get",
  {
    "offset": 20,
    "length": 100,
    "ids": [
      "#b1",
      "#b2"
    ]
  }
]
Responses:

[  
  "Blob/upload",  
  {  
    "oldState": null,  
    "created": {  
      "b2": {  
        "id": "G2aae6c35c94fcb415dbe95f408b9ce91ee846ed",  
        "size": 11,  
        "type": "application/octet-stream"  
      },  
      "b1": {  
        "id": "G72cfa4804194563685d9a4b695f7ba20e7739576",  
        "size": 43,  
        "type": "text/plain"  
      }  
    },  
    "updated": null,  
    "destroyed": null,  
    "notCreated": null,  
    "notUpdated": null,  
    "notDestroyed": null,  
    "accountId": "account1"  
  },  
  "S1"  
],  
[  
  "Blob/get",  
  {  
    "list": [  
      {  
        "id": "G72cfa4804194563685d9a4b695f7ba20e7739576",  
        "isEncodingProblem": true,  
        "data:asBase64": "VGhlIHF1aWNrIGJyb3duIGZveCBqdW1wZWQgb3ZlciB0aGUggYEgZG9nLg==",  
        "size": 43  
      },  
      {  
        "id": "G2aae6c35c94fcb415dbe95f408b9ce91ee846ed",  
        "data:asText": "hello world",  
        "size": 11  
      }  
    ]  
  }  
]
"G1",
"Blob/get",
{
"list": [
{
"id": "G72cfa4804194563685d9a4b695f7ba20e7739576",
"isEncodingProblem": true,
"size": 43
},
{
"id": "G2aae6c35c94fcfb415dbe95f408b9ce91ee846ed",
"data:asText": "hello world",
"size": 11
}
],
"notFound": [],
"accountId": "account1"
},
"G2",
"Blob/get",
{
"list": [
{
"id": "G72cfa4804194563685d9a4b695f7ba20e7739576",
"data:asBase64": "VGhlIHF1aWNrIGJyb3duIGZveCBqdW1wZWQgb3ZlciB0aGUggYEgZG9nLg==",
"size": 43
},
{
"id": "G2aae6c35c94fcfb415dbe95f408b9ce91ee846ed",
"data:asBase64": "aGVsbG8gd29ybGQ=",
"size": 11
}
],
"notFound": [],
"accountId": "account1"
},
"G3"
]
[  "Blob/get",  
  {  
    "list": [  
      {  
        "id": "G72cfa4804194563685d9a4b695f7ba20e7739576",  
        "data:asText": "The q",  
        "size": 43  
      },  
      {  
        "id": "G2aae6c35c94fcfb415dbe95f408b9ce91ee846ed",  
        "data:asText": "hello",  
        "size": 11  
      }  
    ],  
    "notFound": [],  
    "accountId": "account1"  
  },  
  "G4"  
],  
[  "Blob/get",  
  {  
    "list": [  
      {  
        "id": "G72cfa4804194563685d9a4b695f7ba20e7739576",  
        "isTruncated": true,  
        "isEncodingProblem": true,  
        "data:asBase64": "anVtcGVkIG92ZXIgdGhlIIGBIGRvZy4=",  
        "size": 43  
      },  
      {  
        "id": "G2aae6c35c94fcfb415dbe95f408b9ce91ee846ed",  
        "isTruncated": true,  
        "data:asText": "",  
        "size": 11  
      }  
    ],  
    "notFound": [],  
    "accountId": "account1"  
  },  
  "G5"  
]
4.3. Blob/lookup

Given a list of blobIds, this method does a reverse lookup in each of the provided type names to find the list of Ids within that data type which reference the provided blob.

The definition of reference is somewhat loosely defined, but roughly means "you could discover this blobId by looking inside this object", for example if a Mailbox contains an Email which references the blobId, then it references that blobId. Likewise for a Thread.

*Parameters*

* accountId: Id

The id of the account used for the call.

* typeNames: String[]

A list of names from the "JMAP Data Types" registry. Only names for which "Can reference blobs" is true may be specified, and the capability which defines each type must also be used by the overall JMAP request in which this method is called.

If a type name is not known by the server, or the associated capability has not been requested, then the server returns an "unknownDataType" error.

* ids: Id[]

A list of blobId values to be looked for.

*Response*

* list: BlobInfo[]

A list of BlobInfo objects.

*BlobInfo Object*

* id: Id

The Blob Identifier.

* matchedIds: String[Id[]]

A map from type name to list of Ids of that data type (e.g. the name "Email" maps to a list of emailIds)
If a blob is not visible to a user at all, then the server SHOULD return that blobId in the notFound array, however it may also return an empty list for each type name, as it may not be able to know if other data types do reference that blob.

4.3.1. Blob/lookup example
Method call:

[ "Blob/lookup",
  {
    "typeNames": [
      "Mailbox",
      "Thread",
      "Email"
    ],
    "ids": [
      "Gd2f81008cf07d2425418f7f02a3ca63a8bc82003",
      "not-a-blob"
    ],
    "R1"
  }
]

Response:

[ "Blob/lookup",
  {
    "list": [
      {
        "id": "Gd2f81008cf07d2425418f7f02a3ca63a8bc82003",
        "matchedIds": {
          "Mailbox": [
            "M54e97373",
            "Mcbe6b662"
          ],
          "Thread": [
            "T1530616e"
          ],
          "Email": {
            "E16e70a73eb4",
            "E84b0930cf16"
          }
        }
      },
      "notFound": ["not-a-blob"
    ],
    "R1"
  }
]
5. Security considerations

JSON parsers are not all consistent in handling non-UTF-8 data. JMAP requires that all JSON data be UTF-8 encoded, so servers MUST only return a null value if data:asText is requested for a range of octets which is not valid UTF-8, and set isEncodingProblem: true.

Servers MUST apply any access controls, such that if the authenticated user would be unable to discover the blobId by making queries, then this fact can’t be discovered via a Blob/lookup. For example, if an Email exists in a Mailbox which the authenticated user does not have access to see, then that emailId MUST not be returned in a lookup for a blob which is referenced by that email.

If a server might sometimes return all names empty rather than putting a blobId in the notFound response to a Blob/get, then the server SHOULD always return the same type of response, regardless of whether a blob exists but the user can’t access it, or doesn’t exist at all. This avoids leaking information about the existence of the blob.

The server MUST NOT trust that the data given to a Blob/upload is a well formed instance of the specified media type, and if the server attempts to parse the given blob, only hardened parsers designed to deal with arbitrary untrusted data should be used. The server SHOULD NOT reject data on the grounds that it is not a valid specimen of the stated type.

Blob/upload with carefully chosen data sources can be used to recreate dangerous content on the far side of security scanners (anti-virus or exfiltration scanners for example) which may be watching the upload endpoint. Server implementations SHOULD provide a hook to allow security scanners to check the resulting blob after concatenating the data sources in the same way that they do for the upload endpoint.

6. IANA considerations

6.1. JMAP Capability registration for "blob"

IANA is requested to register the "blob" JMAP Capability as follows:

Capability Name: urn:ietf:params:jmap:blob

Specification document: this document

Intended use: common
Change Controller: IETF

Security and privacy considerations: this document, Section XXX

6.2. JMAP Error Codes Registration for "unknownDataType"

IANA is requested to register the "unknownDataType" JMAP Error Code as follows:

JMAP Error Code: unknownDataType

Intended use: common

Change Controller: IETF

Reference: this document

Description: The server does not recognise this data type, or the capability to enable it was not present.

6.3. Creation of "JMAP Data Types" Registry

IANA is requested to create a new registry "JMAP Data Types" with the initial content:
<table>
<thead>
<tr>
<th>Type Name</th>
<th>Can</th>
<th>Can</th>
<th>Capability</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>reference</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Can</td>
<td></td>
<td></td>
<td>blobs</td>
<td>for</td>
</tr>
<tr>
<td>use</td>
<td></td>
<td></td>
<td>state</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>change</td>
<td></td>
</tr>
</tbody>
</table>

| Core            | No     | No     | urn:ietf:params:jmap:core            | [RFC8620] |
| PushSubscription| No     | No     | urn:ietf:params:jmap:core            | [RFC8620] |
| Mailbox         | Yes    | Yes    | urn:ietf:params:jmap:mail            | [RFC8621] |
| Thread          | Yes    | Yes    | urn:ietf:params:jmap:mail            | [RFC8621] |
| Email           | Yes    | Yes    | urn:ietf:params:jmap:mail            | [RFC8621] |
| EmailDelivery   | No     | Yes    | urn:ietf:params:jmap:mail            | [RFC8621] |
| SearchSnippet   | No     | No     | urn:ietf:params:jmap:mail            | [RFC8621] |
| Identity        | No     | Yes    | urn:ietf:params:jmap:submission      | [RFC8621] |
| EmailSubmission | No     | Yes    | urn:ietf:params:jmap:submission      | [RFC8621] |
| VacationResponse| No     | Yes    | urn:ietf:params:jmap:vacationresponse| [RFC8621] |
| MDN             | No     | No     | urn:ietf:params:jmap:mdn             | [RFC9007] |
Table 1

This policy for this registry is "Specification required", either an RFC or a similarly stable reference document which defines a JMAP Data Type and associated capability.

7. Changes

EDITOR: please remove this section before publication.

The source of this document exists on github at:
https://github.com/brong/draft-gondwana-jmap-blob/
(https://github.com/brong/draft-gondwana-jmap-blob/)

*draft-ietf-jmap-blob-12*

* updates based on Neil Jenkins’ feedback:
  - fixed [] positions for type specs
documented delta between /upload and /set better
- allowed zero-length blobId sources
- fixed examples with /set leftovers
- documented datatypes registry policy
* added optional "digest" support

*draft-ietf-jmap-blob-11*:
* updates based on IETF113 feedback:
  - added wording to suggest the a Blob/get of just size might be faster
  - added an example with just the size field being selected

*draft-ietf-jmap-blob-10*:
* removed remaining references to catenate.

*draft-ietf-jmap-blob-09*:
* tidied up introduction text
* replaced Blob/set with Blob/upload
* made all upload creates take an array of sources to normalise behaviour at the cost of a slightly more complex default case.

*draft-ietf-jmap-blob-08*:
* Fixed spelling of Neil’s name in acknowledgements
* Last call review (thanks Jim Fenton)
  - fixed mmark sillyness causing RFC8620 to be non-normative in the references
  - clarified the capability object and accountCapability object requirements
  - made capability keys much more tightly defined, with mandatory minimum catenate limit and default values.
  - increased use of normative language generally
  - lowercased ’blob’ anywhere it wasn’t explicitly the object
  - lowercased titles of the columns in the registry

*draft-ietf-jmap-blob-07*:
* more examples to cover the interactions of offset, length and encoding checks.

*draft-ietf-jmap-blob-06*:
* removed asHex - we only need base64 and text
* added reference to where base64 is defined
* made ’destroy’ not be allowed
* expanded JSON examples for readability
* removed 'expires' from examples

*draft-ietf-jmap-blob-05*:
* discovered I hadn’t actually included typeNames and matchedIds anywhere except the updates section, oops!
* added a catenate example
* tightened up some text

*draft-ietf-jmap-blob-04*:
* added security considerations for scanning catenate results

*draft-ietf-jmap-blob-03*:
* added capabilities object
* renamed types to typeNames and matchedIds
* added details of how to handle non-UTF8 data and truncation in Blob/get
* added isTruncated and isEncodingProblem to Blob/get to tell the client if the request wasn’t entirely satisfied.

*draft-ietf-jmap-blob-02*:
* fixed incorrect RFC number in reference and HTTP PUT -> POST, thanks Ken.
* added acknowledgements section
* removed all 'datatype' text and changed to 'data type' or 'type name' as appropriate (issue #1 proposal)
* expanded security considerations section and moved optional Blob/lookup empty case into Blob/lookup section

*draft-ietf-jmap-blob-01*:
* renamed 'datatypes' to 'types' to align with PushSubscription from RFC8620.
* added example for Blob/get
* specified offset and length precisely

*draft-ietf-jmap-blob-00*:
* initial adoption as an IETF document, otherwise identical to draft-gondwana-jmap-blob-02

*draft-gondwana-jmap-blob-02*:
* renamed 'objects' to 'datatypes'
* specified Blob/lookup
* added IANA registry for datatypes
*draft-gondwana-jmap-blob-01*
* added an example
*draft-gondwana-jmap-blob-00*
* initial proposal

8. Acknowledgements

Joris Baum, Jim Fenton, Neil Jenkins, Alexey Melnikov, Ken Murchison, Robert Stepanek and the JMAP working group at the IETF.

9. Normative References


10. Informative References


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
JMAP for Calendars
draft-ietf-jmap-calendars-08

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 28 August 2022.

Copyright Notice

Copyright (c) 2022 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 Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.
Table of Contents

1.  Introduction ............................................. 4
  1.1.  Notational Conventions ................................. 4
  1.2.  The LocalDate Data Type ................................. 4
  1.3.  Terminology ........................................... 4
  1.4.  Data Model Overview .................................... 5
     1.4.1.  UIDs and CalendarEvent Ids ......................... 6
  1.5.  Addition to the Capabilities Object .................... 6
     1.5.1.  urn:ietf:params:jmap:calendars .................... 6
     1.5.2.  urn:ietf:params:jmap:calendars:preferences ......... 7
     1.5.3.  urn:ietf:params:jmap:principals:availability ....... 7

2.  Principals and Sharing .................................... 8
  2.1.  Principal Capability urn:ietf:params:jmap:calendars .... 8
  2.2.  Principal/getAvailability ............................... 8

3.  Participant Identities .................................... 10
  3.1.  ParticipantIdentity/get ................................ 11
  3.2.  ParticipantIdentity/changes ............................ 11
  3.3.  ParticipantIdentity/set ................................ 11

4.  Calendars ................................................. 11
  4.1.  Calendar/get ........................................... 15
  4.2.  Calendar/changes ....................................... 16
  4.3.  Calendar/set ........................................... 16

5.  Calendar Events ........................................... 17
  5.1.  Additional JSCalendar properties ....................... 19
     5.1.1.  mayInviteSelf ....................................... 19
     5.1.2.  mayInviteOthers ..................................... 19
     5.1.3.  hideAttendees ....................................... 19
  5.2.  Attachments ............................................. 20
  5.3.  Per-user properties ..................................... 20
  5.4.  Recurring events ....................................... 20
  5.5.  Updating for "this-and-future" ......................... 21
     5.5.1.  Splitting an event .................................. 21
     5.5.2.  Updating the base event and overriding previous .... 21
  5.6.  CalendarEvent/get ..................................... 21
  5.7.  CalendarEvent/changes .................................. 23
  5.8.  CalendarEvent/set ...................................... 23
     5.8.1.  Patching ............................................ 26
     5.8.2.  Sending invitations and responses .................. 29
  5.9.  CalendarEvent/copy ..................................... 32
  5.10. CalendarEvent/query .................................... 32
      5.10.1.  Filtering .......................................... 32
      5.10.2.  Sorting ............................................ 34
  5.11. CalendarEvent/queryChanges ............................. 34
  5.12. Examples ............................................... 34

6.  Alerts ................................................... 34
  6.1.  Default alerts ......................................... 35
  6.2.  Acknowledging an alert .................................. 35
1. Introduction

JMAP ([RFC8620] (U+2013) 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, CalendarEventNotification, and CalendarPreferences (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 [RFC8984] 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.

The CalendarPreferences object is a singleton in the account that stores the user’s default calendar and participant identity.

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 ([RFC8984], 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 (U+2019)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 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:calendars:preferences

This represents support for the CalendarPreferences data type and associated API methods. The value of this property in the JMAP Session capabilities property and the account (U+2019)s accountCapabilities property is an empty object.

Any account with this capability MUST also have the urn:ietf:params:jmap:calendars capability.

1.5.3. 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 (U+2019)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 scheduling 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?
*  *mayShareWith*: Boolean May the user add this principal as a calendar sharee (by adding them to the shareWith property of a calendar, see Section XXX)?
*  *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 [RFC8984], 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 "includeIn Availability" 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.

  A participant identity may be destroyed that is referenced as the "defaultParticipantIdentityId" in the CalendarPreferences object for the same account. Doing so updates the defaultParticipantIdentityId property on the CalendarPreferences to null.

4. Calendars

A Calendar is a named collection of events. All events are associated with at least one calendar.
A *Calendar* object has the following properties:

*  *id*: Id (immutable; server-set) The id of the calendar.
*  *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 background.

*  *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 [RFC8984], 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 [RFC8984], 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.

* `*mayWriteAll*`: Boolean The user may create, modify or destroy all events in this calendar, or move events to or from this calendar. If this is true, the mayWriteOwn, mayUpdatePrivate and mayRSVP properties MUST all also be true.

* `*mayWriteOwn*`: Boolean The user may create, modify or destroy an event on this calendar if either they are the owner of the event or the event has no owner. This means the user may also transfer ownership by updating an event so they are no longer an owner.

* `*mayUpdatePrivate*`: Boolean The user may modify the following properties on all events in the calendar, even if they would not otherwise have permission to modify that event. 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 (updating the recurrenceOverrides property of the event to do so).

* `*mayRSVP*`: Boolean The user may modify the following properties of any Participant object that corresponds to one of the user’s ParticipantIdentity objects in the account, even if they would not otherwise have permission to modify that event:
- participationStatus
- participationComment
- expectReply
- scheduleAgent
- scheduleSequence
- scheduleUpdated

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 (updating the recurrenceOverrides property of the event to do so).

* *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, or if none of the Participant objects have the "owner" role.

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 "shareWith" property 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. Each user gets the default value for these properties as the initial value; they do not inherit an initial value from the calendar owner.

If the calendar account’s "shareesActAs" capability is "self" these properties are instead 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 always stored per-user:

* name
* color
* sortOrder
* isVisible

The "name" and "color" 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 "sortOrder" and "isVisible" properties are initially the default value for each sharee; they are not inherited from the owner.

A calendar may be destroyed that is referenced as the "defaultCalendarId" in the CalendarPreferences object for the same account. Doing so updates the defaultCalendarId property on the CalendarPreferences to null.

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 [RFC8984], with the following additional properties:

* id*: Id (immutable; server-set) The id of the CalendarEvent. The id uniquely identifies a JSEvent with a particular "uid" and "recurrenceId" within a particular account.

* baseEventId*: Id|null (immutable; server-set) This is only defined if the _id_ property is a synthetic id, generated by the server to represent a particular instance of a recurring event (see Section XXX). This property gives the id of the "real" CalendarEvent this was generated from.
* `*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 (default: false) 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 may add themselves to the event as a participant with the "attendee" role. This property MUST NOT be altered in the recurrenceOverrides; it may only be set on the base object.

This indicates the owner will accept "party crasher" RSVPs via iTIP, subject to any other domain-specific restrictions, and users may add themselves to the event via JMAP as long as they have the mayRSVP permission for the calendar.

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 base object.

The mayRSVP permission for the calendar is also required in conjunction with this event property for users to be allowed to make this change via JMAP.

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 base object.
5.2. Attachments

The Link object, as defined in [RFC8984] 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 if set. 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 base 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 base 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 base 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 base event, "CalendarEvent/changes" will show the ids of all the individual instances being destroyed and the id for the base event being created.

5.5. Updating for "this-and-future"

When editing a recurring event, you can either update the base event (affecting all instances unless overridden) 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 base event 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 [RFC8984] 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 base event and overriding previous

For scheduled events, a better approach is to avoid splitting and instead update the base 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 base 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 server-expanded 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 [RFC8984], 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.

Synthetic ids generated by the server expanding recurrences in "CalendarEvent/query" do not appear in "CalendarEvent/changes" responses; only the ids of events as actually stored on the server.

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.

An id may represent a server-expanded single instance of a recurring event if the client asked the server to expand recurrences in "CalendarEvent/query". When the synthetic id for such an instance is given, the server MUST process an update as an update to the recurrence override for that instance on the base event, and a destroy as removing just that instance.
Clients MUST NOT send an update/destroy to both the base event and a synthetic instance in a single "/set" request; the result of this is undefined. Note however, a client may replace a series of explicit instances (each with the same uid but a different recurrenceId property) with the base event (same uid, no recurrenceId) in a single "/set" call. (So the /set will destroy the existing instances and create the new base event.) This will happen when someone is initially invited to a specific instance or instances of a recurring event, then later invited to the whole series.

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 of a JSEvent object allows the owner to override how sharees of the calendar see the event. If this is set to "private", a sharee may not delete or update the event (even if only modifying per-user properties); any attempt to modify such an event MUST be rejected with a forbidden SetError. If set to "secret", the server MUST behave as though the event does not exist for all users other than the owner.

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.

If omitted on create, the server MUST set the following properties to an appropriate value:

* @type
* uid
* created

If (and only if) the server is the source of the event (see Section XXX), 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. If the event is being created and the overridden "updated" time is now earlier than a client-supplied "created" time, the "created" time MUST also be overridden to the server’s time. If the server is not the source of the event it MUST NOT automatically set an "updated" time, as this can break correct processing of iTIP messages.

When updating an event, if all of:
* a property has been changed other than "calendarIds", "isDraft" or a per-user property (see Section XXX); 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:

```json
[[ "CalendarEvent/set", {
  "accountId": "ue150411c",
  "update": {
    "123": {
      "recurrenceOverrides/2018-03-08T09:00:00/
          participants"lem9lQGZvb2GFtcGx1LmNvbQ"1participationStatus":
          "declined"
    }
  }
}, "0" ]]
```

This patches the "2018-03-08T09:00:00" PatchObject in recurrenceOverrides so that it ends up like this:

```json
"recurrenceOverrides": {
  "2018-03-08T09:00:00": {
    "start": "2018-03-08T10:00:00",
    "participants/dG9tQGZvb2Jhci5xlLmNvbQ/participationStatus":
      "declined",
    "participants/em9lQGZvb2GFtcGx1LmNvbQ/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:

```json
[[ "CalendarEvent/set", {
  "accountId": "ue150411c",
  "update": {
    "123": {
      "recurrenceOverrides/2018-03-08T09:00:00/
          participants"dG9tQGZvb2Jhci5xlLmNvbQ"1participationStatus": null
    }
  }
}, "0" ]]
```

However, if you instead want to remove Tom from this instance altogether, you could not send this patch:
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 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:

```json
[[ "CalendarEvent/set", {
    "accountId": "ue150411c",
    "update": {
        "123": {
            "recurrenceOverrides/2018-03-08T09:00:00/" :
            participants`1dG9tQGZvb2Jhci5xlLmNvbQ": null
        }
    }
}, "0" ]]
```

5.8.2. Sending invitations and responses

If "sendSchedulingMessages" is true, the server MUST send appropriate iTIP [RFC5546] scheduling messages after successfully 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 ParticipantIdentity 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 either:

* The event is being created; or
* 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 base event. 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 full 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 any of:

* A participant is removed from either the base 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 every participant corresponding to one of the user’s ParticipantIdentitities in the account if any of the following changes are made:

* The "participationStatus" property of the participant is changed, either for the base event or a specific instance, to any value other than "needs-action".
* The event is created and the participationStatus is not "needs-action".
* The event is destroyed and the participationStatus was not "needs-action".
If the participationStatus property is changed for just a single instance of the event (i.e., set in recurrenceOverrides), the REPLY message SHOULD be sent for just that recurrence id.

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 [RFC8984], 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 receiving 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 base 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. CalendarPreferences

A CalendarPreferences object stores information about the principal’s preferences and defaults.

* `id`: Id (immutable; server-set) The id of the object. There is only ever one CalendarPreferences object, and its id is "singleton".

* `defaultCalendarId`: Id|null The id of the principal’s default calendar. If set, clients should default to this calendar when creating new events in this account, unless overridden by a local preference. When the principal is invited to an event, this is the calendar to which it will be added by the server.

If null, no default is defined and clients/servers may choose any calendar.

* `defaultParticipantIdentityId`: Id|null The default participant identity to use for the principal when adding participants to an event. If set, when the user adds an invitee to an event without an owner, the client should use this participant identity to add the principal as an owner participant of the event.

If null, no default is defined and clients/servers may choose any participant identity.

The following JMAP methods are supported.

8.1. CalendarPreferences/get

This is a standard "/get" method as described in [RFC8620], Section 5.1.

There MUST only be exactly one CalendarPreferences object in an account. It MUST have the id "singleton".
8.2. CalendarPreferences/set

This is a standard "/set" method as described in [RFC8620], Section 5.3. There is always exactly one CalendarPreferences object in an account; it cannot be created or destroyed, only updated.

9. Security Considerations

All security considerations of JMAP [RFC8620] and JSCalendar [RFC8984] apply to this specification. Additional considerations specific to the data types and functionality introduced by this document are described in the following subsections.

9.1. Privacy

Calendars often contain the precise movements, activities, and contacts of people, and is therefore intensely private data. Privacy leaks can have real world consequences, and calendar servers and clients MUST be mindful of the need to keep all data secure.

Servers MUST enforce the ACLs set on calendars to ensure only authorised data is shared. The additional restrictions specified by the "privacy" property of a JSEvent object (see [RFC8984] Section 4.4.3) MUST also be enforced.

Users may have multiple Participant Identities that they use for areas of their life kept private from one another. Using one identity with an event MUST NOT leak the existence of any other identity. For example, sending an RSVP from identity worklife@example.com MUST NOT reveal anything about another identity present in the account such as privatelife@example.org.

Servers SHOULD enforce that invitations sent to external systems are only transmitted via secure encrypted and signed connections to protect against eavesdropping and modification of data.

9.2. Spoofing

When receiving events and updates from external systems, it can be hard to verify that the identity of the author is who they claim to be. When receiving events via email, DKIM [RFC6376] and S/MIME [RFC8551] are two mechanisms that may be used to verify certain properties about the email data, which can be correlated with the event information.
9.3. Denial-of-service

There are many ways in which a calendar user can make a request liable to cause a calendar server to spend an inordinate amount of processing time. Care must be taken to limit resources allocated to any one user to ensure the system does not become unresponsive. The following subsections list particularly hazardous areas.

9.3.1. 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, and limit the number of expansions they will create. 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.

9.3.2. Firing alerts

An alert firing for an event can cause a notification to be pused to the user’s devices, or to send them an email. Servers MUST rate limit the number of alerts sent for any one user. The combination of recurring events with multiple alerts can in particular define unreasonably frequent alerts, leading to denial of service for either the server processing them or the user’s devices receiving them.

Similarly, clients generating alerts from the data on device must take the same precautions.

The "email" alert type (see RFC8984, Section 4.5.2) causes an email to be sent when triggered. Clients MUST ignore this alert type; the email is sent only by the calendar server. There is no mechanism in JSCalendar to specify a particular email address: the server MUST only allow alerts to be sent to an address it has verified as belonging to the user to avoid this being used as a spamming vector.

9.3.3. Load spikes

Since most events are likely to start on the hour mark, a large spike of activity is often seen at these times, with particularly large spikes at certain common times in the time zone of the server’s user base. In particular, a large number of alerts (across different users and events) will be triggered at the same time. Servers may mitigate this somewhat by adding jitter to the triggering of the alerts; it is RECOMMENDED to fire them slightly early rather than slightly late if needed to spread load.
9.4. Spam

Invitations received from an untrusted source may be spam. If this is added to the user’s calendar automatically it can be very obtrusive, especially if it is a recurring event that now appears every day. Incoming invitations to events should be subject to spam scanning, and suspicious events should not be added to the calendar automatically.

Servers should strip any alerts on invitations when adding to the user’s calendar; the useDefaultAlerts property should be set instead to apply the user’s preferences.

Similarly, a malicious user may use a calendar system to send spam by inviting people to an event. Outbound iTIP should be subject to all the same controls used on outbound email systems, and rate limited as appropriate. A rate limit on the number of distinct recipients as well as overall messages is recommended.

10. IANA Considerations

10.1. 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

10.2. JMAP Capability Registration for "calendars:preferences"

IANA will register the "calendars:preferences" JMAP Capability as follows:

Capability Name: urn:ietf:params:jmap:calendars:preferences

Specification document: this document

Intended use: common

Change Controller: IETF
10.3. JMAP Capability Registration for "principals:availability"

IANA will register the "principals:availability" JMAP Capability as follows:

Capability Name: urn:ietf:params:jmap:principals:availability

Specification document: this document

Intended use: common

Change Controller: IETF

Security and privacy considerations: this document, Section XXX

10.4. JSCalendar Property Registrations

IANA will register the following additional properties in the JSCalendar Properties Registry.

10.4.1. id

Property Name: id

Property Type: Id

Property Context: JSEvent, JSTask

Intended Use: Reserved

10.4.2. calendarIds

Property Name: calendarIds

Property Type: Id[Boolean]

Property Context: JSEvent, JSTask

Intended Use: Reserved

10.4.3. isDraft

Property Name: isDraft

Property Type: Boolean
Property Context: JSEvent, JSTask
Intended Use: Reserved

10.4.4. utcStart

Property Name: utcStart
Property Type: UTCDatetime
Property Context: JSEvent, JSTask
Intended Use: Reserved

10.4.5. utcEnd

Property Name: utcEnd
Property Type: UTCDatetime
Property Context: JSEvent, JSTask
Intended Use: Reserved

10.4.6. mayInviteSelf

Property Name: mayInviteSelf
Property Type: Boolean (default: false)
Property Context: JSEvent, JSTask
Reference: This document, Section XXX.
Intended Use: Common

10.4.7. mayInviteOthers

Property Name: mayInviteOthers
Property Type: Boolean (default: false)
Property Context: JSEvent, JSTask
Reference: This document, Section XXX.
Intended Use: Common
10.4.8. hideAttendees

Property Name: hideAttendees

Property Type: Boolean (default: false)

Property Context: JSEvent, JSTask

Reference: This document, Section XXX.

Intended Use: Common

11. Normative References


12. Informative References
[RFC4791]  Daboo, C., Desruisseaux, B., and L. Dusseault,
  "Calendaring Extensions to WebDAV (CalDAV)", RFC 4791,
  DOI 10.17487/RFC4791, March 2007,

[RFC5546]  Daboo, C., Ed., "iCalendar Transport-Independent
  Interoperability Protocol (iTIP)", RFC 5546,
  DOI 10.17487/RFC5546, December 2009,

  Interoperability Protocol (iMIP)", RFC 6047,
  DOI 10.17487/RFC6047, December 2010,

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
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 11 August 2022.

Copyright Notice

Copyright (c) 2022 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 Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1.  Introduction ............................................. 2
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 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

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

The term Quota (with that specific capitalization) is used to refer to the data type defined in this document and instance of that data type.

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. Servers supporting this specification MUST add a property called urn::ietf::params::jmap::quota 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.

1.4. Data types

In addition to the standard JSON data types, a couple of additional data types are common to the definition of Quota objects and properties.

1.4.1. Scope

The *Scope* is a String from an enumeration defined list of values, handled by the server.

It explains the entities this value applies to. Values for the *Scope* are:

* account: Applies for this account
* domain: All accounts of this domain share this part of the quota
* global: All accounts of this server share this part of the quota
1.4.2. ResourceType

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. Values for the *ResourceType* are:

* count: The quota is measured in number of data type objects. For example, a quota can have a limit of 50 Mail objects.

* octets: The quota is measured in size (in octets). For example, a quota can have a limit of 25000 octets.

1.5. 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

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.

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 defined quota. Computation of this value is handled by the server.

* *limit*: UnsignedInt The hard limit set by this quota. Objects in scope may not be created or updated if we reach this limit. It should be 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 entity about to reach the hard limit soon, but with no action taken yet. 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 still allow some operations, but refusing some others. What is allowed or not is up to the server. 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 and data types this quota applies to.

2.1. Quota/get

Standard "/get" method as described in [RFC8620] section 5.1. The ids argument may be null to fetch all at once.

2.2. 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.

2.3. 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*

2.4. Quota/queryChanges

This is a standard "/queryChanges" method as described in [RFC8620], Section 5.6.

2.5. Examples

2.5.1. Fetching quotas

Request fetching all quotas related to an account:

```json
[[ "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"]]
```

2.5.2. Requesting latest quota changes

Request fetching the changes for a specific quota:

```
[[ "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"]]
```

With response:
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


Author's Address

René Cordier (editor)
Linagora Vietnam
5 Dien Bien Phu
Hanoi
10000
Vietnam

Email: rcordier@linagora.com
URI:   https://linagora.vn
Abstract

This document specifies a data model for sharing data between users 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 11 August 2022.

Copyright Notice

Copyright (c) 2022 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 Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction ........................................... 2
JMAP ([RFC8620]) (U+2013) 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 to represent entities in a collaborative environment and a framework for sharing data between them that can be used to provide a consistent sharing model for different data types. It does not define _what_ may be shared, or the granularity of permissions, as this will depend on the data in question.

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 terms Principal, and ShareNotification (with these specific capitalizations) are used to refer to the data types defined in this document and instances of those data types.

1.3. Data Model Overview

A Principal (see Section XXX) represents an individual, team or resource (e.g., a room or projector). The object contains information about the entity being represented, such as a name, description and time zone. It may also hold domain-specific information. A Principal may be associated with zero or more Accounts (see [RFC8620], Section 1.6.2) containing data belonging to the principal. Managing the set of principals within a system is out of scope for this specification, as it is highly domain specific.

Data types may allow users to share data with others by assigning permissions to principals. When a user’s permissions are changed, a ShareNotification object is created for them so a client can inform the user of the changes.
1.4. Subscriptions

Permissions determine whether a user _may_ access data, but not whether they _want_ to. Some shared data is of equal importance as the user’s own, while other data is just there should the user wish to explicitly go find it. Clients will often want to differentiate the two; for example, a company may share mailing list archives for all departments with all employees, but a user may only generally be interested in the few they belong to. They would have _permission_ to access many mailboxes, but can _subscribe_ to just the ones they care about. The client would provide separate interfaces for reading mail in subscribed mailboxes and browsing all mailboxes they have permission to access in order to manage their subscriptions.

The JMAP Session object (see [RFC8620], Section 2) typically includes an object in the accounts property for every account that the user has access to. Collaborative systems may share data between a very large number of Principals, most of which the user does not care about day-to-day. The Session object MUST only include Accounts where either the user is subscribed to at least one record (see [RFC8620], Section 1.6.3) in the account, or the account belongs to the user. StateChange events for changes to data SHOULD only be sent for data the user has subscribed to and MUST NOT be sent for any Account where the user is not subscribed to any records in the account, except where that account belongs to the user.

The server MAY reject the user’s attempt to subscribe to some resources even if they have permission to access them, e.g., a calendar representing a location.

A user may query the set of Principals they have access to with "Principal/query" (see Section XXX). The Principal object may then provide Account objects if the user has permission to access data for that principal, even if they are not yet subscribed.

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:principals

Represents support for the Principal and ShareNotification 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 (U+2019)s accountCapabilities property is an object that MUST contain the following information on server capabilities and permissions for that account:

* currentUserPrincipalId*: Id|null The id of the principal in this account that corresponds to the user fetching this object, if any.

1.5.2. urn:ietf:params:jmap:principals:owner

This URI is solely used as a key in an account (U+2019)s accountCapabilities property; it does not appear in the JMAP Session capabilities. Support is implied by the urn:ietf:params:jmap:principals session capability.

If present, the account (and data therein) is owned by a principal. Some accounts may not be owned by a principal (e.g., the account that contains the data for the principals themselves), in which case this property is omitted.

The value of this property is an object with the following properties:

* accountIdForPrincipal*: Id The id of an account with the urn:ietf:params:jmap:principals capability that contains the corresponding Principal object.
* principalId*: Id The id of the principal that owns this account.

2. Principals

A Principal represents an individual, group, location (e.g. a room), resource (e.g. a projector) or other entity in a collaborative environment. Sharing in JMAP is generally configured by assigning rights to certain data within an account to other principals, for example a user may assign permission to read their calendar to a principal representing another user, or their team.

In a shared environment such as a workplace, a user may have access to a large number of principals.

In most systems the user will have access to a single Account containing Principal objects, but they may have access to multiple if, for example, aggregating data from different places.

A *Principal* object has the following properties:

* id*: Id The id of the principal.
* type*: String This MUST be one of the following values:
- individual: This represents a single person.
- group: This represents a group of people.
- resource: This represents some resource, e.g. a projector.
- location: This represents a location.
- other: This represents some other undefined principal.
*  *name*: String The name of the principal, e.g. "Jane Doe", or "Room 4B".
*  *description*: String|null A longer description of the principal, for example details about the facilities of a resource, or null if no description available.
*  *email*: String|null An email address for the principal, or null if no email is available.
*  *timeZone*: String|null The time zone for this principal, if known. If not null, the value MUST be a time zone id from the IANA Time Zone Database TZDB (https://www.iana.org/time-zones).
*  *capabilities*: String[Object] A map of JMAP capability URIs to domain specific information about the principal in relation to that capability, as defined in the document that registered the capability.

2.1. Principal/get

This is a standard "/get" method as described in [RFC8620], Section 5.1.

2.2. Principal/changes

This is a standard "/changes" method as described in [RFC8620], Section 5.2.

2.3. Principal/set

This is a standard "/set" method as described in [RFC8620], Section 5.3.

Users SHOULD be allowed to update the "name", "description" and "timeZone" properties of the Principal with the same id as the "currentUserPrincipalId" in the Account capabilities.

However, the server may, and probably will, reject any change with a forbidden SetError. Managing principals is likely tied to a directory service or some other vendor-specific solution, and may occur out-of-band, or via an additional capability defined elsewhere.

2.4. Principal/query

This is a standard "/query" method as described in [RFC8620], Section 5.5
2.4.1. Filtering

A *FilterCondition* object has the following properties:

* *accountIds*: String[] A list of account ids. The Principal matches if the value for its accountId property is in this list.
* *email*: String Looks for the text in the email property.
* *name*: String Looks for the text in the name property.
* *text* String Looks for the text in the name, email, and description properties.
* *type*: String The type must be exactly as given to match the condition.
* *timeZone*: String The timeZone must be exactly as given to match the condition.

All conditions in the FilterCondition object must match for the Principal to match.

2.5. Principal/queryChanges

This is a standard "/queryChanges" method as described in [RFC8620], Section 5.6.

3. Share Notifications

The ShareNotification data type records when the user’s permissions to access a shared object changes. ShareNotification are only created by the server; users cannot create them explicitly. Notifications are stored in the same Account as the Principals.

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 ShareNotification whenever the user’s permissions change on an object. It SHOULD NOT create a notification for permission changes to a group principal, even if the user is in the group.

3.1. 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 notifications if appropriate, or remove notifications that it deems are no longer relevant or after a certain period of time. The server SHOULD automatically destroy a notification about an object if the user subscribes to that object.

3.2. Object Properties

The *ShareNotification* object has the following properties:

- *id*: String The id of the ShareNotification.
- *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|null The email of the person who made the change, or null if no email is available.
  - *principalId*: String|null The id of the Principal corresponding to the person who made the change, or null if no associated principal.
- *objectType*: String The name of the data type for the object whose permissions have changed, e.g. "Calendar" or "Mailbox".
- *objectAccountId*: String The id of the account where this object exists.
- *objectId*: String The id of the object that this notification is about.
- *name*: String The name of the object at the time the notification was made.
- *oldRights*: String[Boolean]|null The "myRights" property of the object for the user before the change.
- *newRights*: String[Boolean]|null The "myRights" property of the object for the user after the change.

3.3. ShareNotification/get

This is a standard "/get" method as described in [RFC8620], Section 5.1.

3.4. ShareNotification/changes

This is a standard "/changes" method as described in [RFC8620], Section 5.2.

3.5. ShareNotification/set

This is a standard "/set" method as described in [RFC8620], Section 5.3.

Only destroy is supported; any attempt to create/update MUST be rejected with a forbidden SetError.
3.6. ShareNotification/query

This is a standard "/query" method as described in [RFC8620], Section 5.5.

3.6.1. 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.
* *objectType*: String The objectType value must be identical to the given value to match the condition.
* *objectAccountId*: String The objectAccountId value must be identical to the given value to match the condition.

3.6.2. Sorting

The "created" property MUST be supported for sorting.

3.7. ShareNotification/queryChanges

This is a standard "/queryChanges" method as described in [RFC8620], Section 5.6.

3.8. Framework for shared data

Shareable data types SHOULD define the following three properties:

* *isSubscribed*: Boolean Has the user indicated they wish to see this data? The initial value for this when data is shared by another user is implementation dependent, although data types may give advice on appropriate defaults.
* *myRights*: String[Boolean] The set of permissions the user currently has. Appropriate permissions are domain specific and must be defined per data type.
* *shareWith*: Id[String[Boolean]] A map of principal id to rights to give that principal. The account id for the principal id can be found in the capabilities of the Account this object is in (see Section XXX). Users with appropriate permission may set this property to modify who the data is shared with. The principal that owns the account this data is in MUST NOT be in the set of sharees; their rights are implicit.
4. Security Considerations

All security considerations of JMAP [RFC8620] apply to this specification. Additional considerations are detailed below.

4.1. Spoofing

Allowing users to edit their own Principal’s name (and, to a lesser extent, description) could allow a user to change their name to that of another user in the system, potentially tricking others into sharing private data with them. Servers may choose to forbid this, and SHOULD keep logs of such changes to provide an audit trail.

4.2. Unnoticed sharing

Sharing data with another user allows someone to turn a transitory account compromise (e.g. brief access to an unlocked, logged in terminal) into a persistant compromise (by setting up sharing with a user controlled by the attacker). This can be mitigated by requiring further authorisation for configuring sharing, or sending notifications to the sharer via another channel whenever a new sharee is added.

4.3. Unauthorised principals

The set of principals within a shared environment SHOULD be strictly controlled. If adding a new principal is open to the public, risks include: * An increased risk of a user accidentally sharing data with an unintended person. * An attacker may share unwanted or offensive information with the user. * An attacker may share items with spam content in the names in order to generate ShareNotification objects, which are likely to be prominently displayed to the sharee.

5. IANA Considerations

5.1. JMAP Capability Registration for "principals"

IANA will register the "principals" JMAP Capability as follows:

Capability Name: urn:ietf:params:jmap:principals

Specification document: this document

Intended use: common

Change Controller: IETF

Security and privacy considerations: this document, Section XXX
5.2. JMAP Capability Registration for "principals:owner"

IANA will register the "principals:owner" JMAP Capability as follows:

Capability Name: urn:ietf:params:jmap:principals:owner

Specification document: this document

Intended use: common

Change Controller: IETF

Security and privacy considerations: this document, Section XXX

6. Normative References


Author’s Address

Neil Jenkins (editor)
Fastmail
PO Box 234, Collins St West
Melbourne  VIC 8007
Australia

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

This document specifies an extension to JMAP for Mail (RFC 8621) 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 10 July 2022.

Copyright Notice

Copyright (c) 2022 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 Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.
1. Introduction

JMAP for Mail [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 the multipart/signed media type [RFC1847] is used) or to download and decode CMS (when the application/pkcs7-mime media type (Section 3.2 of [RFC8551]) is used). The use of the extension implies the client trusts the JMAP server’s S/MIME signature verification code and configuration. This extension is suitable for cases where reduction in network bandwidth and client-side code complexity outweigh security concerns about trusting the JMAP server to perform S/MIME signature verifications. One possible use case is when the same organization controls both the JMAP server and the JMAP client.

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.
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.

3. Addition to the capabilities object

The capabilities object is returned as part of the standard JMAP Session object; see Section 2 of [RFC8620]. Servers supporting _this_ specification MUST add a property called "urn:ietf:params:jmap:smimeverify" 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 for S/MIME signature verification

4.1. Extension to Email/get

[RFC8621] defines the Email/get method for retrieving message specific information. This document defines the following pseudo values in the _properties_ argument:

* "smimeStatus": If "smimeStatus" is included in the list of requested properties, it MUST be interpreted by the server as a request to return the "smimeStatus" response property.

* "smimeStatusAtDelivery": If "smimeStatusAtDelivery" is included in the list of requested properties, it MUST be interpreted by the server as a request to return the "smimeStatusAtDelivery" response property. (It is effectively the same as the "smimeStatus" value calculated at the date/time of delivery, as specified by "receivedAt".)

* "smimeErrors": If "smimeErrors" is included in the list of requested properties, it MUST be interpreted by the server as a request to return the "smimeErrors" response property.

* "smimeVerifiedAt": If "smimeVerifiedAt" is included in the list of requested properties, it MUST be interpreted by the server as a request to return the "smimeVerifiedAt" response property.

The "smimeStatus" response property is defined as follows:

smimeStatus: "String|null" (server-set). null signifies that the message doesn’t contain any signature. Otherwise, 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, as defined in extensions to this document. Clients 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 was neither signed nor encrypted. This can also be returned for a multipart/signed message which contains an 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 a signature until a particular message is requested by the client. (This is a useful optimization for a JMAP server to avoid doing work until exact information is needed. A JMAP client that only needs to display an icon that signifies presence of an S/MIME signature can still use this value.) JMAP servers compliant with this document SHOULD attempt signature verification and 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 according to [RFC8551] and [RFC8550]. Additionally the signer email address extracted from the S/MIME certificate matches the From header field value, and the signer certificate SHOULD be checked for revocation.

signed/failed: S/MIME signed message, but the signature failed to verify according to [RFC8551] and [RFC8550]. This might be a policy related decision (e.g. the message signer email address doesn’t match the From header field value), message was modified, the signer’s certificate has expired or was revoked, etc.

encrypted+signed/verified: This value is reserved for future use. It is typically handled in the same way as "signed/verified".

encrypted+signed/failed: This value is reserved for future use. It is typically handled in the same way as "signed/failed".

The "smimeStatusAtDelivery" response property has the same syntax as "smimeStatus" but is calculated in relationship to the "receivedAt" date/time. Unlike "smimeStatus", the "smimeStatusAtDelivery" response property value doesn’t change, unless Trust Anchors are added. (For example, addition of a Trust Anchor can change the value of a message "smimeStatusAtDelivery" property from "signed/failed" to "signed/verified". Note that Trust Anchor removal doesn’t affect this response property.) The "smimeStatusAtDelivery" allows clients to compare the S/MIME signature verification status at delivery with
the current status as returned by "smimeStatus", for example to help to answer questions like "was the signature valid at the time of delivery?".

Note that the "smimeStatusAtDelivery" response property value doesn't have to be calculated at delivery time. A JMAP server can defer its calculation until it is explicitly requested, but once calculated its value is remembered for later use.

The "smimeErrors" response property is defined as follows:

smimeErrors: "String[]|null" (server-set). null signifies that the message doesn't contain any signature or that there were no errors when verifying the S/MIME signature. (I.e., this property is non null only when the corresponding "smimeStatus" response property value is "signed/failed" or "encrypted+signed/failed". Note that future extensions to this document can specify other smimeStatus values that can be used with smimeErrors.) Each string in the array is a human readable description (in the language specified in the Content-Language header field, if any) of a problem with the signature, the signing certificate or the signing certificate chain. (See Section 3.8 of [RFC8620] in regards to how this is affected by the language selection.) In one 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. In another example the certificate might be expired and the JMAP server might be unable to retrieve a 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. (Retrieval of the smimeStatus value can be used to distinguish these 2 cases). In all other cases it is set to the date and time of when the S/MIME signature was most recently verified. Note that a request to fetch "smimeStatus", "smimeStatusAtDelivery" and/or "smimeErrors" would force this response property to be set to a non null value, if an S/ MIME signature exists.
"smimeStatus" and "smimeErrors" values are calculated at the time the corresponding JMAP request was processed (but see below about the effect of result caching), not at the time when the message was generated (according to its Date header field value). In all cases "smimeVerifiedAt" is set to the time when "smimeStatus" and "smimeErrors" were last updated. As recalculating these values is expensive for the server, they MAY be cached for up to 24 hours from the moment when they were calculated.

Example 1: Retrieval of minimal information about a message, including its From, Subject and Date header fields, as well as S/MIME signature verification status at delivery and date/time when the message was received.

```json
["Email/get", {
  "ids": [ "fe123u457" ],
  "properties": [ "mailboxIds", "from", "subject", "date", "smimeStatusAtDelivery", "receivedAt" ]
}, "#1"]
```

This might result in the following response:

```json
[["Email/get", {
  "accountId": "abc",
  "state": "51234123231",
  "list": [
    {
      "id": "fe123u457",
      "mailboxIds": { "f123": true },
      "from": [{"name": "Joe Bloggs", "email": "joe@bloggs.example.net"}],
      "subject": "Dinner tonight?",
      "date": "2020-07-07T14:12:00Z",
      "smimeStatusAtDelivery": "signed/verified",
      "receivedAt": "2020-07-07T14:15:18Z"
    }
  ],
}, "#1"]
```

Example 2: Retrieval of minimal information about a message, including its From, Subject and Date header fields, as well as the latest S/MIME signature verification status, S/MIME verification errors (if any) and when was the S/MIME signature status last verified. The response contains 2 S/MIME errors related to S/MIME signature verification.
["Email/get", {
    "ids": [ "ag123u123" ],
    "properties": [ "mailboxIds", "from", "subject", "date",
                    "smimeStatus", "smimeErrors", "smimeVerifiedAt" ]
}, "#1"]

This might result in the following response:

[{{"Email/get", {
    "accountId": "abc",
    "state": "47234123231",
    "list": [
        {
            "id": "ag123u123",
            "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"]

4.1.1. "smimeStatus" response property extensibility

Future extensions to this document can specify extra allowed values for the smimeStatus response property. All values (defined in this document or in extensions to this document) MUST be in ASCII. (Note that this response property contains tokens, thus it is not subject to Internationalization or Localization).

New smimeStatus response property values defined in extensions may affect behaviour of properties such as smimeErrors response property of Email/get (see Section 4.1) or hasVerifiedSmime property of Email/query (see Section 4.2). In particular the new values can be treated similar to values defined in this document.

For example a putative JMAP extension for automatically decrypting S/MIME messages can specify two additional values, one specifying that a message is both encrypted and signed with a valid S/MIME signature and another one specifying that a message is both encrypted and signed with an invalid S/MIME signature. The former value can be
treated as "signed/verified" (and would thus affect hasVerifiedSmime) and the latter can be treated as "signed/failed" (and thus can be used with smimeErrors).

4.2. Extension to Email/query

[RFC8621] defines the Email/query method for searching for messages with specific properties. This document defines the following properties of the *FilterCondition* object:

- **hasSmime**: "Boolean". If "hasSmime" has the value true, only messages with "smimeStatus" other than null match the condition. If "hasSmime" has the value false, only messages with "smimeStatus" equal to null match the condition.

- **hasVerifiedSmime**: "Boolean". If "hasVerifiedSmime" has the value true, only messages with "smimeStatus" equal to "signed/verified" or "encrypted+signed/verified" (*), match the condition. If "hasVerifiedSmime" has the value false, only messages with "smimeStatus" not equal to "signed/verified" and not equal to "encrypted+signed/verified" (*) (including the value null) match the condition. Note that use of this attribute is potentially expensive for a JMAP server, as it forces calculation of smimeStatus property value for each message. However caching of smimeStatus values should ameliorate this cost somewhat. (*), as well as "smimeStatus" values added by future extensions to this document that are explicitly specified as having similar effect to "signed/verified" as far as "hasVerifiedSmime" calculation is concerned.

- **hasVerifiedSmimeAtDelivery**: "Boolean". The "hasVerifiedSmimeAtDelivery" property is handled similar to "hasVerifiedSmime" property, but the value of "smimeStatusAtDelivery" is used instead of "smimeStatus" to assess whether a particular message matches the condition.

4.3. Interaction with Email/changes

Changes to "smimeVerifiedAt" response property value MUST NOT cause the message to be included in the "updated" argument of Email/changes response. However changes to "smimeStatus", "smimeStatusAtDelivery" and/or "smimeErrors" response properties MUST result in message inclusion in the "updated" argument of Email/changes response.
5. IANA Considerations

5.1. JMAP capability registration for "smimeverify"

IANA is requested to register the "smimeverify" JMAP Capability as follows:

Capability Name: "urn:ietf:params:jmap:smimeverify"

Specification document: this document

Intended use: common

Change Controller: IETF

Security and privacy considerations: this document, Section 6

6. Security Considerations

Use of the server-side S/MIME signature verification JMAP extension requires the client to trust the server signature verification code, server configuration and its operational practices to perform S/MIME signature verification, as well as to trust that the channel between the client and the server is integrity protected. (For example, if the server is not configured with some Trust Anchors, some messages will have "signed/failed" status instead of "signed/verified".) A malicious or compromised server could return false verification status to a client. A successful verification could be conveyed to a client for a forged or altered message. A properly signed message could be signaled as having a failed signature verification or no signature at all. In the case of the latter attack, no new attack surface is presented with this extension above what malicious or compromised server could already do by stripping or tampering with the S/MIME information in the message. In the case of the former attack, client software capable of performing S/MIME signature verification could detect this attack. Local configuration of the client should determine if this client-side verification should occur. For clients without local verification capabilities, such an attack would be difficult to detect.

Integrity protection of the channel between the client and the server is provided by use of TLS, as required by JMAP specification (see Section 8.1 of [RFC8620]).

Constant recalculation of S/MIME signature status can result in a Denial-of-Service condition. For that reason, it is RECOMMENDED that servers cache results of signature verification for up to 24 hours.
7. References

7.1. Normative References


7.2. Informative References


Appendix A. Acknowledgements

This document is a product of the JMAP Working Group. Special thank you to Bron Gondwana, Neil Jenkins, Murray Kucherawy, Kirsty Paine, Benjamin Kaduk, Roman Danyliw, Peter Yee, Robert Wilton, Erik Kline and Menachem Dodge for suggestions, comments and corrections to this document.

Author’s Address
Abstract

This document specifies an extension to JMAP for sending S/MIME signed and S/MIME encrypted messages.

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 12 January 2023.

Copyright Notice

Copyright (c) 2022 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 Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.
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 sending S/MIME signed and encrypted messages. It allows JMAP server to sign/encrypt messages on user’s behalf.

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-advanced" 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/set for S/MIME signing and/or encryption

[RFC8621] defines Email/set method for creating new email messages. This document defines the following additional request arguments that can be used to create S/MIME signed and/or encrypted messages:
* smimeSign: "Boolean" (default: false). If included and has the value "true", this requests the JMAP server to create an S/MIME signed message from the message constructed according to other specified arguments (the "original message"). This is done by encapsulating the original message either inside application/pkcs7-mime [RFC8551] or multipart/signed [RFC1847] container. (smimeSignOpaque argument (see below) controls which of the two mechanisms is used.) The signature's private key/certificate is associated with the email address in the Sender header field, if present; otherwise, it is associated with the email address in the From header field, if present.

If multiple addresses are present in one of these header fields, or there is more than one Sender/From header field, the server SHOULD reject the Email/set as invalid with the "invalidEmail" error code; otherwise, it MUST take the first address in the last Sender/From header field. If JMAP account is not authorized to sign message as the selected sender (as above), it SHOULD return "signedSenderNotAllowed" error code.

* smimeEncrypt: "Boolean" (default: false). If included and has the value "true", this requests the JMAP server to create an S/MIME encrypted message from the constructed message. This is done by encapsulating the message inside application/pkcs7-mime [RFC8551] media type. The message MUST be encrypted to the sender and all To/Cc/Bcc recipients. This extension assumes that there is some kind of per user or organizational addressbook, that can be used to lookup public keys of recipients. If lookup of a particular public key fails, or results in an expired or revoked certificate, the Email/set operation MUST fail with the "validEncryptionKeyNotFound" error code.

* headerProtect: "Boolean" (default: true). If has the value "true", this requests the JMAP server to use S/MIME header protection as specified in draft-ietf-lamps-header-protection when at least one of smimeEncrypt/smimeSign is true.

* smimeSignOpaque: "Boolean" (default: true). If has the value "true", this requests the JMAP server to use application/pkcs7-mime media type for S/MIME signing, otherwise multipart/signed media type.

If both "smimeSign" and "smimeEncrypt" are set to true, the message is first signed and then the signed version is encrypted (in that order).
(Note that this extension doesn’t allow management of private keys/certificates. How private keys are managed or configured for a particular user is out of scope for this document.)

```json
[[ "Email/set", {
  "accountId": "ue150411c",
  "create": {
    "k192": {
      "mailboxIds": {
        "2ealca41b38e": true
      },
      "keywords": {
        "$seen": true,
        "$draft": true
      },
      "from": [{
        "name": "Joe Bloggs",
        "email": "joe@example.com"
      }],
      "subject": "World domination",
      "receivedAt": "2021-07-07T01:03:11Z",
      "sentAt": "2021-07-10T11:03:11+10:00",
      "smimeSign": true,
      "smimeEncrypt": true,
      "bodyStructure": {
        "type": "text/plain",
        "partId": "bd48",
        "header:Content-Language": "en"
      },
      "bodyValues": {
        "bd48": {
          "value": "I have the most brilliant plan. Let me tell you all about it."
        }"
      }
    }
  }
}, "0" ]]
```

This will result in the following response:

```json
[[ "Email/set", {
  "accountId": "ue150411c",
  "oldState": "780823",
  "newState": "780839",
}, "0" ]]
```
"created": {
  "k192": {
    "id": "Mf40b5f831efa7233b9eb1c7f",
    "blobId": "Gf40b5f831efa7233b9eb1c7f8f97d84eeeee64f7",
    "threadId": "Td957e72e89f516dc",
    "size": 5096
  }
}, ...
}, "0"]

Figure 1: Example 1:

5. IANA Considerations

5.1. JMAP capability registration for "smime-advanced"

IANA is requested to register the "smime" JMAP Capability as follows:

Capability Name: "urn:ietf:params:jmap:smime-advanced"

Specification document: this document

Intended use: common

Change Controller: IETF

Security and privacy considerations: this document, Section 6

5.2. JMAP Error Codes Registry Updates

5.2.1. signedSenderNotAllowed error code

JMAP Error Code: signedSenderNotAllowed

Intended use: common

Change controller: IETF

Reference: This document, Section 4

Description: JMAP account is not authorized to S/MIME sign message as the specified sender.
5.2.2. validEncryptionKeyNotFound error code

JMAP Error Code: validEncryptionKeyNotFound

Intended use: common

Change controller: IETF

Reference: This document, Section 4

Description: S/MIME encrypted message can’t be generated because no valid certificate (non expired and non revoked) can be found for one of recipients.

6. Security Considerations

This JMAP extension assumes trust between the user and the JMAP server for purposes of signing and encrypting messages on user’s behalf.

This JMAP extension also relies on access to user’s (or organization’s) addressbook which contain up-to-date certificates for recipients.

This JMAP extension doesn’t support management of user’s private keys and corresponding certificates.

7. Normative References


Author’s Address

Alexey Melnikov
Isode Ltd
14 Castle Mews
Hampton
TW12 2NP
United Kingdom
Email: Alexey.Melnikov@isode.com
Sieve Email Filtering: Extension for Processing iMIP Messages

draft-murchison-sieve-processimip-00

Abstract

This document describes the "processimip" extension to the Sieve email filtering language. The "processimip" extension gives Sieve the ability to process messages using the iCalendar Message-Based Interoperability Protocol (iMIP).

Open Issues

1. The Cyrus implementation used at Fastmail also adds an :invitesonly option to the processimip action in order to emulate existing functionality elsewhere within our stack. Is there any interest in formalizing this option? This may be superfluous as it might not make sense to auto-process an initial invitation but then NOT auto-process future updates to an event.

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 28 August 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.
1. Introduction

Users typically receive invites, replies, and cancelations for events, tasks, etc. via Internet mail messages. It is sometimes desirable to have such messages automatically parsed and the attached iCalendar [RFC5545] objects added to, updated on, or deleted from the user’s calendars.

This document defines an extension to the Sieve language [RFC5228] that enables scripts to process messages using the iCalendar Message-Based Interoperability Protocol (iMIP) [RFC6047]. Specifically, this extension provides the ability to alter iCalendar objects on a user’s calendars referenced in iMIP messages.
2. Conventions Used in This Document

Conventions for notations are as in Section 1.1 of [RFC5228], including use of the "Usage:" label for the definition of action and tagged arguments syntax.

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. Capability Identifier

Sieve interpreters that implement this extension have an identifier of "processimip" for use with the capability mechanism.

4. Process iMIP Action

Usage: processimip [:addresses <string-list> ][ :updatesonly | :calendarid <string> ][ :deletecanceled ][ :outcome <variablename: string> ][ :errstr <variablename: string> ]

The "processimip" action can be used with or without the "variables" [RFC5229] extension. When the "variables" extension is enabled in a script using <require "variables">, the script can use the ":outcome" (Section 4.5) and ":errstr" (Section 4.6) arguments to the "processimip" action described below. When the "variables" extension is not enabled, the ":outcome" and ":errstr" arguments MUST NOT be used and MUST cause an error according to [RFC5228].

"processimip" MUST NOT process a message unless it is a well-formed iMIP message and one of the recipient user’s email addresses matches the Calendar User Address (see Section 3.3.3 of [RFC5545]) of the intended target of the message, as determined by the iTIP method (see Section 1.4 of [RFC5546]) of the message:

"REPLY": Value of the "Organizer" property (see Section 3.8.4.1 of [RFC5545])

"REQUEST", "CANCEL", "ADD": Value of one of the "Attendee" properties (see Section 3.8.4.3 of [RFC5545])

The recipient user’s email address matches the Calendar User Address of the target if the Calendar User Address is in the form of a mailto URI and the email address matches the "addr-spec" of the URI.
An email address is considered to belong to the recipient if it is one of:

1. an email address known by the implementation to be associated with the recipient,
2. the final envelope recipient address if it’s available to the implementation, or
3. an address specified by the script writer via the :addresses (Section 4.1) argument.

The "processimip" action does not cancel the implicit keep.

4.1. Addresses Argument

The optional :addresses argument is used to specify email addresses that belong to the recipient in addition to the addresses known to the implementation.

4.2. Updates Only Argument

The optional :updatesonly argument is used to limit the messages processed to those targeting existing iCalendar objects only. If the message contains a new iCalendar object (initial invitation), the implementation MUST NOT add the object to a calendar.

If :updatesonly is omitted, new iCalendar objects (initial invitations) may be added to one of the user’s calendars.

4.3. Calendar ID Argument

The optional :calendarid argument specifies the identifier of the calendar onto which new iCalendar objects (initial invitations) should placed.

If :calendarid is omitted, new iCalendar objects will be placed on the user’s "default" calendar as determined by the implementation.

4.4. Delete Canceled Argument

The optional :deletecanceled argument is used to tell the implementation that if it receives a cancelation message, it should remove the associated iCalendar object from the calendar.

If :deletecanceled is omitted, the associated iCalendar object will be marked as canceled and will remain on the calendar.
4.5. Outcome Argument

The optional :outcome argument specifies the name of a variable into which one of the following strings specifying the outcome of the action will be stored:

* "no_action": No action was performed (E.g., the message wasn’t an iMIP message, or the message contained a new iCalendar object but the ":updatesonly" argument was used)
* "added": A new iCalendar object was added to a calendar
* "update": An iCalendar resource was updated or canceled
* "error": An error processing the iMIP message occurred

4.6. Error String Argument

The optional :errstr argument specifies the name of a variable into which a string describing the reason for the outcome will be stored.

4.7. Examples

TODO: Actually add examples.

5. Implementation Status

< RFC Editor: before publication please remove this section and the reference to [RFC7942] >

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".

5.1. Cyrus Server

The open source Cyrus Server (http://www.cyrusimap.org/) project is a highly scalable enterprise mail system which supports Sieve email filtering at the point of final delivery. This production level Sieve implementation supports all of the requirements described in this document. This implementation is freely distributable under a BSD style license from Computing Services at Carnegie Mellon University (http://www.cmu.edu/computing/).

6. Security Considerations

Security considerations are discussed in [RFC5228].

TODO: Discuss calendar SPAM.

7. Privacy Considerations

It is believed that this extension doesn’t introduce any privacy considerations beyond those in [RFC5228].

8. IANA Considerations

8.1. Registration of Sieve Extension

This document defines the following new Sieve extension to be added to the registry defined in Section 6.2 of [RFC5228] and located here: https://www.iana.org/assignments/sieve-extensions/sieve-extensions.xhtml#sieve-extensions

IANA are requested to add a capability to the Sieve Extensions registry:

To: iana@iana.org

Subject: Registration of new Sieve extension

Capability name: processimip

Description: Adds the "processimip" action command to add and update iCalendar objects on a user’s calendars.
9. Acknowledgments

The authors would like to thank the following individuals for contributing their ideas and support for writing this specification: Ned Freed and Alexey Melnikov.

10. References

10.1. Normative References


10.2. Informative References


Authors’ Addresses

Kenneth Murchison
Fastmail US LLC
1429 Walnut Street - Suite 1201
Philadelphia, PA 19102
United States of America
Email: murch@fastmailteam.com

Ricardo Signes
Fastmail US LLC
1429 Walnut Street - Suite 1201
Philadelphia, PA 19102
United States of America
Email: rjbs@fastmailteam.com

Matthew Horsfall
Fastmail US LLC
1429 Walnut Street - Suite 1201
Philadelphia, PA 19102
United States of America
Email: alh@fastmailteam.com