

JMAP
Internet-Draft
Intended status: Standards Track
Expires: 2 August 2021

J.M. Baum, Ed.
H.J. Happel, Ed.
audriga
29 January 2021

JMAP for Tasks
draft-baum-jmap-tasks-00

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
---------------------------	---

1.1.	Notational Conventions	3
1.2.	Terminology	3
1.3.	Data Model Overview	4
1.4.	Addition to the Capabilities Object	4
1.4.1.	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:

- * ***shareesActAs***: "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 ≥ 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-jscalendar], 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 $0 \leq \text{sortOrder} < 2^{(31)}$.

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 themselves.

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-jmap-calendars], 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-calex-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.
- * `*taskIds*`: `"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]

Jenkins, N. and R. Stepanek, "JSCalendar: A JSON representation of calendar data", Work in Progress, Internet-Draft, draft-ietf-calext-jscalendar-32, 15 October 2020, <<https://tools.ietf.org/html/draft-ietf-calext-jscalendar-32>>.

[I-D.ietf-jmap-calendars]

Jenkins, N. and M. Douglass, "JMAP for Calendars", Work in Progress, Internet-Draft, draft-ietf-jmap-calendars-05, 24 January 2021, <<https://tools.ietf.org/html/draft-ietf-jmap-calendars-05>>.

[I-D.jenkins-jmap-sharing]

Jenkins, N., "JMAP Sharing", Work in Progress, Internet-Draft, draft-jenkins-jmap-sharing-00, 15 December 2020, <<https://tools.ietf.org/html/draft-jenkins-jmap-sharing-00>>.

[RFC2119]

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

- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8620] Jenkins, N. and C. Newman, "The JSON Meta Application Protocol (JMAP)", RFC 8620, DOI 10.17487/RFC8620, July 2019, <<https://www.rfc-editor.org/info/rfc8620>>.

10. Informative References

- [RFC4791] Daboo, C., Desruisseaux, B., and L. Dusseault, "Calendaring Extensions to WebDAV (CalDAV)", RFC 4791, DOI 10.17487/RFC4791, March 2007, <<https://www.rfc-editor.org/info/rfc4791>>.

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>

JMAP
Internet-Draft
Intended status: Standards Track
Expires: 28 August 2022

N.M. Jenkins, Ed.
Fastmail
M. Douglass, Ed.
Spherical Cow Group
24 February 2022

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

6.3.	Snoozing an alert	35
6.4.	Push events	36
7.	Calendar Event Notifications	36
7.1.	Auto-deletion of Notifications	37
7.2.	Object Properties	37
7.3.	CalendarEventNotification/get	38
7.4.	CalendarEventNotification/changes	38
7.5.	CalendarEventNotification/set	38
7.6.	CalendarEventNotification/query	38
7.6.1.	Filtering	38
7.6.2.	Sorting	39
7.7.	CalendarEventNotification/queryChanges	39
8.	CalendarPreferences	39
8.1.	CalendarPreferences/get	39
8.2.	CalendarPreferences/set	40
9.	Security Considerations	40
9.1.	Privacy	40
9.2.	Spoofing	40
9.3.	Denial-of-service	41
9.3.1.	Expanding Recurrences	41
9.3.2.	Firing alerts	41
9.3.3.	Load spikes	41
9.4.	Spam	42
10.	IANA Considerations	42
10.1.	JMAP Capability Registration for "calendars"	42
10.2.	JMAP Capability Registration for "calendars:preferences"	42
10.3.	JMAP Capability Registration for "principals:availability"	43
10.4.	JSCalendar Property Registrations	43
10.4.1.	id	43
10.4.2.	calendarIds	43
10.4.3.	isDraft	43
10.4.4.	utcStart	44
10.4.5.	utcEnd	44
10.4.6.	mayInviteSelf	44
10.4.7.	mayInviteOthers	44
10.4.8.	hideAttendees	45
11.	Normative References	45
12.	Informative References	45
	Authors' Addresses	46

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 "includeInAvailability" property of the calendar is "attending", then the principal is a participant of the event, and has a "participationStatus" of "accepted" or "tentative".

If an event is in more than one calendar, it is relevant if all of the above are true for any one calendar that it is in.

The server then generates a BusyPeriod object for each of these events. A ***BusyPeriod*** object has the following properties:

- * ***utcStart***: UTCDate The start time (inclusive) of the period this represents.
- * ***utcEnd***: UTCDate The end time (exclusive) of the period this represents.
- * ***busyStatus***: String (optional, default "unavailable") This MUST be one of
 - confirmed: The event status is "confirmed".
 - tentative: The event status is "tentative".
 - unavailable: The principal is not available for scheduling at this time for any other reason.
- * ***event***: JSEvent|null The JSEvent representation of the event, or null if any of the following are true:

- The "showDetails" argument is false.
- The "privacy" property of the event is "private".
- The user does not have the "mayReadItems" permission for any of the calendars the event is in.

If an eventProperties argument was given, any properties in the JSEvent that are not in the eventProperties list are removed from the returned representation.

The server MAY also generate BusyPeriod objects based on other information it has about the principal's availability, such as office hours.

Finally, the server MUST merge and split BusyPeriod objects where the "event" property is null, such that none of them overlap and either there is a gap in time between any two objects (the utcEnd of one does not equal the utcStart of another) or those objects have a different busyStatus property. If there are overlapping BusyPeriod time ranges with different "busyStatus" properties the server MUST choose the value in the following order: confirmed > unavailable > tentative.

The response has the following argument:

- * *list*: BusyPeriod[] The list of BusyPeriod objects calculated as described above.

The following additional errors may be returned instead of the "Principal/getAvailability" response:

notFound: No principal with this id exists, or the user does not have permission to see that this principal exists.

forbidden: The user does not have permission to query this principal's availability.

tooLarge: The duration between utcStart and 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 $0 \leq \text{sortOrder} < 2^{31}$.

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 themselves.

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 `calendarHasEventSetError`. 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 "participationStatus" 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": {
    "dG9tQGZvb2Jhci5x1LmNvbQ": {
      "@type": "Participant",
      "name": "Tom",
      "email": "tom@foobar.example.com",
      "sendTo": {
        "imip": "mailto:6489-4f14-a57f-c1@calendar.example.com"
      },
      "participationStatus": "accepted",
      "roles": {
        "attendee": true
      }
    },
    "em9lQGZvb2GFtcGx1LmNvbQ": {
      "@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/dG9tQGZvb2Jhci5x1LmNvbQ/participationStatus":
        "declined"
    }
  }
}
```

In this example, Tom is normally going to the weekly meeting but has declined the occurrence on 2018-03-08, which starts an hour later than normal. Now, if Zoe too were to decline that meeting, she could update the event by just sending a patch like so:

```
[[ "CalendarEvent/set", {
  "accountId": "ue150411c",
  "update": {
    "123": {
      "recurrenceOverrides/2018-03-08T09:00:00/
        participants~1em9lQGZvb2GFtcGx1LmNvbQ~1participationStatus":
          "declined"
    }
  }
}, "0" ]]
```

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

```
"recurrenceOverrides": {
  "2018-03-08T09:00:00": {
    "start": "2018-03-08T10:00:00",
    "participants/dG9tQGZvb2Jhci5x1LmNvbQ/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:

```
[[ "CalendarEvent/set", {
  "accountId": "ue150411c",
  "update": {
    "123": {
      "recurrenceOverrides/2018-03-08T09:00:00/
        participants~1dG9tQGZvb2Jhci5x1LmNvbQ~1participationStatus": null
    }
  }
}, "0" ]]
```

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

```
[[ "CalendarEvent/set", {
  "accountId": "ue150411c",
  "update": {
    "123": {
      "recurrenceOverrides/2018-03-08T09:00:00/
        participants~1dG9tQGZvb2Jhci5x1LmNvbQ": null
    }
  }
}, "0" ]]
```

This would mean remove the "participants/dG9tQGZvb2Jhci5x1LmNvbQ" property at path "recurrenceOverrides" -> "2018-03-08T09:00:00" inside the object; but this doesn't exist. We actually we want to add this property and make it map to null. The client must instead send the full object that contains the property mapping to null, like so:

```
[[ "CalendarEvent/set", {
  "accountId": "ue150411c",
  "update": {
    "123": {
      "recurrenceOverrides/2018-03-08T09:00:00": {
        "start": "2018-03-08T10:00:00",
        "participants/em9lQGZvb2GFtcGx1LmNvbQ/participationStatus":
          "declined"
      }
      "participants/dG9tQGZvb2Jhci5x1LmNvbQ": 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 ParticipantIdentities 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`.

Severs **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 pushed 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

Security and privacy considerations: this document, Section XXX

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

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC2397] Masinter, L., "The "data" URL scheme", RFC 2397, DOI 10.17487/RFC2397, August 1998, <<https://www.rfc-editor.org/info/rfc2397>>.
- [RFC6376] Crocker, D., Ed., Hansen, T., Ed., and M. Kucherawy, Ed., "DomainKeys Identified Mail (DKIM) Signatures", STD 76, RFC 6376, DOI 10.17487/RFC6376, September 2011, <<https://www.rfc-editor.org/info/rfc6376>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8551] Schaad, J., Ramsdell, B., and S. Turner, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 4.0 Message Specification", RFC 8551, DOI 10.17487/RFC8551, April 2019, <<https://www.rfc-editor.org/info/rfc8551>>.
- [RFC8620] Jenkins, N. and C. Newman, "The JSON Meta Application Protocol (JMAP)", RFC 8620, DOI 10.17487/RFC8620, July 2019, <<https://www.rfc-editor.org/info/rfc8620>>.
- [RFC8984] Jenkins, N. and R. Stepanek, "JSCalendar: A JSON Representation of Calendar Data", RFC 8984, DOI 10.17487/RFC8984, July 2021, <<https://www.rfc-editor.org/info/rfc8984>>.

12. Informative References

- [RFC4791] Daboo, C., Desruisseaux, B., and L. Dusseault,
"Calendaring Extensions to WebDAV (CalDAV)", RFC 4791,
DOI 10.17487/RFC4791, March 2007,
<<https://www.rfc-editor.org/info/rfc4791>>.
- [RFC5546] Daboo, C., Ed., "iCalendar Transport-Independent
Interoperability Protocol (iTIP)", RFC 5546,
DOI 10.17487/RFC5546, December 2009,
<<https://www.rfc-editor.org/info/rfc5546>>.
- [RFC6047] Melnikov, A., Ed., "iCalendar Message-Based
Interoperability Protocol (iMIP)", RFC 6047,
DOI 10.17487/RFC6047, December 2010,
<<https://www.rfc-editor.org/info/rfc6047>>.

Authors' Addresses

Neil Jenkins (editor)
Fastmail
PO Box 234, Collins St West
Melbourne VIC 8007
Australia
Email: neilj@fastmailteam.com
URI: <https://www.fastmail.com>

Michael Douglass (editor)
Spherical Cow Group
226 3rd Street
Troy, NY 12180
United States of America
Email: mdouglass@sphericalcowgroup.com
URI: <http://sphericalcowgroup.com>

JMAP
Internet-Draft
Intended status: Standards Track
Expires: 17 July 2022

R. Stepanek
FastMail
M. Loffredo
IIT-CNR
13 January 2022

JSContact: A JSON representation of contact data
draft-ietf-jmap-jscontact-10

Abstract

This specification defines a data model and JSON representation of contact card information that can be used for data storage and exchange in address book or directory applications. It aims to be an alternative to the vCard data format and to be unambiguous, extendable and simple to process. In contrast to the JSON-based jCard format, it is not a direct mapping from the vCard data model and expands semantics where appropriate.

Status of This Memo

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

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

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

This Internet-Draft will expire on 17 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.

Table of Contents

1. Introduction	3
1.1. Relation to the xCard and jCard formats	4
1.2. Terminology	4
1.3. Vendor-specific Property Extensions and Values	4
1.4. Type Signatures	5
1.5. Data types	5
1.5.1. Context	5
1.5.2. Id	6
1.5.3. PatchObject	6
1.5.4. Preference	7
1.5.5. UnsignedInt	7
1.5.6. UTCDateTime	8
2. Card	8
2.1. Metadata properties	8
2.1.1. @type	8
2.1.2. uid	8
2.1.3. prodId	8
2.1.4. created	8
2.1.5. updated	9
2.1.6. kind	9
2.1.7. relatedTo	9
2.1.8. language	10
2.2. Name and Organization properties	10
2.2.1. name	10
2.2.2. fullName	11
2.2.3. nickNames	11
2.2.4. organizations	12
2.2.5. titles	12
2.2.6. speakToAs	12
2.3. Contact and Resource properties	13
2.3.1. emails	13
2.3.2. phones	14
2.3.3. online	15
2.3.4. photos	15
2.3.5. preferredContactMethod	16
2.3.6. preferredContactLanguages	16
2.4. Address and Location properties	17

2.4.1. addresses	17
2.5. Multilingual properties	19
2.5.1. localizations	19
2.6. Additional properties	19
2.6.1. anniversaries	19
2.6.2. personalInfo	20
2.6.3. notes	21
2.6.4. categories	21
2.6.5. timeZones	21
3. CardGroup	21
3.1. Group properties	21
3.1.1. @type	21
3.1.2. uid	21
3.1.3. members	22
3.1.4. name	22
3.1.5. card	22
4. Implementation Status	22
4.1. IIT-CNR/Registro.it	22
5. IANA Considerations	23
6. Security Considerations	23
7. References	23
7.1. Normative References	23
7.2. Informative References	24
Authors' Addresses	25

1. Introduction

This document defines a data model for contact card data normally used in address book or directory applications and services. It aims to be an alternative to the vCard data format [RFC6350] and to provide a JSON-based standard representation of contact card data.

The key design considerations for this data model are as follows:

- * Most of the initial set of attributes should be taken from the vCard data format [RFC6350] and extensions ([RFC6473], [RFC6474], [RFC6715], [RFC6869], [RFC8605]). The specification should add new attributes or value types, or not support existing ones, where appropriate. Conversion between the data formats need not fully preserve semantic meaning.
- * The attributes of the cards data represented must be described as a simple key-value pair, reducing complexity of its representation.
- * The data model should avoid all ambiguities and make it difficult to make mistakes during implementation.

- * Extensions, such as new properties and components, MUST NOT lead to requiring an update to this document.

The representation of this data model is defined in the I-JSON format [RFC7493], which is a strict subset of the JavaScript Object Notation (JSON) Data Interchange Format [RFC8259]. Using JSON is mostly a pragmatic choice: its widespread use makes Card easier to adopt, and the availability of production-ready JSON implementations eliminates a whole category of parser-related interoperability issues.

1.1. Relation to the xCard and jCard formats

The xCard [RFC6351] and jCard [RFC7095] specifications define alternative representations for vCard data, in XML and JSON format respectively. Both explicitly aim to not change the underlying data model. Accordingly, they are regarded as equal to vCard in the context of this document.

1.2. Terminology

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

1.3. Vendor-specific Property Extensions and Values

Vendors MAY add additional properties to the contact object to support their custom features. To avoid conflict, the names of these properties MUST be prefixed by a domain name controlled by the vendor followed by a colon, e.g., "example.com:customprop". If the value is a new JSContact object, it either MUST include an "@type" property, or it MUST explicitly be specified to not require a type designator. The type name MUST be prefixed with a domain name controlled by the vendor.

Some JSContact properties allow vendor-specific value extensions. Such vendor-specific values MUST be prefixed by a domain name controlled by the vendor followed by a colon, e.g., "example.com:customrel".

Vendors are strongly encouraged to register any new property values or extensions that are useful to other systems as well, rather than use a vendor-specific prefix.

1.4. Type Signatures

Type signatures are given for all JSON values in this document. The following conventions are used:

- * * - The type is undefined (the value could be any type, although permitted values may be constrained by the context of this value).
- * String - The JSON string type.
- * Number - The JSON number type.
- * Boolean - The JSON boolean type.
- * A[B] - A JSON object where the keys are all of type A, and the values are all of type B.
- * A[] - An array of values of type A.
- * A|B - The value is either of type A or of type B.

1.5. Data types

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

1.5.1. Context

Contact information typically is associated with a context in which it should be used. For example, someone might have distinct phone numbers for work and private contexts. The Context data type enumerates common contexts.

Common context values are:

- * private: The contact information may be used to contact the card holder in a private context.
- * work: The contact information may be used to contact the card holder in a professional context.

Additional allowed values may be defined in the properties or data types that make use of the Context data type, registered in a future RFC, or a vendor-specific value.

1.5.2. Id

Where Id is given as a data type, it means a String of at least 1 and a maximum of 255 octets in size, and it MUST only contain characters from the URL and Filename Safe base64url alphabet, as defined in Section 5 of [RFC4648], excluding the pad character (=). This means the allowed characters are the ASCII alphanumeric characters (A-Za-z0-9), hyphen (-), and underscore (_).

In many places in JSContact a JSON map is used where the map keys are of type Id and the map values are all the same type of object. This construction represents an unordered set of objects, with the added advantage that each entry has a name (the corresponding map key). This allows for more concise patching of objects, and, when applicable, for the objects in question to be referenced from other objects within the JSContact object.

Unless otherwise specified for a particular property, there are no uniqueness constraints on an Id value (other than, of course, the requirement that you cannot have two values with the same key within a single JSON map). For example, two Card objects might use the same Ids in their respective photos properties. Or within the same Card object the same Id could appear in the emails and phones properties. These situations do not imply any semantic connections among the objects.

1.5.3. PatchObject

A PatchObject is of type String[*], and represents an unordered set of patches on a JSON object. Each key is a path represented in a subset of JSON pointer format [RFC6901]. The paths have an implicit leading /, so each key is prefixed with / before applying the JSON pointer evaluation algorithm.

A patch within a PatchObject is only valid if all of the following conditions apply:

1. The pointer MUST NOT reference inside an array (i.e., you MUST NOT insert/delete from an array; the array MUST be replaced in its entirety instead).
2. All parts prior to the last (i.e., the value after the final slash) MUST already exist on the object being patched.
3. There MUST NOT be two patches in the PatchObject where the pointer of one is the prefix of the pointer of the other, e.g., addresses/1/city and addresses.

4. The value for the patch MUST be valid for the property being set (of the correct type and obeying any other applicable restrictions), or if null the property MUST be optional.

The value associated with each pointer determines how to apply that patch:

- * If null, remove the property from the patched object. If the key is not present in the parent, this a no-op.
- * If non-null, set the value given as the value for this property (this may be a replacement or addition to the object being patched).

A PatchObject does not define its own @type property. Instead, a @type property in a patch MUST be handled as any other patched property value.

Implementations MUST reject in its entirety a PatchObject if any of its patches is invalid. Implementations MUST NOT apply partial patches.

1.5.4. Preference

This data type allows to define a preference order on same-typed contact information. For example, a card holder may have two email addresses and prefer to be contacted with one of them.

A preference value MUST be an integer number in the range 1 and 100. Lower values correspond to a higher level of preference, with 1 being most preferred. If no preference is set, then the contact information MUST be interpreted as being least preferred.

Note that the preference only is defined in relation to contact information of the same type. For example, the preference orders within emails and phone numbers are independent of each other. Also note that the `_preferredContactMethod_` property allows to define a preferred contact method across method types.

1.5.5. UnsignedInt

Where UnsignedInt is given as a data type, it means an integer in the range $0 \leq \text{value} \leq 2^{53}-1$, represented as a JSON Number.

1.5.6. UTCDateTime

This is a string in [RFC3339] date-time format, with the further restrictions that any letters MUST be in uppercase, and the time offset MUST be the character Z. Fractional second values MUST NOT be included unless non-zero and MUST NOT have trailing zeros, to ensure there is only a single representation for each date-time.

For example, 2010-10-10T10:10:10.003Z is conformant, but 2010-10-10T10:10:10.000Z is invalid and is correctly encoded as 2010-10-10T10:10:10Z.

2. Card

MIME type: application/jscontact+json;type=card

A Card object stores information about a person, organization or company.

2.1. Metadata properties

2.1.1. @type

Type: String (mandatory).

Specifies the type of this object. This MUST be Card.

2.1.2. uid

Type: String (mandatory).

An identifier, used to associate the object as the same across different systems, addressbooks and views. [RFC4122] describes a range of established algorithms to generate universally unique identifiers (UUID), and the random or pseudo-random version is recommended. For compatibility with [RFC6350] UUIDs, implementations MUST accept both URI and free-form text.

2.1.3. prodId

Type: String (optional).

The identifier for the product that created the Card object.

2.1.4. created

Type: UTCDateTime (optional).

The date and time when this Card object was created.

2.1.5. updated

Type: `UTCDateTime` (optional).

The date and time when the data in this Card object was last modified.

2.1.6. kind

Type: `String` (optional). The kind of the entity the Card represents.

The value MUST be either one of the following values, registered in a future RFC, or a vendor-specific value:

- * `individual`: a single person
- * `org`: an organization
- * `location`: a named location
- * `device`: a device, such as appliances, computers, or network elements
- * `application`: a software application

2.1.7. relatedTo

Type: `String[Relation]` (optional).

Relates the object to other Card and CardGroup objects. This is represented as a map, where each key is the uid of the related Card or CardGroup and the value defines the relation. The Relation object has the following properties:

- * `@type`: `String` (mandatory). Specifies the type of this object. This MUST be `Relation`.
- * `relation`: `String[Boolean]` (optional, default: empty Object) Describes how the linked object is related to the linking object. The relation is defined as a set of relation types. If empty, the relationship between the two objects is unspecified. Keys in the set MUST be one of the RELATED property [RFC6350] type parameter values, or an IANA-registered value, or a vendor-specific value. The value for each key in the set MUST be true.

2.1.8. language

Type: String (optional).

This defines the locale in which free-text property values can be assumed to be written in. The value MUST be a language tag as defined in [RFC5646]. Note that such values MAY be localized in the localizations Section 2.5.1 property.

2.2. Name and Organization properties

2.2.1. name

Type: Name (optional).

The name of the entity represented by this Card.

A Name object has the following properties

- * @type: Name (mandatory). Specifies the type of this object. This MUST be Name.
- * components: NameComponent[] (mandatory). The components making up the name. The component list MUST have at least one entry. Name components SHOULD be ordered such that their values joined by whitespace produce a valid full name of this entity. Doing so, implementations MAY ignore any components of type separator.
- * locale: String (optional). The locale of the name. The value MUST be a language tag as defined [RFC5646].

A NameComponent object has the following properties:

- * @type: String (mandatory). Specifies the type of this object. This MUST be NameComponent.
- * value: String (mandatory). The value of this name component.
- * type: String (mandatory). The type of this name component. The value MUST be either one of the following values, registered in a future RFC, or a vendor-specific value:
 - prefix. The value is a honorific title(s), e.g. "Mr", "Ms", "Dr".
 - given. The value is a given name, also known as "first name", "personal name".

- surname. The value is a surname, also known as "last name", "family name".
 - middle. The value is a middle name, also known as "additional name".
 - suffix. The value is a honorific suffix, e.g. "B.A.", "Esq.".
 - separator. A formatting separator for two name components. The value property of the component includes the verbatim separator, for example a newline character.
- * nth: UnsignedInt (optional, default: 1). Defines the rank of this name component to other name components of the same type. If set, the property value MUST be higher than or equal to 1.

For example, two name components of type surname may have their nth property value set to 1 and 2, respectively. In this case, the first name component defines the surname, and the second name component the secondary surname.

Note that this property value does not indicate the order in which to print name components of the same type. Some cultures print the secondary surname before the first surname, others the first before the second. Implementations SHOULD inspect the locale property of the Name object to determine the appropriate formatting. They MAY print name components in order of appearance in the components property of the Name object.

2.2.2. fullName

Type: String (optional).

The full name (e.g. the personal name and surname of an individual, the name of an organization) of the entity represented by this card. The purpose of this property is to define a name, even if the individual name components are not known. In addition, it is meant to provide alternative versions of the name for internationalisation. Implementations SHOULD prefer using the `_name_` property over this one and SHOULD NOT store the concatenated name component values in this property.

2.2.3. nickNames

Type: String[] (optional).

The nick names of the entity represented by this card.

2.2.4. organizations

Type: Id[Organization] (optional).

The companies or organization names and units associated with this card. An Organization object has the following properties:

- * @type: String (mandatory). Specifies the type of this object. This MUST be Organization.
- * name: String (mandatory). The name of this organization.
- * units: String[] (optional). Additional levels of organizational unit names.

2.2.5. titles

Type : Id[Title] (optional).

The job titles or functional positions of the entity represented by this card. A Title has object the following properties:

- * @type: String (mandatory). Specifies the type of this object. This MUST be Title.
- * title: String (mandatory). The title of the entity represented by this card.
- * organization: Id (optional). The id of the organization in which this title is held.

2.2.6. speakToAs

Type: SpeakToAs (optional).

Provides information how to address, speak to or refer to the entity that is represented by this card. A SpeakToAs object has the following properties, of which at least one property other than @type MUST be set:

- * @type: String (mandatory). Specifies the type of this object. This MUST be SpeakToAs.
- * grammaticalGender: String (optional). Defines which grammatical gender to use in salutations and other grammatical constructs. Allowed values are:
 - animate

- female
- inanimate
- male
- neuter

Note that the grammatical gender does not allow to infer the gender identities or biological sex of the contact.

- * `pronouns`: String (optional). Defines the gender pronouns that the contact chooses to use for themselves. Any value or form is allowed. Examples in English include she/her and they/them/theirs.

The property values SHOULD be localized in the language defined in the language property. They MAY be overridden in the localizations property (Section 2.5.1).

2.3. Contact and Resource properties

2.3.1. emails

Type: Id[EmailAddress] (optional).

The email addresses to contact the entity represented by this card. An EmailAddress object has the following properties:

- * `@type`: String (mandatory). Specifies the type of this object. This MUST be EmailAddress.
- * `email`: String (mandatory). The email address. This MUST be an `_addr-spec_` value as defined in Section 3.4.1 of [RFC5322].
- * `contexts`: Context[Boolean] (optional) The contexts in which to use this email address. The value for each key in the object MUST be true.
- * `pref`: Preference (optional) The preference of this email address in relation to other email addresses.
- * `label`: String (optional). A label describing the value in more detail.

2.3.2. phones

Type: Id[Phone] (optional).

The phone numbers to contact the entity represented by this card. A Phone object has the following properties:

- * @type: String (mandatory). Specifies the type of this object. This MUST be Phone.
- * phone: String (mandatory). The phone value, as either a URI or a free-text phone number. Typical URI schemes are the [RFC3966] tel or [RFC3261] sip schemes, but any URI scheme is allowed.
- * features: String[Boolean] (optional). The set of contact features that this phone number may be used for. The set is represented as an object, with each key being a method type. The value for each key in the object MUST be true. The method type MUST be either one of the following values, registered in a future RFC, or a vendor-specific value:
 - voice The number is for calling by voice.
 - fax The number is for sending faxes.
 - pager The number is for a pager or beeper.
 - text The number supports text messages (SMS).
 - cell The number is for a cell phone.
 - textphone The number is for a device for people with hearing or speech difficulties.
 - video The number supports video conferencing.
- * contexts: Context[Boolean] (optional) The contexts in which to use this number. The value for each key in the object MUST be true.
- * pref: Preference (optional) The preference of this number in relation to other numbers.
- * label: String (optional). A label describing the value in more detail.

2.3.3. online

Type: Id[Resource] (optional).

The online resources and services that are associated with the entity represented by this card. A Resource object has the following properties:

- * @type: String (mandatory). Specifies the type of this object. This MUST be Resource.
- * resource: String (mandatory). The resource value, where the allowed value form is defined by the _type_ property. In any case the value MUST NOT be empty.
- * type: String (optional). The type of the resource value. Allowed values are:
 - uri The resource value is a URI, e.g. a website link. This MUST be a valid _URI_ as defined in Section 3 of [RFC3986] and updates.
 - username The resource value is a username associated with the entity represented by this card (e.g. for social media, or an IM client). The _label_ property SHOULD be included to identify what service this is for. For compatibility between clients, this label SHOULD be the canonical service name, including capitalisation. e.g. Twitter, Facebook, Skype, GitHub, XMPP. The resource value may be any non-empty free text.
- * mediaType: String (optional). Used for URI resource values. Provides the media type [RFC2046] of the resource identified by the URI.
- * contexts: Context[Boolean] (optional) The contexts in which to use this resource. The value for each key in the object MUST be true.
- * pref: Preference (optional) The preference of this resource in relation to other resources.
- * label: String (optional). A label describing the value in more detail.

2.3.4. photos

Type: Id[File] (optional).

A map of photo ids to File objects that contain photographs or images associated with this card. A typical use case is to include an avatar for display along the contact name.

A File object has the following properties:

- * @type: String (mandatory). Specifies the type of this object. This MUST be File.
- * href: String (mandatory). A URI where to fetch the data of this file.
- * mediaType: String (optional). The content-type of the file, if known.
- * size: UnsignedInt (optional). The size, in octets, of the file when fully decoded (i.e., the number of octets in the file the user would download), if known.
- * pref: Preference (optional) The preference of this photo in relation to other photos.
- * label: String (optional). A label describing the value in more detail.

2.3.5. preferredContactMethod

Type : String (optional)

Defines the preferred method to contact the holder of this card. The value MUST be the property names: emails, phones, online.

2.3.6. preferredContactLanguages

Type : String[ContactLanguage[]] (optional)

Defines the preferred languages for contacting the entity associated with this card. The keys in the object MUST be [RFC5646] language tags. The values are a (possibly empty) list of contact language preferences for this language. A valid ContactLanguage object MUST have at least one of its properties set.

A ContactLanguage object has the following properties:

- * @type: String (mandatory). Specifies the type of this object. This MUST be ContactLanguage.

- * context: Context (optional). Defines the context in which to use this language.
- * pref: Preference (optional). Defines the preference of this language in relation to other languages of the same context.

Also see the definition of the VCARD LANG property (Section 6.4.4., [RFC6350]).

2.4. Address and Location properties

2.4.1. addresses

Type: Id[Address] (optional).

A map of address ids to Address objects, containing physical locations. An Address object has the following properties:

- * @type: String (mandatory). Specifies the type of this object. This MUST be Address.
- * fullAddress: String (optional). The complete address, excluding type and label. This property is mainly useful to represent addresses of which the individual address components are unknown, or to provide localized representations.
- * street: StreetComponent[] (optional). The street address. The concatenation of the component values, separated by whitespace, SHOULD result in a valid street address for the address locale. Doing so, implementations MAY ignore any separator components. The StreetComponent object type is defined in the paragraph below.
- * locality: String (optional). The city, town, village, post town, or other locality within which the street address may be found.
- * region: String (optional). The province, such as a state, county, or canton within which the locality may be found.
- * country: String (optional). The country name.
- * postcode: String (optional). The postal code, post code, ZIP code or other short code associated with the address by the relevant country's postal system.
- * countryCode: String (optional). The ISO-3166-1 country code.
- * coordinates: String (optional) A [RFC5870] "geo:" URI for the address.

- * `timeZone`: String (optional) Identifies the time zone this address is located in. This either MUST be a time zone name registered in the IANA Time Zone Database (<https://www.iana.org/time-zones>), or it MUST be a valid `TimeZoneId` as defined in [RFC8984]. For the latter, a corresponding time zone MUST be defined in the `timeZones` property.
- * `contexts`: Context[Boolean] (optional). The contexts of the address information. In addition to the common contexts, allowed values are:
 - `billing` An address to be used for billing.
 - `postal` An address to be used for delivering physical items. The value for each key in the object MUST be true.
- * `pref`: Preference (optional) The preference of this address in relation to other addresses.
- * `label`: String (optional). A label describing the value in more detail.

A `StreetComponent` object has the following properties:

- * `@type`: String (mandatory). Specifies the type of this object. This MUST be `StreetComponent`.
- * `type`: String (mandatory). The type of this street component. The value MUST be either one of the following values, registered in a future RFC, or a vendor-specific value:
 - `name`. The street name.
 - `number`. The street number.
 - `apartment`. The apartment number or identifier.
 - `room`. The room number or identifier.
 - `extension`. The extension designation or box number.
 - `direction`. The cardinal direction, e.g. "North".
 - `building`. The building or building part this address is located in.
 - `floor`. The floor this address is located on.

- postOfficeBox. The post office box number or identifier.
 - separator. A separator for two street components. The value property of the component includes the verbatim separator, for example a newline character.
 - unknown. A name component value for which no type is known.
- * value: String (mandatory). The value of this street component.

2.5. Multilingual properties

2.5.1. localizations

Type: String[PatchObject] (optional).

A map of language tags [RFC5646] to patches, which localize a property value into the locale of the respective language tag. The paths in the PatchObject keys are relative to the Card object that includes the localizations property. A patch MUST NOT target the localizations property.

The following example shows a Card object, where one of its addresses Tokyo is localized for the jp locale.

```
"@type": "Card",
...
"addresses": {
  "addr1": {
    "@type": "Address",
    "locality": "Tokyo",
  }
},
"localizations": {
  "jp": {
    "addresses/addr1/locality":""
  }
}
```

Figure 1

2.6. Additional properties

2.6.1. anniversaries

Type : Id[Anniversary] (optional).

These are memorable dates and events for the entity represented by this card. An Anniversary object has the following properties:

- * @type: String (mandatory). Specifies the type of this object. This MUST be Anniversary.
- * type: String (optional). Specifies the type of the anniversary. This RFC predefines the following types, but implementations MAY use additional values:
 - birth: a birth day anniversary
 - death: a death day anniversary
- * date: String (mandatory). The date of this anniversary, in the form "YYYY-MM-DD" (any part may be all 0s for unknown) or a [RFC3339] timestamp.
- * place: Address (optional). An address associated with this anniversary, e.g. the place of birth or death.
- * label: String (optional). A label describing the value in more detail.

2.6.2. personalInfo

Type: Id[PersonalInformation] (optional).

Defines personal information about the entity represented by this card. A PersonalInformation object has the following properties:

- * @type: String (mandatory). Specifies the type of this object. This MUST be PersonalInformation.
- * type: String (mandatory). Specifies the type for this personal information. Allowed values are:
 - expertise: a field of expertise or credential
 - hobby: a hobby
 - interest: an interest
- * value: String (mandatory). The actual information. This generally is free-text, but future specifications MAY restrict allowed values depending on the type of this PersonalInformation.

- * `level`: String (optional) Indicates the level of expertise, or engagement in hobby or interest. Allowed values are: high, medium and low.
- * `label`: String (optional). A label describing the value in more detail.

2.6.3. notes

Type: String (optional).

Arbitrary notes about the entity represented by this card.

2.6.4. categories

Type: String[Boolean] (optional). The set of free-text or URI categories that relate to the card. The set is represented as an object, with each key being a category. The value for each key in the object MUST be true.

2.6.5. timeZones

Type: String[TimeZone] (optional). Maps identifiers of custom time zones to their time zone definitions. For a description of this property see the timeZones property definition in [RFC8984].

3. CardGroup

MIME type: application/jscontact+json;type=cardgroup

A CardGroup object represents a group of cards. Its members may be Cards or CardGroups.

3.1. Group properties

3.1.1. @type

Type: String (mandatory).

Specifies the type of this object. This MUST be CardGroup.

3.1.2. uid

Type: String (mandatory). The uid of this group. Both CardGroup and Card share the same namespace for the uid property.

3.1.3. members

Type: String[Boolean] (mandatory). The members of this group.

The set is represented as an object, with each key being the uid of another Card or CardGroup. The value for each key in the object MUST be true.

3.1.4. name

Type: String (optional). The user-visible name for the group, e.g. "Friends". This may be any UTF-8 string of at least 1 character in length and maximum 255 octets in size. The same name may be used by two different groups.

3.1.5. card

Type: Card (optional). The card that represents this group.

4. Implementation Status

NOTE: Please remove this section and the reference to [RFC7942] prior to publication as an RFC. This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [RFC7942]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist. According to [RFC7942], "this will allow reviewers and working groups to assign due consideration to documents that have the benefit of running code, which may serve as evidence of valuable experimentation and feedback that have made the implemented protocols more mature. It is up to the individual working groups to use this information as they see fit".

4.1. IIT-CNR/Registro.it

- * Responsible Organization: Institute of Informatics and Telematics of National Research Council (IIT-CNR)/Registro.it
- * Location: <https://rdap.pubtest.nic.it/>
(<https://rdap.pubtest.nic.it/>)

- * **Description:** This implementation includes support for RDAP queries using data from the public test environment of .it ccTLD. The RDAP server returns responses including Card in place of jCard when queries contain the parameter jscard=1.
- * **Level of Maturity:** This is an "alpha" test implementation.
- * **Coverage:** This implementation includes all of the features described in this specification.
- * **Contact Information:** Mario Loffredo, mario.loffredo@iit.cnr.it

5. IANA Considerations

TBD

6. Security Considerations

TBD

7. References

7.1. Normative References

- [RFC2046] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, DOI 10.17487/RFC2046, November 1996, <<https://www.rfc-editor.org/info/rfc2046>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, DOI 10.17487/RFC4122, July 2005, <<https://www.rfc-editor.org/info/rfc4122>>.
- [RFC5646] Phillips, A., Ed. and M. Davis, Ed., "Tags for Identifying Languages", BCP 47, RFC 5646, DOI 10.17487/RFC5646, September 2009, <<https://www.rfc-editor.org/info/rfc5646>>.
- [RFC5870] Mayrhofer, A. and C. Spanring, "A Uniform Resource Identifier for Geographic Locations ('geo' URI)", RFC 5870, DOI 10.17487/RFC5870, June 2010, <<https://www.rfc-editor.org/info/rfc5870>>.

- [RFC6350] Perreault, S., "vCard Format Specification", RFC 6350, DOI 10.17487/RFC6350, August 2011, <<https://www.rfc-editor.org/info/rfc6350>>.
- [RFC6351] Perreault, S., "xCard: vCard XML Representation", RFC 6351, DOI 10.17487/RFC6351, August 2011, <<https://www.rfc-editor.org/info/rfc6351>>.
- [RFC6901] Bryan, P., Ed., Zyp, K., and M. Nottingham, Ed., "JavaScript Object Notation (JSON) Pointer", RFC 6901, DOI 10.17487/RFC6901, April 2013, <<https://www.rfc-editor.org/info/rfc6901>>.
- [RFC7095] Kewisch, P., "jCard: The JSON Format for vCard", RFC 7095, DOI 10.17487/RFC7095, January 2014, <<https://www.rfc-editor.org/info/rfc7095>>.
- [RFC7493] Bray, T., Ed., "The I-JSON Message Format", RFC 7493, DOI 10.17487/RFC7493, March 2015, <<https://www.rfc-editor.org/info/rfc7493>>.
- [RFC7942] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", BCP 205, RFC 7942, DOI 10.17487/RFC7942, July 2016, <<https://www.rfc-editor.org/info/rfc7942>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <<https://www.rfc-editor.org/info/rfc8259>>.
- [RFC8984] Jenkins, N. and R. Stepanek, "JSCalendar: A JSON Representation of Calendar Data", RFC 8984, DOI 10.17487/RFC8984, July 2021, <<https://www.rfc-editor.org/info/rfc8984>>.

7.2. Informative References

- [RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, DOI 10.17487/RFC3261, June 2002, <<https://www.rfc-editor.org/info/rfc3261>>.

- [RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002, <<https://www.rfc-editor.org/info/rfc3339>>.
- [RFC3966] Schulzrinne, H., "The tel URI for Telephone Numbers", RFC 3966, DOI 10.17487/RFC3966, December 2004, <<https://www.rfc-editor.org/info/rfc3966>>.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<https://www.rfc-editor.org/info/rfc3986>>.
- [RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <<https://www.rfc-editor.org/info/rfc4648>>.
- [RFC5322] Resnick, P., Ed., "Internet Message Format", RFC 5322, DOI 10.17487/RFC5322, October 2008, <<https://www.rfc-editor.org/info/rfc5322>>.
- [RFC6473] Saint-Andre, P., "vCard KIND:application", RFC 6473, DOI 10.17487/RFC6473, December 2011, <<https://www.rfc-editor.org/info/rfc6473>>.
- [RFC6474] Li, K. and B. Leiba, "vCard Format Extensions: Place of Birth, Place and Date of Death", RFC 6474, DOI 10.17487/RFC6474, December 2011, <<https://www.rfc-editor.org/info/rfc6474>>.
- [RFC6715] Cauchie, D., Leiba, B., and K. Li, "vCard Format Extensions: Representing vCard Extensions Defined by the Open Mobile Alliance (OMA) Converged Address Book (CAB) Group", RFC 6715, DOI 10.17487/RFC6715, August 2012, <<https://www.rfc-editor.org/info/rfc6715>>.
- [RFC6869] Salgueiro, G., Clarke, J., and P. Saint-Andre, "vCard KIND:device", RFC 6869, DOI 10.17487/RFC6869, February 2013, <<https://www.rfc-editor.org/info/rfc6869>>.
- [RFC8605] Hollenbeck, S. and R. Carney, "vCard Format Extensions: ICANN Extensions for the Registration Data Access Protocol (RDAP)", RFC 8605, DOI 10.17487/RFC8605, May 2019, <<https://www.rfc-editor.org/info/rfc8605>>.

Authors' Addresses

Robert Stepanek
FastMail
PO Box 234, Collins St West
Melbourne VIC 8007
Australia

Email: rsto@fastmailteam.com

Mario Loffredo
IIT-CNR
Via Moruzzi,1
56124 Pisa
Italy

Email: mario.loffredo@iit.cnr.it

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 10 July 2022

A. Melnikov
Isode Ltd
6 January 2022

S/MIME signature verification extension to JMAP
draft-ietf-jmap-smime-12

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.

Table of Contents

1. Introduction	2
2. Conventions Used in This Document	2
3. Addition to the capabilities object	3
4. Extension for S/MIME signature verification	3
4.1. Extension to Email/get	3
4.1.1. "smimeStatus" response property extensibility	7
4.2. Extension to Email/query	8
4.3. Interaction with Email/changes	8
5. IANA Considerations	9
5.1. JMAP capability registration for "smimeverify"	9
6. Security Considerations	9
7. References	10
7.1. Normative References	10
7.2. Informative References	10
Appendix A. Acknowledgements	10
Author's Address	10

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.

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

This might result in the following response:

```
[ [ "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

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8550] Schaad, J., Ramsdell, B., and S. Turner, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 4.0 Certificate Handling", RFC 8550, DOI 10.17487/RFC8550, April 2019, <<https://www.rfc-editor.org/info/rfc8550>>.
- [RFC8551] Schaad, J., Ramsdell, B., and S. Turner, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 4.0 Message Specification", RFC 8551, DOI 10.17487/RFC8551, April 2019, <<https://www.rfc-editor.org/info/rfc8551>>.
- [RFC8620] Jenkins, N. and C. Newman, "The JSON Meta Application Protocol (JMAP)", RFC 8620, DOI 10.17487/RFC8620, July 2019, <<https://www.rfc-editor.org/info/rfc8620>>.
- [RFC8621] Jenkins, N. and C. Newman, "The JSON Meta Application Protocol (JMAP) for Mail", RFC 8621, DOI 10.17487/RFC8621, August 2019, <<https://www.rfc-editor.org/info/rfc8621>>.

7.2. Informative References

- [RFC1847] Galvin, J., Murphy, S., Crocker, S., and N. Freed, "Security Multiparts for MIME: Multipart/Signed and Multipart/Encrypted", RFC 1847, DOI 10.17487/RFC1847, October 1995, <<https://www.rfc-editor.org/info/rfc1847>>.

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

Alexey Melnikov
Isode Ltd
14 Castle Mews
Hampton
TW12 2NP
United Kingdom

Email: Alexey.Melnikov@isode.com