JMAP
Internet-Draft
Intended status: Standards Track
Expires: 14 October 2024

N.M. Jenkins, Ed.
Fastmail
M. Douglass, Ed.
Spherical Cow Group

12 April 2024

JMAP for Calendars
draft-ietf-jmap-calendars-17

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 14 October 2024.

Copyright Notice

Table of Contents

1. Introduction

JMAP ([RFC8620] JSON Meta Application Protocol) is a generic protocol for synchronizing data, such as mail, calendars or contacts, between a client and a server. It is optimized for mobile and web environments, and aims to provide a consistent interface to different data types.

This specification defines a data model for synchronizing calendar data between a client and a server using JMAP. The data model is designed to allow a server to provide consistent access to the same data via CalDAV [RFC4791] as well as JMAP, however the functionality offered over the two protocols may differ. Unlike CalDAV, this specification does not define access to tasks or journal entries (VTODO or VJOURNAL iCalendar components in CalDAV).

1.1. Notational Conventions

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

Type signatures, examples, and property descriptions in this document follow the conventions established in Section 1.1 of [RFC8620].

1.2. Data Types

The Id data type defined in Section 1.2 of [RFC8620] is used in this document. So too are the UnsignedInt, UTCDateTime, LocalDateTime, Duration, and TimeZoneId data types defined in Sections 1.4.3, 1.4.4, 1.4.5, 1.4.6, and 1.4.8 of [RFC8984] respectively.

1.3. Terminology

The same terminology is used in this document as in the core JMAP specification, see Section 1.6 of [RFC8620].

The terms ParticipantIdentity, Calendar, CalendarEvent, and CalendarEventNotification (with these specific capitalizations) are used to refer to the data types defined in this document and instances of those data types.

1.4. Data Model Overview

An Account (see Section 1.6.2 of [RFC8620]) 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 JSCalendar Event [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.

In servers with support for JMAP Sharing [I-D.ietf-jmap-sharing], 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 (Section 4.1.2 of [RFC8984]), 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 Section 2 of [RFC8620]. This document defines three 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, except for "CalendarEvent/parse". 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:

* *maxCalendarsPerEvent*: UnsignedInt|null
  The maximum number of Calendars (see Section 4) that can be can assigned to a single CalendarEvent object (see Section 5). 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*: LocalDateTime
  The earliest date-time the server is willing to accept for any date stored in a CalendarEvent.
* *maxDateTime*: LocalDateTime
  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*: UnsignedInt|null
  The maximum number of participants a single event may have, or null for no limit.
* *mayCreateCalendar*: Boolean
  If true, the user may create a calendar in this account.

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

Represents support for the Principal/getAvailability method. Any account with this capability MUST also have the urn:ietf:params:jmap:principals capability (see Section 1.5.1 of [I-D.ietf-jmap-sharing]).

The value of this property in the JMAP Session capabilities property is an empty object.

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

* *maxAvailabilityDuration*: Duration
  The maximum duration over which the server is prepared to calculate availability in a single call (see Section 2.2).

1.5.3. urn:ietf:params:jmap:calendars:parse

This represents support for the CalendarEvent/parse method (see Section 5.12). The value of this property is an empty object in both the JMAP session capabilities property and an account's accountCapabilities property.

2. Principals and Sharing

For systems that also support JMAP Sharing [I-D.ietf-jmap-sharing], the urn:ietf:params:jmap: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). The corresponding Account object can be found in the principal's "accounts" property, as per Section 2 of [I-D.ietf-jmap-sharing].
* *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 4)?
* *calendarAddress*: String
  If this principal may be added as a participant to an event, this is the calendarAddress to use to represent it.
* *sendTo*: String[String]|null
  If this principal may be added as a participant to an event, this is the Participant#sendTo property to add (see Section 4.4.5 of [RFC8984]) for scheduling messages to reach it.

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*: UTCDateTime
  The start time (inclusive) of the period for which to return availability.
* *utcEnd*: UTCDateTime
  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 JSCalendar Event 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.
* 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*: UTCDateTime

  The start time (inclusive) of the period this represents.

* *utcEnd*: UTCDateTime

  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" and the principal's "participationStatus" is "attending".
  - tentative: The event status is "tentative" or the principal's "participationStatus" is "tentative".
  - unavailable: The principal is not available for scheduling at this time

for any other reason.

* *event*: JSCalendar Event|null

  The JSCalendar Event 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 JSCalendar Event 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".

* *calendarAddress*: String

  The URI that represents this participant for scheduling. This URI may also be the URI for one of the sendTo methods.

* *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.

* *isDefault*: Boolean (server-set)

  This SHOULD be true for exactly one participant identity in any account, and MUST NOT be true for more than one participant identity within an account. The default identity should be used by clients whenever they need to choose an identity for the user within this account, and they do not have any other information on which to make a choice. For example, if creating a scheduled event in this account, the default identity may be automatically added as an owner. (But the client may ignore this if, for example, it has its own feature to allow users to choose which identity to use based on the invitees.)

A participant in an event corresponds to a ParticipantIdentity if the calendarAddress property of the participant is equivalent to the calendarAddress property of the identity after syntax-based normalisation, as per Section 6.2.2 of [RFC3986].

The following JMAP methods are supported.

3.1. ParticipantIdentity/get

This is a standard "/get" method as described in Section 5.1 of [RFC8620]. The "ids" argument may be null to fetch all at once.

3.2. ParticipantIdentity/changes

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

3.3. ParticipantIdentity/set

This is a standard "/set" method as described in Section 5.3 of [RFC8620], but with the following additional request argument:

* *onSuccessSetIsDefault*: Id|null
  If an id is given, and all creates, updates and destroys (if any) succeed without error, the server will try to set this identity as the default. (For references to ParticipantIdentity creations, this is equivalent to a creation-reference, so the id will be the creation id prefixed with a "#".)

If the id is not found, or the change is not permitted by the server for policy reasons, it MUST be ignored and the currently default ParticipantIdentity (if any) will remain as such. No error is returned to the client in this case.

As per Section 5.3 of [RFC8620], if the default is successfully changed, any changed objects MUST be reported in either the "created" or "updated" argument in the response as appropriate, with the server-set value included.

The server MAY restrict the uri values the user may claim, for example only allowing mailto: URIs with email addresses that belong to the user. A standard forbidden error is returned to reject non-permissible changes.

4. Calendars

A Calendar is a named collection of events. All events are associated with at least one calendar.

A *Calendar* object has the following properties:

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

The user-visible name of the calendar. This may be any UTF-8 string of at least 1 character in length and maximum 255 octets in size.

* *description*: String|null (default: null)
  An optional longer-form description of the calendar, to provide context in shared environments where users need more than just the name.
* *color*: String|null (default: null)
  A color to be used when displaying events associated with the calendar.
  If not null, the value MUST be a case-insensitive color name taken from the set of names defined in Section 4.3 of CSS Color Module Level 3 COLORS (https://www.w3.org/TR/css-color-3/), or an RGB value in hexadecimal notation, as defined in Section 4.2.1 of CSS Color Module Level 3.
  The color SHOULD have sufficient contrast to be used as text on a white background.

* *sortOrder*: UnsignedInt (default: 0)

  Defines the sort order of calendars when presented in the client's UI, so it is consistent between devices. The number MUST be an integer in the range 0 <= sortOrder < 2^(31.)

  A calendar with a lower order is to 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

  True if the user has indicated they wish to see this Calendar in their client. This should default to false for Calendars in shared accounts the user has access to and true for any new Calendars created by the user themself.

  If false, the calendar should only be displayed when the user explicitly requests it or to offer it for the user to subscribe to. For example, a company may have a large number of shared calendars which all employees have permission to access, but you would only subscribe to the ones you care about and want to be able to have normally accessible.

* *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.

* *isDefault*: Boolean (server-set)

  This SHOULD be true for exactly one calendar in any account, and MUST NOT be true for more than one calendar within an account. The default calendar should be used by clients whenever they need to choose a calendar for the user within this account, and they do not have any other information on which to make a choice. For example, if the user creates a new event, the client may automatically set the event as belonging to the default calendar from the user's primary account.

* *includeInAvailability*: String

  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).

  This should default to "all" for the calendars in the user's own account, and "none" for calendars shared with the user.

* *defaultAlertsWithTime*: Id[Alert]|null

  A map of alert ids to Alert objects (see Section 4.5.2 of [RFC8984]) 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.

  If omitted on creation, the default is server dependent. For example, servers may choose to always default to null, or may copy the alerts from the default calendar.

* *defaultAlertsWithoutTime*: Id[Alert]|null

A map of alert ids to Alert objects (see Section 4.5.2 of [RFC8984]) 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.

If omitted on creation, the default is server dependent. For example, servers may choose to always default to null, or may copy the alerts from the default calendar.

* *timeZone*: TimeZoneId|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 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 2.2).

* *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 (see below) 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 per-user properties (see Section 5.3) on all events in the calendar, even if they would not otherwise have permission to modify that event. These properties MUST all be stored per-user, and changes do not affect any other user of the calendar.

  The user may also modify these properties 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 5.1.2), then the user may also add a new Participant to the event with calendarAddress/sendTo properties that are the same as the calendarAddress/sendTo properties 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 5.1.3) 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 the "shareWith" property for this calendar.

* *mayDelete*: Boolean

  The user may delete the calendar itself.

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 (as per Section 3).

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 Section 5.1 of [RFC8620]. 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 Section 5.2 of [RFC8620].

4.3. Calendar/set

This is a standard "/set" method as described in Section 5.3 of [RFC8620] but with the following additional request arguments:

* *onDestroyRemoveEvents*: Boolean (default: false)
  If false, any attempt to destroy a Calendar that still has CalendarEvents in it will be rejected with a calendarHasEvent SetError. If true, any CalendarEvents that were in the Calendar will be removed from it, and if in no other Calendars they will be destroyed. This SHOULD NOT send scheduling messages to participants or create CalendarEventNotification objects.
* *onSuccessSetIsDefault*: Id|null
  If an id is given, and all creates, updates and destroys (if any) succeed without error, the server will try to set this calendar as the default. (For references to Calendar creations, this is equivalent to a creation-reference, so the id will be the creation id prefixed with a "#".)
  If the id is not found, or the change is not permitted by the server for policy reasons, it MUST be ignored and the currently default calendar (if any) will remain as such. No error is returned to the client in this case.
  As per Section 5.3 of [RFC8620], if the default is successfully changed, any changed objects MUST be reported in either the "created" or "updated" argument in the response as appropriate, with the server-set value included.

The "shareWith" property may only be set by users that have the mayAdmin right. 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 following properties may be set by anyone who is subscribed to the calendar and are always stored per-user:

* name

* color
* sortOrder
* isVisible
* timeZone
* includeInAvailability
* defaultAlertsWithoutTime
* defaultAlertsWithTime

The "name", "color", and "timeZone" properties are initially inherited from the owner's copy of the calendar, but if set by a sharee then they get 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", "isVisible", "includeInAvailability", "defaultAlertsWithTime", and "defaultAlertsWithoutTime" properties are initally the default value for each sharee; they are not inherited from the owner.

The following extra SetError type is 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 JSCalendar Event 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 JSCalendar Event 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 5.10). 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".

* *isOrigin*: Boolean (server-set)

  Is this the authoritative source for this event (i.e., does it control scheduling for this event; the event has not been added as a result of an invitation from another calendar system)? This is true if, and only if:

  - the event's "replyTo" property is null; or
  - the account will receive messages sent to at least one of the methods specified in the "replyTo" property of the event.

* *utcStart*: UTCDateTime

  For simple clients that do not 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, the server will convert the UTC date to the event's current time zone and store the local time.

  This property is not included in CalendarEvent/get responses 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 5.10).

* *utcEnd*: UTCDateTime

  The server calculates the end time in UTC from the start/timeZone/duration properties of the event. This property is not included by default and must be requested explicitly. Like utcStart, it 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 four new JSCalendar properties for common use.

5.1.1. calendarAddress

Type: String

Context: Participant

This is a URI as defined by [RFC3986] or any other IANA-registered form for a URI. It is the same as the CAL-ADDRESS value of an ATTENDEE or ORGANIZER in iCalendar ([RFC5545]) it globally identifies a particular participant, even across different events.

5.1.2. mayInviteSelf

Type: Boolean (default: false)

Context: Event, Task

If true, anyone 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.

5.1.3. mayInviteOthers

Type: Boolean (default: false)

Context: Event, Task

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.

5.1.4. hideAttendees

Type: Boolean (default: false)

Context: Event, Task

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 Section 4.2.7 of [RFC8984], 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 Section 6 of [RFC8620].

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, any top-level property registered in the IANA registry as "Is Per-User: yes" (see Section 10.8) MUST be stored per-user. This includes:

* keywords
* color
* freeBusyStatus
* useDefaultAlerts
* alerts

If the user modifies any such properties on a per-occurrence basis for recurring events then again, these MUST also be stored per-user. Sharees initially receive the default value for each of these properties, not whatever value another user may have set.

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 overriden) or update an override for a specific occurrence. To update all occurrences from a specific point onwards, there are therefore two options: split the event, or update the 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 Section 4.1.3 of [RFC8984], 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 participants will see it as two separate changes, and may not understand the connection between the two.

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 Section 5.1 of [RFC8620], with three extra arguments:

* *recurrenceOverridesBefore*: UTCDateTime|null
  If given, only recurrence overrides with a recurrence id before this date (when translated into UTC) will be returned.
* *recurrenceOverridesAfter*: UTCDateTime|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*: TimeZoneId (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 JSCalendar Event 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 JSCalendar Event object in the store are returned, along with the "id", "calendarIds", "isDraft", and "isOrigin" 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 JSCalendar Event and the property is not present on the object in the server's store, the server MUST 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 "recurrenceRules" 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 "hideAttendees" property of a JSCalendar Event object allows the event owner(s) 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.

The "privacy" property of a JSCalendar Event object allows the principal that owns the calendar to override how sharees of the calendar see the event. If set to "private", then when a sharee fetches the event the server MUST only return properties that are:

* the basic time and metadata properties of the JSCalendar Event object as specified in Section 4.4.3 of [RFC8984]; or
* properties that are wholly derived from these permitted properties (i.e., utcStart, utcEnd); or
* Additional CalendarEvent properties not derived from the JSCalendar Event data (i.e., id, baseEventId, calendarIds, isDraft, isOrigin).

If "privacy" is set to "secret", the server MUST behave as though the event does not exist for all users other than the principal that owns the calendar.

5.7. CalendarEvent/changes

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

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 Section 5.3 of [RFC8620], 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 server is the origin of the event) or back to the origin (otherwise). If false, the changes only affect this account and no scheduling messages will be sent.
  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 origin) 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. At time of writing, the most common interoperable protocol for sending scheduling methods between participants on different servers is iMIP [RFC5546]. A future document will provide recommendations of what iMIP messages to send based on the change for best compatibility.

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.

If a property is set to null in a create/update, this is equivalent to omitting/removing the property from the JSCalendar Event object.

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 JSCalendar Event object allows the principal 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 principal that owns the calendar.

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, or a calendar shared by another user). 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 origin of the event (i.e., the event's "isOrigin" property is true), 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 origin of the event it MUST NOT automatically set an "updated" time, as this can break correct processing of scheduling messages.

Clients SHOULD NOT allow users to manually edit anything other than per-user properties when the "isOrigin" property is false, even if the calendar "myRights" allows them to do so. All other properties may be overwritten when a future update arrives to this event from the origin (e.g., via an iTIP REQUEST message). Such updates may be directly applied by the server, or applied at the user's request by a client if it has access to the data through some other means (e.g., the client also has access to the user's email and can parse an iMIP message).

When updating an event, if all of:

* a property has been changed other than "calendarIds", "isDraft", "updated" or a per-user property (see Section 5.3); and
* the server is the origin of the event (the "isOrigin" property is true); 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 "method" property MUST NOT be set. Any attempt to do so is rejected with a standard invalidProperties SetError.

If "utcStart" is set, this is translated into a "start" property using the server's current time zone information. It MUST NOT be set in addition to a "start" property and it cannot be set inside "recurrenceOverrides"; this MUST be rejected with an invalidProperties SetError.

Similarly, the "utcEnd" property is translated into a "duration" property if set. It MUST NOT be set in addition to a "duration" property and it cannot be set inside "recurrenceOverrides"; this MUST be rejected with an invalidProperties SetError.

The server does not automatically reset the "partipationStatus" or "expectReply" properties of a Participant when changing other event details. Clients should either be intelligent about whether the change invalidates previous RSVPs, or ask the user whether to reset 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 3).

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.

The following extra SetError type is defined:

* *noSupportedScheduleMethods*: The server was requested to send scheduling messages, but does not support any of the methods available for at least one of the recipients.

5.8.1. Patching

The JMAP "/set" method allows you to update an object by sending a patch, rather than having to supply the whole object. When doing so, care must be taken if updating a property of a CalendarEvent where the value is itself a PatchObject, e.g. inside "localizations" or "recurrenceOverrides". In particular, you cannot add a property with value null to the CalendarEvent using a direct patch on that property, as this is interpreted instead as a patch to remove the property.

This is more easily understood with an example. Suppose you have a CalendarEvent object like so:

```
{
  "id": "123",
  "title": "FooBar team meeting",
  "start": "2018-01-08T09:00:00",
  "recurrenceRules": [{
   "@type": "RecurrenceRule",
   "frequency": "weekly"
  }],
  "replyTo": {
   "imip": "mailto:6489-4f14-a57f-c1@schedule.example.com"
  },
  "participants": {
   "dG9tQGZvb2Jhci5xlLmNvbQ": {
     "@type": "Participant",
     "name": "Tom",
     "email": "tom@foobar.example.com",
     "calendarAddress": "mailto:6489-4f14-a57f-c1@calendar.example.com",
     "sendTo": {
       "imip": "mailto:6489-4f14-a57f-c1@calendar.example.com"
     },
     "participationStatus": "accepted",
     "roles": {
       "attendee": true
     }
   },
   "em9lQGZvb2GFtcGxlLmNvbQ": {
     "@type": "Participant",
     "name": "Zoe",
     "email": "zoe@foobar.example.com",
     "calendarAddress": "mailto:zoe@foobar.example.com",
     "sendTo": {
       "imip": "mailto:zoe@foobar.example.com",
       "other": "https://foobar.example.com/zoe/itip"
     },
```

```
     "participationStatus": "accepted",
     "roles": {
       "owner": true,
       "attendee": true,
       "chair": true
     }
   },
   "recurrenceOverrides": {
     "2018-03-08T09:00:00": {
       "start": "2018-03-08T10:00:00",
       "participants/dG9tQGZvb2Jhci5xlLmNvbQ/participationStatus":
                                                          "declined"
     }
   }
  }
}
```

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

```
[[ "CalendarEvent/set", {
  "accountId": "ue150411c",
  "update": {
    "123": {
      "recurrenceOverrides/2018-03-08T09:00:00/
          participants~1em9lQGZvb2GFtcGxlLmNvbQ~1participationStatus":
              "declined"
    }
  }
}, "0" ]]
```

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

```
"recurrenceOverrides": {
  "2018-03-08T09:00:00": {
    "start": "2018-03-08T10:00:00",
    "participants/dG9tQGZvb2Jhci5xlLmNvbQ/participationStatus":
                                                    "declined",
    "participants/em9lQGZvb2GFtcGxlLmNvbQ/participationStatus":
                                                    "declined"
  }
}
```

Now if Tom were to change his mind and remove his declined status override (thus meaning he is attending, as inherited from the top-level event), he might remove his patch from the overrides like so:

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

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

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

This would mean remove the "participants/dG9tQGZvb2Jhci5xlLmNvbQ" property at path "recurrenceOverrides" -> "2018-03-08T09:00:00" inside the object; but this doesn't exist. We actually want to add this property and make it map to null. The client must instead send the full object that contains the property mapping to null, like so:

```
[[ "CalendarEvent/set", {
  "accountId": "ue150411c",
  "update": {
    "123": {
      "recurrenceOverrides/2018-03-08T09:00:00": {
        "start": "2018-03-08T10:00:00",
        "participants/em9lQGZvb2GFtcGxlLmNvbQ/participationStatus":
                                                        "declined",
        "participants/dG9tQGZvb2Jhci5xlLmNvbQ": null
      }
    }
  }
}, "0" ]]
```

5.9. CalendarEvent/copy

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

5.10. CalendarEvent/query

This is a standard "/query" method as described in Section 5.5 of [RFC8620], 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*: TimeZoneId
  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*: LocalDateTime|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*: LocalDateTime|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".
* *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 Section 5.6 of [RFC8620].

5.12. CalendarEvent/parse

This method allows the client to parse blobs as iCalendar files [RFC5545] to get CalendarEvent objects. This can be used to parse, display, and import information from iCalendar files without having to implement iCalendar parsing in the client. Server support for this is optional, and indicated via the urn:ietf:params:jmap:calendars:parse capability, as per Section 1.5.3.

The following metadata properties on the CalendarEvent objects will be null if requested:

* id
* baseEventId
* calendarIds
* isDraft
* isOrigin

The "CalendarEvent/parse" method takes the following arguments:

* *accountId*: Id
  The id of the account to use.
* *blobIds*: Id[]
  The ids of the blobs to parse.
* *properties*: String[]
  If supplied, only the properties listed in the array are returned for each CalendarEvent object. If omitted, defaults to all the properties.

The response object contains the following arguments:

* *accountId*: Id
  The id of the account used for the call.
* *parsed*: Id[CalendarEvent[]]|null
  A map of blob ids to parsed CalendarEvent objects representations for each successfully parsed blob, or null if none.
* *notFound*: Id[]|null
  A list of blob ids given that could not be found, or null if none.
* *notParsable*: Id[]|null
  A list of blob ids given that corresponded to blobs that could not be parsed as CalendarEvents, or null if none.

Parsed iCalendars are to be converted into CalendarEvent objects following the process defined in the JSCalendar: Converting from and to iCalendar (https://datatracker.ietf.org/doc/draft-ietf-calext-jscalendar-icalendar) document.

6. Alerts

Alerts may be specified on events as described in Section 4.5 of [RFC8984].

Alerts MUST only be triggered for events in calendars where the user is subscribed.

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 (see Section 4) the event is in, 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 6.2 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 6.3.

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 6.2.
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*: Id
  The account id for the calendar in which the alert triggered.
* *calendarEventId*: Id
  The CalendarEvent id for the alert that triggered. Note, for a recurring event this is the id of the base event, never a synthetic id for a particular instance.
* *uid*: String
  The uid property of the CalendarEvent for the alert that triggered.
* *recurrenceId*: LocalDateTime|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 may 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*: Id
  The id of the CalendarEventNotification.
* *created*: UTCDateTime
  The time this notification was created.
* *changedBy*: Person
  Who made the change. The Person object has the following properties:
  - *name*: String
    The name of the person who made the change.
  - *email*: String|null
    The email of the person who made the change, or null if no email is available.
  - *principalId*: Id|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 an entity outside of this user's organisation, e.g. an iTIP invitation from an external person.
  - *calendarAddress*: String|null

The calendarAddress URI of the person who made the change, if any. This may be set if the change was made due to receving a scheduling message, such as an iTIP message, in addition to changes made by internal users.

* *comment*: String|null
  Comment sent along with the change by the user that made it. (e.g. COMMENT property in an iTIP message), if any.
* *type*: String
  This MUST be one of
  - created
  - updated
  - destroyed
* *calendarEventId*: Id
  The id of the CalendarEvent that this notification is about.
* *isDraft*: Boolean (created/updated only)
  Is this event a draft?
* *event*: JSCalendar Event
  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.

If the change only affects a single instance of a recurring event, the server MAY set the event and eventPatch properties for just that instance; the calendarEventId MUST still be for the base event.

7.3. CalendarEventNotification/get

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

7.4. CalendarEventNotification/changes

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

7.5. CalendarEventNotification/set

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

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 Section 5.5 of [RFC8620].

7.6.1. Filtering

A *FilterCondition* object has the following properties:

* *after*: UTCDateTime|null
  The creation date must be on or after this date to match the condition.
* *before*: UTCDateTime|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 Section 5.6 of [RFC8620].

8. Examples

For brevity, in the following examples only the "methodCalls" property of the Request object, and the "methodResponses" property of the Response object is shown.

8.1. Fetching initial data

A user has authenticated and the client has fetched the JMAP Session object. It finds a single Account with the urn:ietf:params:jmap:calendars capability, with id "a0x9", and wants to display all the calendar information for January 2023 in the Australia/Melbourne time zone. It might make the following request:

```
[
  ["Calendar/get", {
    "accountId": "a0x9"
  }, "0"],
  ["ParticipantIdentity/get", {
    "accountId": "a0x9"
  }, "1"],
  ["CalendarEvent/query", {
    "accountId": "a0x9",
    "timeZone": "Australia/Melbourne",
    "filter": {
      "after": "2023-01-01T00:00:00",
      "before": "2023-02-01T00:00:00"
    }
  }, "2"],
  ["CalendarEvent/get", {
        "accountId": "a0x9",
        "#ids":{
          "resultOf":"3",
          "name":"CalendarEvent/query",
          "path":"/ids"
        }
  }, "3"]
]
```

The server might respond with something like:

```
[
  ["Calendar/get", {
    "accountId": "a0x9",
    "list": [{
       "id": "062adcfa-105d-455c-bc60-6db68b69c3f3",
       "name": "Private",
       "sortOrder": 12,
       "isDefault": false,
       "defaultAlertsWithTime": null,
       ...
     }, {
       "id": "3ddf2ad7-0e0c-4fb5-852d-f0ff56f3c662",
       "name": "Work",
       "sortOrder": 4,
       "isDefault": true,
       "defaultAlertsWithTime": {
         "631BE24C-A3B6-11EC-BF4C-B027680D752E": {
           "@type": "Alert",
           "action": "display",
           "trigger": {
             "@type": "OffsetTrigger",
```

```
              "offset": "-PT1H",
              "relativeTo": "start"
            }
          }
        },
        ...
      }],
      "notFound": [],
      "state": "~506"
    }, "0"],
    ["ParticipantIdentity/get", {
         "accountId": "a0x9",
      "list": [{
        "id": "3",
        "name": "Jane Doe",
        "calendarAddress": "mailto:jane@example.com",
        "sendTo": {
          "imip": "mailto:jane@example.com",
          "other":
                 "https://example.com/uri/for/internal/scheduling"
        },
        "isDefault": true
      }],
      "notFound": [],
      "state": "lgkf:98144:aae"
    }, "1"],
    ["CalendarEvent/query", {
      "accountId": "a0x9",
      "canCalculateChanges": false,
      "position": 0,
      "queryState": "~206",
      "ids": [
        "E-01c9626e-1490-43df-a34f-457021256281",
        "E-07a2b89d-96b6-4920-982a-54fdf0a386ce",
        ...
      ]
    }, "2"],
    ["CalendarEvent/get", {
         "accountId": "a0x9",
         "list": [{
        "id": "E-01c9626e-1490-43df-a34f-457021256281",
        "calendarIds": {
           "3ddf2ad7-0e0c-4fb5-852d-f0ff56f3c662": true
        },
        "title": "Q1 All hands",
        "start": "20230109T10:00:00",
        "duration": "PT1H",
        "timeZone": "Australia/Sydney",
```

```
        ...
      }, ...],
      "notFound": [],
      "state": "$$/413/206"
    }, "3"]
  ]
```

The client now has everything it needs to display that month in full.

8.2. Creating an event

Suppose the user asks the client to create a new event. The client should default to adding it to the "Work" calendar, as this is the default calendar for the user, unless it has information to make a more informed decision. (e.g. The client may have a feature to automatically choose the calendar based on the time of day, and the user indicates the event is at 7pm, so it knows to default to "Private".)

```
[
  ["CalendarEvent/set", {
    "accountId":"a0x9",
    "create":{
      "k559":{
        "uid":"5d5776f6-ff8e-4bfd-ab3e-fe2fe5d4fa91",
        "calendarIds":{
          "3ddf2ad7-0e0c-4fb5-852d-f0ff56f3c662":true
        },
        "title":"Party at Petes",
        "start":"2023-02-03T19:00:00",
        "duration":"PT3H0M0S",
        "timeZone":"Australia/Melbourne",
        "showWithoutTime":false,
        "participants":{
          "1":{
            "@type":"Participant",
            "name": "Jane Doe",
            "calendarAddress": "mailto:jane@example.com",
            "sendTo": {
              "imip": "mailto:jane@example.com",
              "other":
                 "https://example.com/uri/for/internal/scheduling"
            },
            "kind":"individual",
            "roles":{
              "attendee":true,
              "owner":true
            },
```

```
              "participationStatus":"accepted",
              "expectReply":false
            },
            "2":{
              "@type":"Participant",
              "name":"Joe Bloggs",
              "calendarAddress": "mailto:joe@example.com",
              "sendTo":{
                "imip":"mailto:joe@example.com"
              },
              "kind":"individual",
              "roles":{
                "attendee":true
              },
              "participationStatus":"needs-action",
              "expectReply":true
            }
          },
          "mayInviteSelf":false,
          "mayInviteOthers":false,
          "useDefaultAlerts":false,
          "alerts":null
        }
      },
      "sendSchedulingMessages":true
    }, "0"]
  ]
```

As the event has participants, the server sets a "replyTo" property. This server uses a special email address for receiving iMIP RSVPs ([RFC5546]) rather than just receiving them at the owner's regular email address, and also provides a web page for people that don't have calendar clients supporting iMIP. The response may look something like this:

```
NOTE: '\' line wrapping per RFC 8792

[
  ["CalendarEvent/set", {
    "accountId":"a0x9",
    "created":{
      "k559":{
        "id":"E-5d5776f6-ff8e-4bfd-ab3e-fe2fe5d4fa91",
        "isOrigin": true,
        "@type": "Event",
        "created": "20221005T20:42:13Z",
        "updated": "20221005T20:42:13Z",
        "sequence": 1,
        "replyTo": {
          "imip": "3e87-1b18bb5e6b4@itip.example.com",
          "web": "https://cal.example.com/\
                 5d5776f6-ff8e-4bfd-ab3e-fe2fe5d4fa91/?\
                 auth=bfc0-4ba3-9e44"
        }
      }
    },
    ...
  }, "0"]
]
```

8.3. Snoozing an alert

The client is connected to the event source and receives a push:

```
{
        "@type": "CalendarAlert",
        "accountId": "a0x9",
        "calendarEventId": "E-7e93e3ee-4e6e-408a-9adc-cbaf1dbd0a3f",
        "uid": "b6f7e27b-5872-4b52-b457-0242541bb01c",
        "recurrenceId": null,
        "alertId": "7519a951-1e6f-4a6c-b08b-20dd2e5a89cd"
}
```

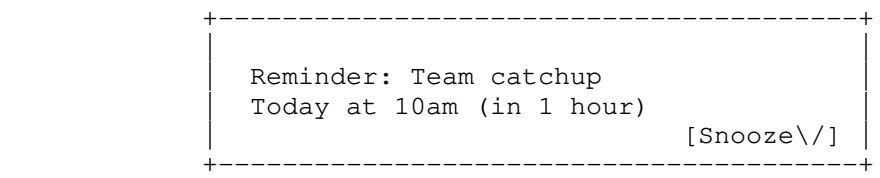
Not finding this event in its local cache, the client fetches the information for this event that it needs to show the alert by making the following request:

```
[
  ["CalendarEvent/get", {
        "accountId": "a0x9",
        "ids":["E-7e93e3ee-4e6e-408a-9adc-cbaf1dbd0a3f"],
        "properties": ["calendarIds", "title", "start",
          "timeZone", "useDefaultAlerts", "alerts"]
  }, "0"]
]
```

In response it receives:

```
[
  ["CalendarEvent/get", {
        "accountId": "a0x9",
        "list": [{
       "id": "E-01c9626e-1490-43df-a34f-457021256281",
       "calendarIds": {
          "3ddf2ad7-0e0c-4fb5-852d-f0ff56f3c662": true
       },
       "title": "Team catchup",
       "start": "20230210T17:00:00",
       "timeZone": "America/New_York",
       "useDefaultAlerts": false,
       "alerts": {
         "7519a951-1e6f-4a6c-b08b-20dd2e5a89cd": {
           "@type": "Alert",
           "action": "display",
           "trigger": {
             "@type": "OffsetTrigger",
             "relativeTo": "start",
             "offset": "-PT1H"
           }
         }
       }
     }],
     "notFound": [],
     "state": "$$/414/208"
  }, "0"]
]
```

The client displays an alert in a platform-appropriate manner. Presuming the user here is in the Australia/Melbourne time zone, this might look something like:



The user snoozes the notification for 30 minutes. The client dismisses the current notification and sends an update to the event to the server:

NOTE: '\' line wrapping per RFC 8792

```
[
  ["CalendarEvent/set", {
        "accountId": "a0x9",
        "update": {
       "E-01c9626e-1490-43df-a34f-457021256281": {
        "alerts/7519a951-1e6f-4a6c-b08b-20dd2e5a89cd\
                /acknowledged": "20230110T23:00:31Z",
        "alerts/86b0-318b8291045f": {
          "@type": "Alert",
          "action": "display",
          "trigger": {
            "@type": "AbsoluteTrigger",
            "when": "20230210T23:30:00Z",
            "relatedTo": {
              "7519a951-1e6f-4a6c-b08b-20dd2e5a89cd": {
                "@type": "Relation",
                "relation": {
                  "parent": true
                }
              }
                }
          }
        }
       }
    }
  }, "0"]
]
```

Any other connected client will receive a push, sync the change and dismiss any duplicate alert. After the snooze time has elapsed, the new alert will trigger.

8.4. Changing the default calendar

The client tries to change the default calendar from "Work" to "Private" (and makes no other change):

```
[
  ["Calendar/set", {
    "accountId": "a0x9",
    "onSuccessSetIsDefault": "062adcfa-105d-455c-bc60-6db68b69c3f3"
  }, "0"]
]
```

The server allows the change, returning the following response:

```
[
  ["Calendar/set", {
    "accountId": "a0x9",
    "updated": {
      "062adcfa-105d-455c-bc60-6db68b69c3f3": {
        "isDefault": true
      },
      "3ddf2ad7-0e0c-4fb5-852d-f0ff56f3c662": {
        "isDefault": false
      }
    }
  }, "0"]
]
```

8.5. Parsing an iCalendar file

The client makes a request to parse the calendar event from a blob id representing an icalendar file:

```
[
  [ "CalendarEvent/parse", {
    "accountId": "a0x9",
    "blobIds": ["Ge682d5d7aad50b3a4f7180a7ed9276476485ea52"]
  }, "c1"]
]
```

The server responds:

```
[[ "CalendarEvent/parse", {
  "accountId": "ue150411c",
  "parsed": {
    "Ge682d5d7aad50b3a4f7180a7ed9276476485ea52": [{
      "@type": "Event",
      "method": "publish",
      "prodId": "-//IETF//datatracker.ietf.org ical agenda//EN",
      "uid": "ietf-119-16811-jmap",
      "sequence": 2,
      "updated": "2024-02-09T22:49:26Z",
      "start": "2024-03-19T13:00:00",
      "duration": "PT2H",
      "timeZone": "Australia/Brisbane",
      "showWithoutTime": false,
      "title": "jmap - JSON Mail Access Protocol",
      "freeBusyStatus": "busy",
      "descriptionContentType": "text/plain",
      "description": "Session II\n\nRemember to sign the blue sheets!",
      "locations": {
          "eec47e7589ce131d6331b10383f89f91f8d4a4ef": {
              "@type": "Location",
              "name": "P3, Brisbane Convention Centre"
          }
      },
      "status": "confirmed"
    }]
  },
  "notFound": null,
  "notParsable": null
}, "c1"]]
```

If the blob id had not been found, the server would have responded:

```
[[ "CalendarEvent/parse", {
  "accountId": "a0x9",
  "notFound": ["Ge682d5d7aad50b3a4f7180a7ed9276476485ea52"]
}, "c1" ]]
```

If the blob id had been found but was not parsable, the server would have responded:

```
[[ "CalendarEvent/parse", {
  "accountId": "a0x9",
  "notParsable": ["Ge682d5d7aad50b3a4f7180a7ed9276476485ea52"]
}, "c1" ]]
```

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; all 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 JSCalendar Event object (see Section 4.4.3 of [RFC8984]) 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 pused to the user's devices, or to send them an email. Servers MUST rate limit the number of alerts sent for any one user. The combination of recurring events with multiple alerts can in particular define unreasonably frequent alerts, leading to denial of service for either the server processing them or the user's devices receiving them.

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

The "email" alert type (see Section 4.5.2 of [RFC8984]) causes an email to be sent when triggered. Clients MUST ignore this alert type; the email is sent only be 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 scheduling messages 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 1.5.1

10.2. 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 1.5.2

10.3. JMAP Data Type Registration for "Calendar"

IANA will register the "Calendar" JMAP Data Type as follows:

Type Name: Calendar

Can reference blobs: no

Can Use for State Change: yes

Capability: urn:ietf:params:jmap:calendars

Specification document: this document

10.4. JMAP Data Type Registration for "CalendarEvent"

IANA will register the "CalendarEvent" JMAP Data Type as follows:

Type Name: CalendarEvent

Can reference blobs: yes

Can Use for State Change: yes

Capability: urn:ietf:params:jmap:calendars

Specification document: this document

10.5. JMAP Data Type Registration for "CalendarEventNotification"

IANA will register the "CalendarEventNotification" JMAP Data Type as follows:

Type Name: CalendarEventNotification

Can reference blobs: no

Can Use for State Change: yes

Capability: urn:ietf:params:jmap:calendars

Specification document: this document

10.6. JMAP Data Type Registration for "ParticipantIdentity"

IANA will register the "ParticipantIdentity" JMAP Data Type as follows:

Type Name: ParticipantIdentity

Can reference blobs: no

Can Use for State Change: yes

Capability: urn:ietf:params:jmap:calendars

Specification document: this document

10.7. JMAP Error Codes Registry

The following subsections register some new error codes in the "JMAP Error Codes" registry, as defined in [RFC8620].

10.7.1. calendarHasEvent

JMAP Error Code: calendarHasEvent

Intended use: common

Change controller: IETF

Reference: This document, Section 4.3

Description: The Calendar has at least one CalendarEvent assigned to it, and the "onDestroyRemoveEvents" argument was false.

10.7.2. noSupportedScheduleMethods

JMAP Error Code: noSupportedScheduleMethods

Intended use: common

Change controller: IETF

Reference: This document, Section 5.8

Description: The server was requested to send scheduling messages, but does not support any of the methods available for at least one of the recipients.

10.7.3. cannotCalculateOccurrences

JMAP Error Code: cannotCalculateOccurrences

Intended use: common

Change controller: IETF

Reference: This document, Section 5.10

Description: The server cannot expand a recurrence required to return the results for the requested query.

10.8. Update to the JSCalendar Properties Registry

IANA will update the "JSCalendar Properties" registry to add a new column called "Is Per-User". The value in this column for each entry MUST be either "yes" or "no", indicating whether each sharee of the object should be able to set their own value for this property without affecting the value for other sharees.

10.8.1. Update to "JSCalendar Properties" Registry Template

An additional field is added to the template:

Is Per-User

10.8.2. Initial values for existing registrations

IANA will set "Is per-user: yes" on the following property registrations:

* keywords
* color
* freeBusyStatus
* useDefaultAlerts
* alerts

All other existing registrations will have "Is per-user: no".

10.9. JSCalendar Property Registrations

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

10.9.1. id

Property Name: id

Property Type: Not applicable

Property Context: Event, Task

Intended Use: Reserved

Is per-user: no

10.9.2. baseEventId

Property Name: baseEventId

Property Type: Not applicable

Property Context: Event, Task

Intended Use: Reserved

Is per-user: no

10.9.3. calendarIds

Property Name: calendarIds

Property Type: Not applicable

Property Context: Event, Task

Intended Use: Reserved

Is per-user: no

10.9.4. isDraft

Property Name: isDraft

Property Type: Not applicable

Property Context: Event, Task

Intended Use: Reserved

Is per-user: no

10.9.5. isOrigin

Property Name: isOrigin

Property Type: Not applicable

Property Context: Event, Task

Intended Use: Reserved

Is per-user: no

10.9.6. utcStart

Property Name: utcStart

Property Type: Not applicable

Property Context: Event, Task

Intended Use: Reserved

Is per-user: no

10.9.7. utcEnd

Property Name: utcEnd

Property Type: Not applicable

Property Context: Event, Task

Intended Use: Reserved

Is per-user: no

10.9.8. calendarAddress

Property Name: calendarAddress

Property Type: String

Property Context: Participant

Reference: This document, Section 5.1.1.

Intended Use: Common

Is per-user: no

10.9.9. mayInviteSelf

Property Name: mayInviteSelf

Property Type: Boolean (default: false)

Property Context: Event, Task

Reference: This document, Section 5.1.2.

Intended Use: Common

Is per-user: no

10.9.10. mayInviteOthers

Property Name: mayInviteOthers

Property Type: Boolean (default: false)

Property Context: Event, Task

Reference: This document, Section 5.1.3.

Intended Use: Common

Is per-user: no

10.9.11. hideAttendees

Property Name: hideAttendees

Property Type: Boolean (default: false)

Property Context: Event, Task

Reference: This document, Section 5.1.4.

Intended Use: Common

Is per-user: no

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>.

[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>.

[RFC5545] Desruisseaux, B., Ed., "Internet Calendaring and Scheduling Core Object Specification (iCalendar)", RFC 5545, DOI 10.17487/RFC5545, September 2009, <https://www.rfc-editor.org/info/rfc5545>.

[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>.

[RFC8792] Watsen, K., Auerswald, E., Farrel, A., and Q. Wu, "Handling Long Lines in Content of Internet-Drafts and RFCs", RFC 8792, DOI 10.17487/RFC8792, June 2020, <https://www.rfc-editor.org/info/rfc8792>.

[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>.

[I-D.ietf-jmap-sharing] Jenkins, N., "JMAP Sharing", Work in Progress, Internet-Draft, draft-ietf-jmap-sharing-08, 6 February 2024, <https://datatracker.ietf.org/api/v1/doc/document/draft-ietf-jmap-sharing/>.

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