JMAP
Internet-Draft
Intended status: Standards Track
Expires: December 17, 2020

N. Jenkins
Fastmail
M. Douglass
Spherical Cow Group

June 15, 2020

JMAP for Calendars
draft-ietf-jmap-calendars-03

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 December 17, 2020.

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. Data Model Overview

A CalendarPrincipal (see Section XXX) represents an individual, team or resource (e.g. a room or projector). The object contains information about the entity being represented, such as a name, description and time zone. A CalendarPrincipal has a 1:1 correspondence with an Account (see [RFC8620], Section 1.6.2) that supports the "urn:ietf:params:jmap:calendars" capability.

Each such Account contains zero or more Calendar objects, which is a named collection of CalendarEvents belonging to the CalendarPrincipal. 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. 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.

A CalendarEvent is a representation of an event or recurring series of events in JSEvent [I-D.ietf-calext-jscalendar] format. Simple clients may ask the server to expand recurrences for them within a specific time period, and optionally convert times into UTC so they do not have to handle time zone conversion. More full-featured clients will want to access the full event information and handle recurrence expansion and time zone conversion locally.

CalendarEventNotification objects keep track of the history of changes made to a calendar by other users, allowing calendar clients to notify the user of changes to their schedule. Similarly, the CalendarShareNotification type notifies the user when their access to another user's calendar is granted or revoked.

1.2. Accounts, Push, and the Session Object

The JMAP Session object (see [RFC8620], Section 2) typically includes an object in the "accounts" property for every account that the user has access to. Calendaring systems may share data between a (potentially very) large number of CalendarPrincipals, most of which the user does not care about day-to-day but may occasionally need to query when scheduling events.

Users can normally subscribe to any calendar to which they have access (see Section XXX). 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.

The Session object MUST only include Accounts where the user is subscribed to at least one Calendar or they have access to some other data type in the account. StateChange events for changes to CalendarEvent data SHOULD only be sent for events in calendars the user has subscribed to and MUST NOT be sent for any Account where the user is not subscribed to at least one calendar.

The server MAY reject the user's attempt to subscribe to some calendars, e.g. those representing resources.

A user may query the set of CalendarPrincipals they have access to with "CalendarPrincipal/query" (see Section XXX). The CalendarPrincipal object may have an "accountId" property that can be used to then fetch calendars and events associated with that principal, subject to appropriate permissions.

1.2.1. UIDs and CalendarEvent Ids

Each CalendarEvent has a "uid" property ([I-D.ietf-calext-jscalendar], Section 4.1.2), which is a globally unique identifier that identifies the same event in different Accounts, or different instances of the same recurring event within an Account.

An Account MUST NOT contain more than one CalendarEvent with the same uid unless all of the CalendarEvent objects have distinct, non-null values for their "recurrenceId" property. (This situation occurs if the principal is added to one or more specific instances of a recurring event without being invited to the whole series.)

Each CalendarEvent also has an id, which is scoped to the JMAP Account and used for referencing it in JMAP methods. There is no necessary link between the uid property and the CalendarEvent's id. CalendarEvents with the same uid in different Accounts MAY have different ids.

1.3. Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP

14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

Type signatures, examples, and property descriptions in this document follow the conventions established in Section 1.1 of [RFC8620]. Data types defined in the core specification are also used in this document.

1.4. The LocalDate Data Type

Where "LocalDate" is given as a type, it means a string in the same format as "Date" (see [RFC8620], Section 1.4), but with the "time-offset" omitted from the end. The interpretation in absolute time depends upon the time zone for the event, which may not be a fixed offset (for example when daylight saving time occurs). For example, "2014-10-30T14:12:00".

1.5. Terminology

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

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

1.6. Addition to the Capabilities Object

The capabilities object is returned as part of the JMAP Session object; see [RFC8620], Section 2. This document defines two additional capability URIs.

1.6.1. urn:ietf:params:jmap:calendars

This represents support for the Calendar, CalendarEvent, and CalendarEventNotification data types and associated API methods. The value of this property in the JMAP Session capabilities property is an empty object.

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

- *accountIdForCalendarPrincipal*: "String|null" The id of an account with the "urn:ietf:params:jmap:calendarprincipals" capability that contains the corresponding CalendarPrincipal object. This may be the same account id. This is null for

single-user systems that do not support the CalendarPrincipal data type.

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

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

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

- *maxParticipantsPerEvent*: "Number|null" The maximum number of participants a single event may have, or null for no limit.

- *mayCreateCalendar*: "Boolean" If true, the user may create a calendar in this account.

1.6.2. urn:ietf:params:jmap:calendarprincipals

Represents support for the CalendarPrincipal and CalendarShareNotification data types and associated API methods. Single user systems do not need this and MAY choose not to support it.

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:

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

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

2. Calendar Principals

A CalendarPrincipal represents an individual, group, schedulable location (e.g. a room), bookable resource (e.g. a projector) or other entity in the calendar system. In a shared calendar environment such as a workplace, a user may have access to a large number of principals.

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

A *CalendarPrincipal* object has the following properties:

- *id*: "Id" The id of the principal.
- *name*: "String" The name of the principal, e.g. "Jane Doe", or "Room 4B".
- *description*: "String|null" A longer description of the principal, for example details about the facilities of a resource, or null if no description available.
- *email*: "String|null" An email address for the principal, or null if no email is available.
- *type*: "String" This MUST be one of the following values:
  - "individual": This represents a single person.
  - "group": This represents a group of people.
  - "resource": This represents some resource, e.g. a projector.
  - "location": This represents a location.
  - "other": This represents some other undefined principal.
- *timeZone*: "String" The time zone for this principal. The value MUST be a time zone id from the IANA Time Zone Database.
- *mayGetAvailability*: "Boolean" May the user call the "CalendarPrincipal/getAvailability" method with this CalendarPrincipal?
- *accountId*: "Id|null" Id of Account with the "urn:ietf:params:jmap:calendars" capability that contains the data for this principal, or null if none (e.g. the CalendarPrincipal is a group just used for permissions management), or the user does not have access to any data in the account (with the exception of free/busy, which is governed by the mayGetAvailability property).
- *account*: "Account|null" The JMAP Account object corresponding to the accountId, null if none.

- *sendTo*: "String[String]|null" If this principal may be added as a participant to an event, this is the map of methods for adding it, in the same format as Participant#sendTo in JSEvent (see [I-D.ietf-calext-jscalendar], Section 4.4.5).

2.1. CalendarPrincipal/get

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

2.2. CalendarPrincipal/changes

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

2.3. CalendarPrincipal/set

This is a standard "/set" method as described in [RFC8620], Section 5.3. However, the user may only update the "timeZone" property of the CalendarPrincipal with the same id as the "currentUserPrincipalId" in the Account capabilities. Any other change MUST be rejected with a "forbidden" SetError.

Managing calendar principals is likely tied to a directory service or some other vendor-specific solution, and occurs out-of-band, or via an additional capability defined elsewhere.

2.4. CalendarPrincipal/query

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

2.4.1. Filtering

A *FilterCondition* object has the following properties:

- *accountIds*: "String[]" A list of account ids. The CalendarPrincipal matches if the value for its accountId property is in this list.

- *email*: "String" Looks for the text in the email property.

- *name*: "String" Looks for the text in the name property.

- *text* "String" Looks for the text in the name, email, and description properties.

- *type*: "String" The type must be exactly as given to match the condition.

- *timeZone*: "String" The timeZone must be exactly as given to match the condition.

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

2.5. CalendarPrincipal/queryChanges

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

2.6. CalendarPrincipal/getAvailability

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 CalendarPrincipal to calculate availability for.

- *utcStart*: "UTCDate" The start time (inclusive) of the period for which to return availability.

- *utcEnd*: "UTCDate" The end time (exclusive) of the period for which to return availability.

- *showDetails*: "Boolean" If true, event details will be returned if the user has permission to view them.

The server will first find all relevant events, expanding any recurring events. Relevant events are ones where all of the following is true:

- The principal is subscribed to the calendar.

- Either the calendar belongs to the principal or the "shareesActAs" property of the calendar is "self".

- The "includeInAvailability" property of the calendar for the principal is "all" or "attending".

- The user has the "mayReadFreeBusy" permission for the calendar.

- The event finishes after the "utcStart" argument and starts before the "utcEnd" argument.
- The event's "privacy" property is not "secret".
- The "freeBusyStatus" property of the event is "busy" (or omitted, as this is the default).
- The "status" property of the event is not "cancelled".
- If the "includeInAvailability" property of the calendar is "attending", then the principal is a participant of the event, and has a "participationStatus" of "accepted" or "tentative".

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

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

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 "CalendarPrincipal/getAvailability" response:

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

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

"tooLarge": The duration between utcStart an utcEnd is longer than the server is willing to calculate availability for.

"rateLimit": Too many availability requests have been made recently and the user is being rate limited. It may work to try again later.

3. Calendars

A Calendar is a named collection of events. All events are associated with one, and only one, calendar.

A *Calendar* object has the following properties:

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

- *role*: "String|null" Denotes the calendar has a special purpose. This MUST be one of the following:

  - "inbox": This is the principal's default calendar; when the principal is invited to an event, this is the calendar to which it will be added by the server. There MUST NOT be more than one calendar with this role in an account.

  - "templates": This calendar holds templates for creating new events. All events in this calendar MUST have the "isDraft"

property set to true. Clients should not show this as a regular calendar to users, but may offer users to create new events by copying one of the events in here.

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

- *description*: "String|null" An optional longer-form description of the calendar, to provide context in shared environments where users need more than just the name.

- *color*: "String" The color to be used when displaying events associated with the calendar. The value MUST be a case-insensitive color name taken from the CSS3 set of names, defined in Section 4.3 of W3C.REC-css3-color-20110607, or a CSS3 RGB color hex value. The color SHOULD have sufficient contrast to be used as text on a white background.

- *sortOrder*: "UnsignedInt" (default: 0) Defines the sort order of calendars when presented in the client's UI, so it is consistent between devices. The number MUST be an integer in the range 0 <= sortOrder < 2^31. A calendar with a lower order should be displayed before a calendar with a higher order in any list of calendars in the client's UI. Calendars with equal order SHOULD be sorted in alphabetical order by name. The sorting should take into account locale-specific character order convention.

- *isSubscribed*: "Boolean" Has the user indicated they wish to see this Calendar in their client? This SHOULD default to false for Calendars in shared accounts the user has access to and true for any new Calendars created by the user themself. If false, the calendar should only be displayed when the user explicitly requests it or to offer it for the user to subscribe to.

- *isVisible*: "Boolean" (default: true) Should the calendar's events be displayed to the user at the moment? Clients MUST ignore this property if isSubscribed is false.

- *includeInAvailability*: "String" (default: all) Should the calendar's events be used as part of availability calculation? This MUST be one of:

  * "all": all events are considered.

  * "attending": events the user is a confirmed or tentative participant of are considered.

    * "none": all events are ignored.

- *defaultAlertsWithTime*: "Alert[]" The alerts to apply for events where showWithoutTime is false that have "useDefaultAlerts" set. See [I-D.ietf-calext-jscalendar], Section 4.5.2 for the definition of an Alert object.

- *defaultAlertsWithoutTime*: "Alert[]" The alerts to apply for events where showWithoutTime is true that have "useDefaultAlerts" set. See [I-D.ietf-calext-jscalendar], Section 4.5.2 for the definition of an Alert object.

- *timeZone*: "String|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 reminders, queries, or availability calculation. The value MUST be a time zone id from the IANA Time Zone Database. If "null", the timeZone of the account's associated CalendarPrincipal will be used. Clients SHOULD use this as the default for new events in this calendar if set.

- *participantIdentities*: "ParticipantIdentity[]|null" (server-set) The identities that represent the user in this calendar. The first item in the array is the default. A *ParticipantIdentity* object has the following properties:

    * *name*: "String" The display name of the participant to use when adding this participant to an event, e.g. "Joe Bloggs".

    * *type*: "String" The method for sending scheduling messages to this identity, e.g. "imip"

    * *uri*: "String" The URI for sending scheduling messages to this identity, e.g. "mailto:foo@example.com"

  The user is an *owner* for an event if the CalendarEvent object has a "participants" property, and one of the Participant objects has both: a) The "owner" role. b) A "sendTo" property that has "type" and "uri" equal to one of the ParticipantIdentity objects returned with the calendar.

- *shareWith*: "Id[CalendarRights]|null" A map of CalendarPrincipal id to rights for principals this calendar is shared with. The pricincipal to which this calendar belongs MUST NOT be in this set. This is null if the user requesting the object does not have the mayAdmin right, or if the calendar is not shared with anyone. May be modified only if the user has the mayAdmin right.

- *shareesActAs*: "String" (immutable; default server-dependent) This MUST be one of:

  - "secretary"

  - "self"

  If "self", sharees act as themselves when using this calendar. If "secretary", they act as the pricincipal to which this calendar belongs (secretary mode). If omitted, the default is server dependent. For example, it may be "self" if creating a calendar in a CalendarPrincipal representing a group, and "secretary" if creating a calendar for an individual. Users may attempt to set this on creation, but the server may reject with an "invalidProperties" error if the value is not permissible.

- *myRights*: "CalendarRights" (server-set) The set of access rights the user has in relation to this Calendar.

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 CalendarPrincipal/getAvailability (see Section XXX).

- *mayReadItems*: "Boolean" The user may fetch the events in this calendar.

- *mayAddItems*: "Boolean" The user may create new events on this calendar or move events to this calendar. For recurring events, they may add an override to add an occurrence, or remove an existing override that is excluding an occurrence.

- *mayUpdatePrivate*: "Boolean" The user may modify the following properties on all events in the calendar. If shareesActAs is "self", these properties MUST all be stored per-user, and changes do not affect any other user of the calendar. If shareesActAs is "secretary", the values are shared between all users.

  - keywords

  - color

  - freeBusyStatus

  - useDefaultAlerts

  - alerts

The user may also modify the above on a per-occurrence basis for recurring events.

- *mayRSVP*: "Boolean" The user may modify the "participationStatus", "participationComment", "expectReply", "scheduleAgent", "scheduleSequence", and "scheduleUpdated" properties of any Participant object that is represented in the "participantIdentities" property of the calendar. The user may also modify the above on a per-occurrence basis for recurring events.

- *mayUpdateOwn*: "Boolean" The user may modify an existing event on this calendar if either they are the owner of the event or the event has no owner.

- *mayUpdateAll*: "Boolean" The user may modify all existing events on this calendar.

- *mayRemoveOwn*: "Boolean" The user may delete an event or move it to a different calendar if either they are the owner of the event or the event has no owner. For recurring events, they may add an override to remove an occurrence.

- *mayRemoveAll*: "Boolean" The user may delete any event or move it to a different calendar. For recurring events, they may add an override to remove an occurrence.

- *mayAdmin*: "Boolean" The user may modify sharing for this calendar.

- *mayDelete*: "Boolean" (server-set) The user may delete the calendar itself. This property MUST be false if the account to which this calendar belongs has the _isReadOnly_ property set to true.

3.1. Per-user properties

The following properties MUST be stored per-user:

- name

- color

- sortOrder

- isVisible

3.2. Calendar/get

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

If mayReadFreeBusy is the only permission the user has, the calendar MUST NOT be returned in Calendar/get and Calendar/query; it must behave as though it did not exist. The data is just used as part of CalendarPrincipal/getAvailability.

3.3. Calendar/changes

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

3.4. Calendar/set

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

o *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 destroyed. This SHOULD NOT send scheduling messages to participants or create CalendarEventNotification objects.

The "role" and "shareWith" properties may only be set by users that have the mayAdmin right. The value is shared across all users, although users without the mayAdmin right cannot see the value.

Users can subscribe or unsubscribe to a calendar by setting the "isSubscribed" property. The server MAY forbid users from subscribing to certain calendars even though they have permission to see them, rejecting the update with a "forbidden" SetError.

The "timeZone", "includeInAvailability", "defaultAlertsWithoutTime" and "defaultAlertsWithTime" properties are stored per-user if the calendar "shareesActAs" value is "self", and may be set by any user who is subscribed to the calendar. Otherwise, these properties are shared, and may only be set by users that have the mayAdmin right.

The following properties may be set by anyone who is subscribed to the calendar and are all stored per-user:

o name

- color
- sortOrder
- isVisible

These properties are initially inherited from the owner's copy of the calendar, but if set by a sharee that user gets their own copy of the property; it does not change for any other principals. If the value of the property in the owner's calendar changes after this, it does not overwrite the sharee's value.

The following extra SetError types are defined:

For "destroy":

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

4. Calendar Share Notifications

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

Clients SHOULD present the list of notifications to the user and allow them to dismiss them. To dismiss a notification you use a standard "/set" call to destroy it.

The server SHOULD create a CalendarShareNotification whenever the user's permissions change on a calendar. It SHOULD NOT create a notification for permission changes to a group principal, even if the user is in the group.

4.1. 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 a calendar if the user subscribes to that calendar.

4.2. Object Properties

The *CalendarShareNotification* object has the following properties:

- *id*: "String" The id of the CalendarShareNotification.
- *created*: "UTCDate" The time this notification was created.
- *changedBy*: "Person" Who made the change.
  - *name*: "String" The name of the person who made the change.
  - *email*: "String|null" The email of the person who made the change, or null if no email is available.
  - *calendarPrincipalId*: "String|null" The id of the CalendarPrincipal corresponding to the person who made the change, or null if no associated princiapal.
- *calendarAccountId*: "String" The id of the account where this Calendar exists.
- *calendarId*: "String" The id of the Calendar that this notification is about.
- *calendarName*: "String" The name of the Calendar at the time the notification was made.
- *oldRights*: "CalendarRights|null" The rights the user had before the change.
- *newRights*: "CalendarRights|null" The rights the user has after the change.

4.3. CalendarShareNotification/get

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

4.4. CalendarShareNotification/changes

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

4.5. CalendarShareNotification/set

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

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

4.6. CalendarShareNotification/query

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

4.6.1. Filtering

A *FilterCondition* object has the following properties:

- *after*: "UTCDate|null" The creation date must be on or after this date to match the condition.
- *before*: "UTCDate|null" The creation date must be before this date to match the condition.

4.6.2. Sorting

The "created" property MUST be supported for sorting.

4.7. CalendarShareNotification/queryChanges

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

5. Calendar Events

A *CalendarEvent* object contains information about an event, or recurring series of events, that takes place at a particular time. It is a JSEvent object, as defined in [I-D.ietf-calext-jscalendar], with the following additional properties:

- *id*: "Id" The id of the CalendarEvent. This property is immutable. The id uniquely identifies a JSEvent with a particular "uid" and "recurrenceId" within a particular account.
- *calendarId*: "Id" The id of the Calendar this event belongs to.
- *isDraft*: "Boolean" If true, this event is to be considered a draft. The server will not send any scheduling messages to participants or send push notifications for alerts. This may only

be set to true upon creation. Once set to false, the value cannot be updated to true. This property MUST NOT appear in "recurrenceOverrides".

- *utcStart*: "UTCDate" For simple clients that do not or cannot implement time zone support. Clients should only use this if also asking the server to expand recurrences, as you cannot accurately expand a recurrence without the original time zone. This property is calculated at fetch time by the server. Time zones are political and they can and do change at any time. Fetching exactly the same property again may return a different results if the time zone data has been updated on the server. Time zone data changes are not considered "updates" to the event. If set, server will convert to the event's current time zone using its current time zone data and store the local time. This is not included by default and must be requested explicitly. Floating events will be interpreted as per calendar's time zone property; or if not set, the the principal's time zone property. Note that it is not possible to accurately calculate the expansion of recurrence rules or recurrence overrides with the utcStart property rather than the local start time. Even simple recurrences such as "repeat weekly" may cross a daylight-savings boundary and end up at a different UTC time. Clients that wish to use "utcStart" are RECOMMENDED to request the server expand recurrences (see Section XXX).

- *utcEnd*: "UTCDate" The server calculates the end time in UTC from the start/timeZone/duration properties of the event. This is not included by default and must be requested explicitly. Like utcStart, this is calculated at fetch time if requested and may change due to time zone data changes.

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 a new JSCalendar property.

5.1.1. hideAttendees

Type: "Boolean" (default: false)

If "true", only the owners of the event may see the full set of participants. Other sharees of the event may only see the owners and themselves.

5.2. Attachments

The Link object, as defined in [I-D.ietf-calext-jscalendar] Section 4.2.7, with a "rel" property equal to "enclosure" is used to represent attachments. Instead of mandating an "href" property, clients may set a "blobId" property instead to reference a blob of binary data in the account, as per [RFC8620] Section 6.

The server MUST translate this to an embedded "data:" URL [RFC2397] when sending the event to a system that cannot access the blob. Servers that support CalDAV access to the same data are recommended to expose these files as managed attachments [?@RFC8607].

5.3. Per-user properties

In shared calendars where "shareesActAs" is "self", the following properties MUST be stored per-user:

- keywords
- color
- freeBusyStatus
- useDefaultAlerts
- alerts

The user may also modify these properties on a per-occurrence basis for recurring events; again, these MUST be stored per-user.

When writing per-user properties, the "updated" property MUST also be stored just for that user. When fetching the "updated" property, the value to return is whichever is later of the per-user updated time or the updated time of the master event.

5.4. Recurring events

Events may recur, in which case they represent multiple occurrences or instances. The data store will either contain a single master event, containing a recurrence rule and/or recurrence overrides; or, a set of individual instances (when invited to specific occurrences only).

The client may ask the server to expand recurrences within a specific time range in "CalendarEvent/query". This will generate synthetic ids representing individual instances in the requested time range. The client can fetch and update the objects using these ids and the

server will make the appropriate changes to the master event. Synthetic ids do not appear in "CalendarEvent/changes" responses; only the ids of events as actually stored on the server.

If the user is invited to specific instances then later added to the master event, "CalendarEvent/changes" will show the ids of all the individual instances being destroyed and the id for the master event being created.

5.5. Updating for "this-and-future"

When editing a recurring event, you can either update the master event (affecting all instances unless overriden) or update an override for a specific occurrence. To update all occurrences from a specific point onwards, there are therefore two options: split the event, or update the master and override all occurrences before the split point back to their original values.

5.5.1. Splitting an event

If the event is not scheduled (has no participants), the simplest thing to do is to duplicate the event, modifying the recurrence rules of the original so it finishes before the split point, and the duplicate so it starts at the split point. As per JSCalendar [I-D.ietf-calext-jscalendar] Section 4.1.3, a "next" and "first" relation MUST be set on the new objects respectively.

Splitting an event however is problematic in the case of a scheduled event, because the iTIP messages generated make it appear like two unrelated changes, which can be confusing.

5.5.2. Updating the master and overriding previous

For scheduled events, a better approach is to avoid splitting and instead update the master event with the new property value for "this and future", then create overrides for all occurrences before the split point to restore the property to its previous value. Indeed, this may be the only option the user has permission to do if not an owner of the event.

Clients may choose to skip creating the overrides if the old data is not important, for example if the "alerts" property is being updated, it is probably not important to create overrides for events in the past with the alerts that have already fired.

5.6. CalendarEvent/get

This is a standard "/get" method as described in [RFC8620], Section 5.1, with three extra arguments:

- *recurrenceOverridesBefore*: "UTCDate|null" If given, only recurrence overrides with a recurrence id before this date (when translated into UTC) will be returned.

- *recurrenceOverridesAfter*: "UTCDate|null" If given, only recurrence overrides with a recurrence id on or after this date (when translated into UTC) will be returned.

- *reduceParticipants*: "Boolean" (default: false) If true, only participants with the "owner" role or corresponding to the user's participant identities will be returned in the "participants" property of the master event and any recurrence overrides. If false, all participants will be returned.

A CalendarEvent object is a JSEvent object so may have arbitrary properties. If the client makes a "CalendarEvent/get" call with a null or omitted "properties" argument, all properties defined on the JSEvent object in the store are returned, along with the "id", "calendarId", and "isDraft" properties. The "utcStart" and "utcEnd" computed properties are only returned if explicitly requested. If either are requested, the "recurrenceOverrides" property MUST NOT be requested (recurrence overrides cannot be interpreted accurately with just the UTC times).

If specific properties are requested from the JSEvent and the property is not present on the object in the server's store, the server SHOULD return the default value if known for that property.

A requested id may represent a single instance of a recurring event if the client asked the server to expand recurrences in "CalendarEvent/query". In such a case, the server will resolve any overrides and set the appropriate "start" and "recurrenceId" properties on the CalendarEvent object returned to the client. The "recurrenceRule" and "recurrenceOverrides" properties MUST be returned as null if requested for such an event.

An event with the same uid/recurrenceId may appear in different accounts. Clients may coalesce the view of such events, but must be aware that the data may be different in the different accounts due to per-user properties, difference in permissions etc.

The "privacy" property of a JSEvent object allows the owner to override how sharees of the calendar see the event. If this is set

to "private", when a sharee fetches the event the server MUST only return the basic time and metadata properties of the JSEvent object as specified in [I-D.ietf-calext-jscalendar], Section 4.4.3. If set to "secret", the server MUST behave as though the event does not exist for all users other than the owner.

This "hideAttendees" property of a JSEvent object allows the owner to reduce the visibility of sharees into the set of participants. If this is "true", when a non-owner sharee fetches the event, the server MUST only return participants with the "owner" role or corresponding to the user's participant identities.

5.7. CalendarEvent/changes

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

5.8. CalendarEvent/set

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

- *sendSchedulingMessages*: "Boolean" (default: true) If true then any changes to scheduled events will be sent to all the participants (if the user is an owner of the event) or back to the owners (otherwise). If false, the changes only affect this calendar and no scheduling messages will be sent.

For recurring events, an id may represent the master event or a specific instance. When the id for a specific instance is given, the server MUST process an update as an update to the recurrence override for that instance on the master event, and a destroy as removing just that instance.

Clients MUST NOT send an update/destroy to both the master event and a specific instance in a single "/set" request; the result of this is undefined.

Servers MUST enforce the user's permissions as returned in the "myRights" property of the Calendar object and reject changes with a "forbidden" SetError if not allowed.

The "privacy" property MUST NOT be set to anything other than "public" (the default) for events in a calendar that does not belong to the user (e.g. a shared team calendar). The server MUST reject this with an "invalidProperties" SetError.

The server MUST reject attempts to add events with a "participants" property where none of the participants correspond to one of the calendar's participant identities with a "forbidden" SetError.

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

- @type
- uid
- created
- updated

When modifying the event, the server MUST set the following properties if the server is the source of the event (see Section XXX) and the property is not explicitly set in the update:

- updated: set to the current time.
- sequence: increment by one, unless only per-user properties (see Section XXX) were changed, in which case do not modify.

The "created" property MUST NOT be updated after creation. The "sequence" property MUST NOT be set to a lower number than its current value. The "method" property MUST NOT be set. Any attempt to do these is rejected with a standard "invalidProperties" SetError.

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

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

The server does not automatically reset the "partipationStatus" or "expectReply" properties of a Participant when changing other event details. Clients should either be intelligent about whether the change necessitates resending RSVP requests, or ask the user whether to send them.

The server MAY enforce that all events have an owner, for example in team calendars. If the user tries to create an event without

participants in such a calendar, the server MUST automatically add a participant with the "owner" role corresponding to one of the user's "participantIdentities" for the calendar.

When creating an event with participants, or adding participants to an event that previously did not have participants, the server MUST set the "replyTo" property of the event if not present. Clients SHOULD NOT set the replyTo property for events when the user adds participants; the server is better positioned to add all the methods it supports to receive replies.

5.8.1. Patching

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

```
{
  "id": "123",
  "title": "FooBar team meeting",
  "start": "2018-01-08T09:00:00",
  "recurrenceRules": [{
   "@type": "RecurrenceRule",
   "frequency": "weekly"
  }],
  "replyTo": {
   "imip": "mailto:6489-4f14-a57f-c1@schedule.example.com"
  },
  "participants": {
   "dG9tQGZvb2Jhci5xlLmNvbQ": {
     "@type": "Participant",
     "name": "Tom",
     "email": "tom@foobar.example.com",
     "sendTo": {
       "imip": "mailto:6489-4f14-a57f-c1@calendar.example.com"
     },
     "participationStatus": "accepted",
     "roles": {
       "attendee": true
     }
   },
   "em9lQGZvb2GFtcGxlLmNvbQ": {
     "@type": "Participant",
     "name": "Zoe",
     "email": "zoe@foobar.example.com",
     "sendTo": {
       "imip": "mailto:zoe@foobar.example.com"
     },
     "participationStatus": "accepted",
     "roles": {
       "owner": true,
       "attendee": true,
       "chair": true
     }
   },
   "recurrenceOverrides": {
     "2018-03-08T09:00:00": {
       "start": "2018-03-08T10:00:00",
       "participants/dG9tQGZvb2Jhci5xlLmNvbQ/participationStatus":
                                                              "declined"
     }
   }
  }
}
```

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

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

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

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

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

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

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

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

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

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

5.8.2. Sending invitations and responses

Unless "sendSchedulingMessages" is false, the server MUST send appropriate iTIP [RFC5546] scheduling messages after successfuly creating, updating or destroying a calendar event.

When determining which scheduling messages to send, the server must first establish whether it is the _source_ of the event. The server is the source if it will receive messages sent to any of the methods specified in the "replyTo" property of the event.

Messages are only sent to participants with a "scheduleAgent" property set to "server" or omitted. If the effective "scheduleAgent" property is changed:

- to "server" from something else: send messages to this participant as though the event had just been created.

- from "server" to something else: send messages to this participant as though the event had just been destroyed.

- any other change: do not send any messages to this participant.

The server may send the scheduling message via any of the methods defined on the sendTo property of a participant (if the server is the source) or the replyTo property of the event (otherwise) that it supports. If no supported methods are available, the server MUST reject the change with a "noSupportedScheduleMethods" SetError.

If the server is the source of the event it MUST NOT send messages to any participant corresponding to the participantIdentities of the calendar it is in.

If sending via iMIP [RFC6047], the server MAY choose to only send updates it deems "essential" to avoid flooding the recipient's email with changes they do not care about. For example, changes to the participationStatus of another participant, or changes to events solely in the past may be omitted.

5.8.2.1. REQUEST

When the server is the source for the event, a REQUEST message ([RFC5546], Section 3.2.2) is sent to all current participants if:

- The event is being created.

- Any non per-user property (see Section XXX) is updated on the event (including adding/removing participants), except if just modifying the recurrenceOverrides such that CANCEL messages are generated (see the next section).

Note, if the only change is adding an additional instance (not generated by the event's recurrence rule) to the recurrenceOverrides, this MAY be handled via sending an ADD message ([RFC5546], Section 3.2.4) for the single instance rather than a REQUEST message for the master. However, for interoperability reasons this is not recommended due to poor support in the wild for this type of message.

The server MUST ensure participants are only sent information about recurrence instances they are added to when sending scheduling messages for recurring events. If the participant is not invited to the master recurring event but only individual instances, scheduling messages MUST be sent for just those expanded occurrences individually. If a participant is invited to a recurring event, but removed via a recurrence override from a particular instance, any scheduling messages to this participant MUST return the instance as

"excluded" (if it matches a recurrence rule for the event) or omit the instance entirely (otherwise).

If the event's "hideAttendees" property is set to "true", the recipient MUST be the only attendee in the message; all others are omitted.

5.8.2.2. CANCEL

When the server is the source for the event, a CANCEL message ([RFC5546], Section 3.2.5) is sent if:

- A participant is removed from either the master event or a single instance (the message is only sent to this participant; remaining participants will get a REQUEST, as described above).
- The event is destroyed.
- An exclusion is added to recurrenceOverrides to remove an instance generated by the event's recurrence rule.
- An additional instance (not generated by the event's recurrence rule) is removed from the recurrenceOverrides.

In each of the latter 3 cases, the message is sent to all participants.

5.8.2.3. REPLY

When the server is _not_ the source for the event, a REPLY message ([RFC5546], Section 3.2.3) is sent for any participant corresponding to the participantIdentities of the calendar it is in if:

- The "participationStatus" property of the participant is changed.
- The event is destroyed and the participationStatus was not "needs-action".
- The event is created and the participationStatus is not "needs-action".
- An exclusion is added to recurrenceOverrides to remove an instance generated by the event's recurrence rule.
- An exclusion is removed from recurrenceOverrides (this is presumed to be the client undoing the deletion of a single instance).

- An instance not generated by the event's recurrence rule is removed from the recurrenceOverrides.

- An instance not generated by the event's recurrence rule is added to the recurrenceOverrides (this is presumed to be the client undoing the deletion of a single instance).

A reply is not sent when deleting an event where the current status is "needs-action" as if a junk calendar event gets added by an automated system, the user MUST be able to delete the event without sending a reply.

5.9. CalendarEvent/copy

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

5.10. CalendarEvent/query

This is a standard "/query" method as described in [RFC8620], Section 5.5, with two extra arguments:

- *expandRecurrences*: "Boolean" (default: false) If true, the server will expand any recurring event. If true, the filter MUST be just a FilterCondition (not a FilterOperator) and MUST include both a before and after property. This ensures the server is not asked to return an infinite number of results.

- *timeZone*: "String" The time zone for before/after filter conditions (default: "Etc/UTC")

If expandRecurrences is true, a separate id will be returned for each instance of a recurring event that matches the query. This synthetic id is opaque to the client, but allows the server to resolve the id + recurrence id for "/get" and "/set" operations. Otherwise, a single id will be returned for matching recurring events that represents the entire event.

There is no necessary correspondence between the ids of different instances of the same expanded event.

The following additional error may be returned instead of the "CalendarEvent/query" response:

"cannotCalculateOccurrences": the server cannot expand a recurrence required to return the results for this query.

5.10.1. Filtering

A *FilterCondition* object has the following properties:

- *inCalendars*: "Id[]|null" A list of calendar ids. An event must be in ANY of these calendars to match the condition.
- *after*: "LocalDate|null" The end of the event, or any recurrence of the event, in the time zone given as the timeZone argument, must be after this date to match the condition.
- *before*: "LocalDate|null" The start of the event, or any recurrence of the event, in the time zone given as the timeZone argument, must be before this date to match the condition.
- *text*: "String|null" Looks for the text in the _title_, _description_, _locations_ (matching name/description), _participants_ (matching name/email) and any other textual properties of the event or any recurrence of the event.
- *title*: "String|null" Looks for the text in the _title_ property of the event, or the overridden _title_ property of a recurrence.
- *description*: "String|null" Looks for the text in the _description_ property of the event, or the overridden _description_ property of a recurrence.
- *location*: "String|null" Looks for the text in the _locations_ property of the event (matching name/description of a location), or the overridden _locations_ property of a recurrence.
- *owner*: "String|null" Looks for the text in the name or email fields of a participant in the _participants_ property of the event, or the overridden _participants_ property of a recurrence, where the participant has a role of "owner".
- *attendee*: "String|null" Looks for the text in the name or email fields of a participant in the _participants_ property of the event, or the overridden _participants_ property of a recurrence, where the participant has a role of "attendee".
- *participationStatus*: Must match. If owner/attendee condition, status must be of that participant. Otherwise any.
- *uid*: "String" The uid of the event is exactly the given string.

If expandRecurrences is true, all conditions must match against the same instance of a recurring event for the instance to match. If

expandRecurrences is false, all conditions must match, but they may each match any instance of the event.

If zero properties are specified on the FilterCondition, the condition MUST always evaluate to "true". If multiple properties are specified, ALL must apply for the condition to be "true" (it is equivalent to splitting the object into one-property conditions and making them all the child of an AND filter operator).

The exact semantics for matching "String" fields is *deliberately not defined* to allow for flexibility in indexing implementation, subject to the following:

- Text SHOULD be matched in a case-insensitive manner.
- Text contained in either (but matched) single or double quotes SHOULD be treated as a *phrase search*, that is a match is required for that exact sequence of words, excluding the surrounding quotation marks. Use "\"", "\'" and "\\" to match a literal """, "'" and "\" respectively in a phrase.
- Outside of a phrase, white-space SHOULD be treated as dividing separate tokens that may be searched for separately in the event, but MUST all be present for the event to match the filter.
- Tokens MAY be matched on a whole-word basis using stemming (so for example a text search for "bus" would match "buses" but not "business").

5.10.2. Sorting

The following properties MUST be supported for sorting:

- start
- uid
- recurrenceId

The following properties SHOULD be supported for sorting:

- created
- updated

5.11. CalendarEvent/queryChanges

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

5.12. Examples

TODO: Add example of how to get event by uid: query uid=foo and backref. Return multiple with recurrenceId set (user invited to specific instances of recurring event).

6. Alerts

Alerts may be specified on events as described in [I-D.ietf-calext-jscalendar], Section 4.5. If the "useDefaultAlerts" property is true, the alerts are taken from the Calendar "defaultAlertsWithTime" or "defaultAlertsWithoutTime" property, as described in Section XXX. Otherwise, the alerts are taken from the "alerts" property of the CalendarEvent.

Alerts MUST only be triggered for events in calendars where the user is subscribed and either the user owns the calendar or the calendar's "shareesActAs" property is "self".

When an alert with an "email" action is triggered, the server MUST send an email to the user to notify them of the event. The contents of the email is implementation specific. Clients MUST NOT perform an action for these alerts.

When an alert with a "display" action is triggered, clients SHOULD display an alert in a platform-appropriate manner to the user to remind them of the event. Clients with a full offline cache of events may choose to calculate when alerts should trigger locally. Alternatively, they can subscribe to push events from the server.

6.1. Push events

Servers that support the "urn:ietf:params:jmap:calendars" capability MUST support registering for the pseudo-type "CalendarAlert" in push subscriptions and event source connections, as described in [RFC8620], Sections 7.2 and 7.3.

If requested, a CalendarAlert notification will be pushed whenever an alert is triggered for the user. For Event Source connections, this notification is pushed as an event called "calendaralert".

A *CalendarAlert* object has the following properties:

- *@type*: "String" This MUST be the string "CalendarAlert".
- *accountId*: "String" The account id for the calendar in which the alert triggered.
- *calendarEventId*: "String" The CalendarEvent id for the alert that triggered.
- *uid*: "String" The uid property of the CalendarEvent for the alert that triggered.
- *recurrenceId*: "String|null" The recurrenceId for the instance of the event for which this alert is being triggered, or "null" if the event is not recurring.
- *alertId*: "String" The id for the alert that triggered.

6.2. Acknowledging an alert

To dismiss an alert, clients set the "acknowledged" property of the Alert object to the current date-time. When other clients fetch the CalendarEvent with the updated Alert they SHOULD automatically dismiss or suppress duplicate alerts (alerts with the same alert id that triggered on or before this date-time).

Setting the "acknowledged" property MUST NOT create a new recurrence override. For a recurring calendar object, the "acknowledged" property of the parent object MUST be updated, unless the alert is already overridden in the "recurrenceOverrides" property.

6.3. Snoozing an alert

Users may wish to dismiss an alert temporarily and have it come back after a specific period of time. To do this, clients MUST:

1. Acknowledge the alert as described in Section XXX.
2. Add a new alert 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 being snoozed.

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.

7. Calendar Event Notifications

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

Notifications are only created by the server; users cannot create them directly. Clients SHOULD present the list of notifications to the user and allow them to dismiss them. To dismiss a notification you use a standard "/set" call to destroy it.

The server SHOULD create a CalendarEventNotification whenever an event is added, updated or destroyed by another user or due to receiving an iTIP [RFC5546] or other scheduling message in a calendar this user is subscribed to. The server SHOULD NOT create notifications for events implicitly deleted due to the containing calendar being deleted.

7.1. Auto-deletion of Notifications

The server MAY limit the maximum number of notifications it will store for a user. When the limit is reached, any new notification will cause the previously oldest notification to be automatically deleted.

The server MAY coalesce events if appropriate, or remove events that it deems are no longer relevant or after a certain period of time. The server SHOULD automatically destroy a notification about an event if the user updates or destroys that event (e.g. if the user sends an RSVP for the event).

7.2. Object Properties

The *CalendarEventNotification* object has the following properties:

- *id*: "String" The id of the CalendarEventNotification.
- *created*: "UTCDate" The time this notification was created.
- *changedBy*: "Person" Who made the change.
  - *name*: "String" The name of the person who made the change.
  - *email*: "String" The email of the person who made the change, or null if no email is available.

    * *calendarPrincipalId*: "String|null" The id of the calendar principal corresponding to the person who made the change, if any. This will be null if the change was due to receving an iTIP message.

- *comment*: "String|null" Comment sent along with the change by the user that made it. (e.g. COMMENT property in an iTIP message).

- *type*: "String" This MUST be one of

    * created

    * updated

    * destroyed

- *calendarEventId*: "String" The id of the CalendarEvent that this notification is about.

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

- *event*: "JSEvent" The data before the change (if updated or destroyed), or the data after creation (if created).

- *eventPatch*: "PatchObject" (updated only) A patch encoding the change between the data in the event property, and the data after the update.

To reduce data, if the change only affects a single instance of a recurring event, the server MAY set the event and eventPatch properties for the instance; the calendarEventId MUST still be for the master event.

7.3. CalendarEventNotification/get

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

7.4. CalendarEventNotification/changes

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

7.5. CalendarEventNotification/set

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

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

7.6. CalendarEventNotification/query

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

7.6.1. Filtering

A *FilterCondition* object has the following properties:

- *after*: "UTCDate|null" The creation date must be on or after this date to match the condition.
- *before*: "UTCDate|null" The creation date must be before this date to match the condition.
- *type*: "String" The type property must be the same to match the condition.
- *calendarEventIds*: "Id[]|null" A list of event ids. The calendarEventId property of the notification must be in this list to match the condition.

7.6.2. Sorting

The "created" property MUST be supported for sorting.

7.7. CalendarEventNotification/queryChanges

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

8. Security Considerations

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

8.1. Denial-of-service Expanding Recurrences

Recurrence rules can be crafted to occur as frequently as every second. Servers MUST be careful to not allow resources to be exhausted when expanding. Equally, rules can be generated that never

create any occurrences at all. Servers MUST be careful to limit the work spent iterating in search of the next occurrence.

8.2. Privacy

TODO.

9. IANA Considerations

9.1. JMAP Capability Registration for "calendars"

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

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

Specification document: this document

Intended use: common

Change Controller: IETF

Security and privacy considerations: this document, Section XXX

9.2. JSCalendar Property Registrations

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

9.2.1. id

Property Name: id

Property Type: "Id"

Property Context: JSEvent, JSTask

Intended Use: Reserved

9.2.2. calendarId

Property Name: calendarId

Property Type: "Id"

Property Context: JSEvent, JSTask

Intended Use: Reserved

9.2.3. isDraft

Property Name: isDraft

Property Type: "Boolean"

Property Context: JSEvent, JSTask

Intended Use: Reserved

9.2.4. utcStart

Property Name: utcStart

Property Type: "UTCDateTime"

Property Context: JSEvent, JSTask

Intended Use: Reserved

9.2.5. utcEnd

Property Name: utcEnd

Property Type: "UTCDateTime"

Property Context: JSEvent, JSTask

Intended Use: Reserved

9.2.6. hideAttendees

Property Name: hideAttendees

Property Type: "Boolean" (default: false)

Property Context: JSEvent, JSTask

Reference: This document, Section XXX.

Intended Use: Common

10. References

10.1. Normative References

[I-D.ietf-calext-jscalendar]
Jenkins, N. and R. Stepanek, "JSCalendar: A JSON representation of calendar data", draft-ietf-calext-jscalendar-26 (work in progress), March 2020.

[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>.

[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>.

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

[RFC8620] Jenkins, N. and C. Newman, "The JSON Meta Application Protocol (JMAP)", RFC 8620, DOI 10.17487/RFC8620, July 2019, <https://www.rfc-editor.org/info/rfc8620>.

10.2. 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>.

[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
Fastmail
PO Box 234, Collins St West
Melbourne VIC 8007
Australia

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

Michael Douglass
Spherical Cow Group
226 3rd Street
Troy NY 12180
United States of America

Email: mdouglass@sphericalcowgroup.com
URI: http://sphericalcowgroup.com