CalConnect TC

JSCalendar: A JSON representation of calendar data

Working Draft Standard

Warning for drafts

This document is not a CalConnect Standard. It is distributed for review and comment, and is subject to change without notice and may not be referred to as a Standard. Recipients of this draft are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

Recipients of this draft are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

The Calendaring and Scheduling Consortium, Inc. 2020

The Calendaring and Scheduling Consortium, Inc.

4390 Chaffin Lane
McKinleyville
California 95519
United States of America

copyright@calconnect.org
www.calconnect.org

Contents

Abstract

This specification defines a data model and JSON representation of calendar data that can be used for storage and data exchange in a calendaring and scheduling environment. It aims to be an alternative and, over time, successor to the widely deployed iCalendar data format, and to be unambiguous, extendable, and simple to process. In contrast to the jCal format, which is also JSON-based, JSCalendar is not a direct mapping from iCalendar, but defines the data model independently and expands semantics where appropriate.

Introduction

This document defines a data model for calendar event and task objects, or groups of such objects, in electronic calendar applications and systems. The format aims to be unambiguous, extendable and simple to process.

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

— The attributes of the calendar entry represented must be described as simple key-value pairs. Simple events are simple to represent; complex events can be modelled accurately.
— Wherever possible, there should be only one way to express the desired semantics, reducing complexity.
— The data model should avoid ambiguities, which often lead to interoperability issues between implementations.
— The data model should be generally compatible with the iCalendar data format IETF RFC 5545 IETF RFC 7986 and extensions, but the specification should add new attributes where the iCalendar format currently lacks expressivity, and drop seldom-used, obsolete, or redundant properties. This means translation with no loss of semantics should be easy with most common iCalendar files.
— Extensions, such as new properties and components, should not require updates to this document.

The representation of this data model is defined in the I-JSON format IETF RFC 7493, which is a strict subset of the JavaScript Object Notation (JSON) Data Interchange Format IETF RFC 8259. Using JSON is mostly a pragmatic choice: its widespread use makes JSCalendar easier to adopt, and the ready availability of production-ready JSON implementations eliminates a whole category of parser-related interoperability issues, which iCalendar has often suffered from.

Motivation and Relation to iCalendar and jCal

The iCalendar data format IETF RFC 5545, a widely deployed interchange format for calendaring and scheduling data, has served calendaring vendors for a long while, but contains some ambiguities and pitfalls that can not be overcome without backward-incompatible changes.

Sources of implementation errors include the following:

— iCalendar defines various formats for local times, UTC time, and dates.
— iCalendar requires custom time zone definitions within a single calendar component.
— iCalendar's definition of recurrence rules is ambiguous and has resulted in differing understandings even between experienced calendar developers.
— The iCalendar format itself causes interoperability issues due to misuse of CRLF-terminated strings, line continuations, and subtle differences among iCalendar parsers.

In recent years, many new products and services have appeared that wish to use a JSON representation of calendar data within their APIs. The JSON format for iCalendar data, jCal IETF RFC 7265, is a direct mapping between iCalendar and JSON. In its effort to represent full iCalendar semantics, it inherits all the same pitfalls and uses a complicated JSON structure.

As a consequence, since the standardization of jCal, the majority of implementations and service providers either kept using iCalendar, or came up with their own proprietary JSON representations, which are incompatible with each other and often suffer from common pitfalls, such as storing event start times in UTC (which become incorrect if the timezone's rules change in the future). JSCalendar meets the demand for JSON-formatted calendar data that is free of such known problems and provides a standard representation as an alternative to the proprietary formats.

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 IETF RFC 2119 IETF RFC 8174 when, and only when, they appear in all capitals, as shown here.

The underlying format used for this specification is JSON. Consequently, the terms "object" and "array" as well as the four primitive types (strings, numbers, booleans, and null) are to be interpreted as described in IETF RFC 8259, Section 1.

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

Type Signatures

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

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

Other types may also be given, with their representations defined elsewhere in this document.

Data Types

In addition to the standard JSON data types, the following data types are used in this specification:

Id

Where `Id` is given as a data type, it means a `String` of at least 1 and a maximum of 255 octets in size, and it MUST only contain characters from the "URL and Filename Safe" base64url alphabet, as defined in IETF RFC 4648, Section 5, excluding the pad character (=). This means the allowed characters are the ASCII alphanumeric characters (`A-Za-z0-9`), hyphen (`-`), and underscore (_).

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

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

Int

Where `Int` is given as a data type, it means an integer in the range -2^53+1 ⇐ value ⇐ 2^53-1, the safe range for integers stored in a floating-point double, represented as a JSON `Number`.

UnsignedInt

Where `UnsignedInt` is given as a data type, it means an integer in the range 0 ⇐ value ⇐ 2^53-1, represented as a JSON `Number`.

UTCDateTime

This is a string in IETF RFC 3339 `date-time` format, with the further restrictions that any letters MUST be in uppercase, and the time offset MUST be the character `Z`. Fractional second values MUST NOT be included unless non-zero and MUST NOT have trailing zeros, to ensure there is only a single representation for each date-time.

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

LocalDateTime

This is a date-time string with no time zone/offset information. It is otherwise in the same format as UTCDateTime, including fractional seconds. For example `2006-01-02T15:04:05` and `2006-01-02T15:04:05.003` are both valid. The time zone to associate with the LocalDateTime comes from the `timeZone` property of the JSCalendar object (see Clause 6.8.1). If no time zone is specified, the LocalDateTime is `floating`. Floating date-times are not tied to any specific time zone. Instead, they occur in each time zone at the given wall-clock time (as opposed to the same instant point in time).

A time zone may have a period of discontinuity, for example a change from standard time to daylight-savings time. When converting local date-times that fall in the discontinuity to UTC, the offset before the transition MUST be used.

For example, in the America/Los_Angeles time zone, the date-time 2020-11-01T01:30:00 occurs twice: before the DST transition with a UTC offset of -07:00, and again after the transition with an offset of -08:00. When converting to UTC, we therefore use the offset before the transition (-07:00) and so it becomes 2020-11-01T08:30:00Z.

Similarly, in the Australia/Melbourne time zone, the date-time 2020-10-04T02:30:00 does not exist: the clocks are moved forward one hour for DST on that day at 02:00. However, such a value may appear during calculations (see duration semantics in Duration), or due to a change in time zone rules (so it was valid when the event was first created). Again, it is interpreted as though the offset before the transition is in effect (+10:00), therefore when converted to UTC we get 2020-10-03T16:30:00Z.

Duration

Where Duration is given as a type, it means a length of time represented by a subset of ISO 8601 duration format, as specified by the following ABNF IETF RFC 5234:

```
dur-secfrac = "." 1*DIGIT
dur-second  = 1*DIGIT [dur-secfrac] "S"
dur-minute  = 1*DIGIT "M" [dur-second]
dur-hour    = 1*DIGIT "H" [dur-minute]
dur-time    = "T" (dur-hour / dur-minute / dur-second)
dur-day     = 1*DIGIT "D"
dur-week    = 1*DIGIT "W"
dur-cal     = (dur-week [dur-day] / dur-day)

duration    = "P" (dur-cal [dur-time] / dur-time)
```

In addition, the duration MUST NOT include fractional second values unless the fraction is non-zero. Fractional second values MUST NOT have trailing zeros, to ensure there is only a single representation for each duration.

A duration specifies an abstract number of weeks, days, hours, minutes, and/or seconds. A duration specified using weeks or days does not always correspond to an exact multiple of 24 hours. The number of hours/minutes/seconds may vary if it overlaps a period of discontinuity in the event's time zone, for example a change from standard time to daylight-savings time. Leap seconds MUST NOT be considered when adding or subtracting a duration to/from a LocalDateTime.

To add a duration to a LocalDateTime:

1) Add any week or day components of the duration to the date. A week is always the same as 7 days.
2) If a time zone applies to the LocalDateTime, convert it to a UTCDateTime following the semantics in LocalDateTime.
3) Add any hour, minute or second components of the duration (in absolute time).
4) Convert the resulting UTCDateTime back to a LocalDateTime in the time zone that applies.

To subtract a duration from a LocalDateTime, the steps apply in reverse:

1) If a time zone applies to the LocalDateTime, convert it to UTC following the semantics in LocalDateTime.
2) Subtract any hour, minute or second components of the duration (in absolute time).
3) Convert the resulting UTCDateTime back to LocalDateTime in the time zone that applies.
4) Subtract any week or day components of the duration from the date.
5) If the resulting time does not exist on the date due to a discontinuity in the time zone, use the semantics in LocalDateTime to convert to UTC and back to get a valid LocalDateTime.

These semantics match the iCalendar DURATION value type (IETF RFC 5545, Section 3.3.6).

SignedDuration

A SignedDuration represents a length of time that may be positive or negative and is typically used to express the offset of a point in time relative to an associated time. It is represented as a Duration, optionally preceded by a sign character. It is specified by the following ABNF:

```
signed-duration = ["+" / "-"] duration
```

A negative sign indicates a point in time at or before the associated time, a positive or no sign a time at or after the associated time.

TimeZoneId

Where `TimeZoneId` is given as a data type, it means a `String` that is either a time zone name in the IANA Time Zone Database TZDB or a custom time zone identifier defined in the `timeZones` property (see Clause 6.8.2).

Where an IANA time zone is specified, the zone rules of the respective zone records apply. Custom time zones are interpreted as described in Clause 6.8.2.

PatchObject

A PatchObject is of type `String[*]`, and represents an unordered set of patches on a JSON object. Each key is a path represented in a subset of JSON pointer format IETF RFC 6901. The paths have an implicit leading `/`, so each key is prefixed with `/` before applying the JSON pointer evaluation algorithm.

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

1) The pointer MUST NOT reference inside an array (i.e., you MUST NOT insert/delete from an array; the array MUST be replaced in its entirety instead).
2) All parts prior to the last (i.e., the value after the final slash) MUST already exist on the object being patched.
3) There MUST NOT be two patches in the PatchObject where the pointer of one is the prefix of the pointer of the other, e.g., `alerts/1/offset` and `alerts`.
4) The value for the patch MUST be valid for the property being set (of the correct type and obeying any other applicable restrictions), or if null the property MUST be optional.

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

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

A PatchObject does not define its own `@type` property (see Clause 6.1.1). A `@type` property in a patch MUST be handled as any other patched property value.

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

The PatchObject format is used to significantly reduce file size and duplicated content when specifying variations to a common object, such as with recurring events or when translating the data into multiple languages. It can also better preserve semantic intent if only the properties that should differ between the two objects are patched. For example, if one person is not going to a particular instance of a regularly scheduled event, in iCalendar you would have to duplicate the entire event in the override. In JSCalendar this is a small patch to show the difference. As only this property is patched, if the location of the event is changed, the occurrence will automatically still inherit this.

Relation

A Relation object defines the relation to other objects, using a possibly empty set of relation types. The object that defines this relation is the linking object, while the other object is the linked object. A Relation object has the following properties:

— @type: `String` (mandatory)
Specifies the type of this object. This MUST be `Relation`.
— relation: `String[Boolean]` (optional, default: empty Object)
Describes how the linked object is related to the linking object. The relation is defined as a set of relation types. If empty, the relationship between the two objects is unspecified.
Keys in the set MUST be one of the following values, or specified in the property definition where the Relation object is used, or a value registered in the IANA JSCalendar Enum Values registry, or a vendor-specific value (see Clause 5.3):
  — `first`: The linked object is the first in a series the linking object is part of.
  — `next`: The linked object is the next in a series the linking object is part of.
  — `child`: The linked object is a subpart of the linking object.
  — `parent`: The linking object is a subpart of the linked object.

  The value for each key in the map MUST be true.

Link

A Link object represents an external resource associated with the linking object. It has the following properties:

— @type: `String` (mandatory)
Specifies the type of this object. This MUST be `Link`.
— href: `String` (mandatory)
A URI IETF RFC 3986 from which the resource may be fetched.
This MAY be a `data:` URL IETF RFC 2397, but it is recommended that the file be hosted on a server to avoid embedding arbitrarily large data in JSCalendar object instances.
— cid: `String` (optional)
This MUST be a valid `content-id` value according to the definition of IETF RFC 2392, Section 2. The value MUST be unique within this Link object but has no meaning beyond that. It MAY be different from the link id for this Link object.
— contentType: `String` (optional)
The media type IETF RFC 6838 of the resource, if known.
— size: `UnsignedInt` (optional)
The size, in octets, of the resource when fully decoded (i.e., the number of octets in the file the user would download), if known. Note that this is an informational estimate, and

implementations must be prepared to handle the actual size being quite different when the resource is fetched.

— rel: `String` (optional)
Identifies the relation of the linked resource to the object. If set, the value MUST be a relation type from the IANA registry LINKRELS, as established in IETF RFC 8288.

— display: `String` (optional)
Describes the intended purpose of a link to an image. If set, the `rel` property MUST be set to `icon`. The value MUST be one of the following values, another value registered in the IANA JSCalendar Enum Values registry, or a vendor-specific value (see Clause 5.3):
  — `badge`: an image meant to be displayed alongside the title of the object.
  — `graphic`: a full image replacement for the object itself.
  — `fullsize`: an image that is used to enhance the object.
  — `thumbnail`: a smaller variant of `fullsize` to be used when space for the image is constrained.

— title: `String` (optional)
A human-readable plain-text description of the resource.

JSCalendar: A JSON representation of calendar data

1. Scope

This specification defines a data model and JSON representation of calendar data that can be used for storage and data exchange in a calendaring and scheduling environment.

2. Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

IETF RFC 2119, S. BRADNER. *Key words for use in RFCs to Indicate Requirement Levels*. 1997. RFC Publisher. https://www.rfc-editor.org/info/rfc2119.

IETF RFC 2392, E. LEVINSON. *Content-ID and Message-ID Uniform Resource Locators*. 1998. RFC Publisher. https://www.rfc-editor.org/info/rfc2392.

IETF RFC 2397, L. MASINTER. *The "data" URL scheme*. 1998. RFC Publisher. https://www.rfc-editor.org/info/rfc2397.

IETF RFC 3339, G. KLYNE and C. NEWMAN. *Date and Time on the Internet: Timestamps*. 2002. RFC Publisher. https://www.rfc-editor.org/info/rfc3339.

IETF RFC 3986, T. BERNERS-LEE, R. FIELDING and L. MASINTER. *Uniform Resource Identifier (URI): Generic Syntax*. 2005. RFC Publisher. https://www.rfc-editor.org/info/rfc3986.

IETF RFC 4122, P. LEACH, M. MEALLING and R. SALZ. *A Universally Unique IDentifier (UUID) URN Namespace*. 2005. RFC Publisher. https://www.rfc-editor.org/info/rfc4122.

IETF RFC 4589, H. SCHULZRINNE and H. TSCHOFENIG. *Location Types Registry*. 2006. RFC Publisher. https://www.rfc-editor.org/info/rfc4589.

IETF RFC 4648, S. JOSEFSSON. *The Base16, Base32, and Base64 Data Encodings*. 2006. RFC Publisher. https://www.rfc-editor.org/info/rfc4648.

IETF RFC 5234, P. OVERELL. *Augmented BNF for Syntax Specifications: ABNF*. 2008. RFC Publisher. https://www.rfc-editor.org/info/rfc5234.

IETF RFC 5545, B. DESRUISSEAUX (ed.). *Internet Calendaring and Scheduling Core Object Specification (iCalendar)*. 2009. RFC Publisher. https://www.rfc-editor.org/info/rfc5545.

IETF RFC 5546, C. DABOO (ed.). *iCalendar Transport-Independent Interoperability Protocol (iTIP)*. 2009. RFC Publisher. https://www.rfc-editor.org/info/rfc5546.

IETF RFC 5646, A. PHILLIPS and M. DAVIS (eds.). *Tags for Identifying Languages*. 2009. RFC Publisher. https://www.rfc-editor.org/info/rfc5646.

IETF RFC 5870, A. MAYRHOFER and C. SPANRING. *A Uniform Resource Identifier for Geographic Locations ('geo' URI)*. 2010. RFC Publisher. https://www.rfc-editor.org/info/rfc5870.

IETF RFC 6047, A. MELNIKOV (ed.). *iCalendar Message-Based Interoperability Protocol (iMIP)*. 2010. RFC Publisher. https://www.rfc-editor.org/info/rfc6047.

IETF RFC 6838, N. FREED, J. KLENSIN and T. HANSEN. *Media Type Specifications and Registration Procedures*. 2013. RFC Publisher. https://www.rfc-editor.org/info/rfc6838.

IETF RFC 6901, K. ZYP. *JavaScript Object Notation (JSON) Pointer*. 2013. RFC Publisher. https://www.rfc-editor.org/info/rfc6901.

IETF RFC 7493, T. BRAY (ed.). *The I-JSON Message Format*. 2015. RFC Publisher. https://www.rfc-editor.org/info/rfc7493.

IETF RFC 7529, C. DABOO and G. YAKUSHEV. *Non-Gregorian Recurrence Rules in the Internet Calendaring and Scheduling Core Object Specification (iCalendar)*. 2015. RFC Publisher. https://www.rfc-editor.org/info/rfc7529.

IETF RFC 7808, M. DOUGLASS and C. DABOO. *Time Zone Data Distribution Service*. 2016. RFC Publisher. https://www.rfc-editor.org/info/rfc7808.

IETF RFC 8126, M. COTTON, B. LEIBA and T. NARTEN. *Guidelines for Writing an IANA Considerations Section in RFCs*. 2017. RFC Publisher. https://www.rfc-editor.org/info/rfc8126.

IETF RFC 8174, B. LEIBA. *Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words*. 2017. RFC Publisher. https://www.rfc-editor.org/info/rfc8174.

IETF RFC 8259, T. BRAY (ed.). *The JavaScript Object Notation (JSON) Data Interchange Format*. 2017. RFC Publisher. https://www.rfc-editor.org/info/rfc8259.

IETF RFC 8288, M. NOTTINGHAM. *Web Linking*. 2017. RFC Publisher. https://www.rfc-editor.org/info/rfc8288.

CLDR, “Unicode Common Locale Data Repository”, http://cldr.unicode.org/

TZDB, “IANA Time Zone Database”, https://www.iana.org/time-zones

COLORS, “CSS Color Module”, https://www.w3.org/TR/css-color-3/

3. Terms and definitions

No terms and definitions are listed in this document.

4. JSCalendar Objects

This section describes the calendar object types specified by JSCalendar.

4.1. Event

Media type: `application/jscalendar+json;type=jsevent`

A Event represents a scheduled amount of time on a calendar, typically a meeting, appointment, reminder or anniversary. It is required to start at a certain point in time and typically has a non-zero duration. Multiple participants may partake in the event at multiple locations.

The @type Clause 6.1.1 property value MUST be `Event`.

4.2. Task

Media type: `application/jscalendar+json;type=jstask`

A Task represents an action-item, assignment, to-do or work item. It may start and be due at certain points in time, may take some estimated time to complete, and may recur, none of which is required.

The @type Clause 6.1.1 property value MUST be `Task`.

4.3. Group

Media type: `application/jscalendar+json;type=jsgroup`

A Group is a collection of Event Clause 4.1 and/or Task Clause 4.2 objects. Typically, objects are grouped by topic (e.g., by keywords) or calendar membership.

The @type Clause 6.1.1 property value MUST be `Group`.

5. Structure of JSCalendar Objects

A JSCalendar object is a JSON object IETF RFC 8259, which MUST be valid I-JSON (a stricter subset of JSON) IETF RFC 7493. Property names and values are case-sensitive.

The object has a collection of properties, as specified in the following sections. Properties are specified as being either mandatory or optional. Optional properties may have a default value, if explicitly specified in the property definition.

5.1. Object Type

JSCalendar objects MUST name their type in the `@type` property, if not explicitly specified otherwise for the respective object type. A notable exception to this rule is the PatchObject PatchObject.

5.2. Normalization and Equivalence

JSCalendar aims to provide unambiguous definitions for value types and properties, but does not define a general normalization or equivalence method for JSCalendar objects and types. This is because the notion of equivalence might range from byte-level equivalence to semantic equivalence, depending on the respective use case.

Normalization of JSCalendar objects is hindered because of the following reasons:

— Custom JSCalendar properties may contain arbitrary JSON values, including arrays. However, equivalence of arrays might or might not depend on the order of elements, depending on the respective property definition.
— Several JSCalendar property values are defined as URIs and media types, but normalization of these types is inherently protocol- and scheme-specific, depending on the use-case of the equivalence definition (see IETF RFC 3986, Section 6).

Considering this, the definition of equivalence and normalization is left to client and server implementations and to be negotiated by a calendar exchange protocol or defined elsewhere.

5.3. Vendor-specific Property Extensions, Values and Types

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

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

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

6. Common JSCalendar Properties

This section describes the properties that are common to the various JSCalendar object types. Specific JSCalendar object types may only support a subset of these properties. The object type definitions in Clause 7 describe the set of supported properties per type.

6.1. Metadata Properties

6.1.1. @type

Type: `String` (mandatory).

Specifies the type this object represents. The allowed value differs by object type and is defined in sections Clause 4.1, Clause 4.2, and Clause 4.3

6.1.2. uid

Type: `String` (mandatory).

A globally unique identifier, used to associate the object as the same across different systems, calendars and views. The value of this property MUST be unique across all JSCalendar objects, even if they are of different type. IETF RFC 4122 describes a range of established algorithms to generate universally unique identifiers (UUID). UUID version 4, described in IETF RFC 4122, Section 4.4, is RECOMMENDED.

For compatibility with IETF RFC 5545 UIDs, implementations MUST be able to receive and persist values of at least 255 octets for this property, but they MUST NOT truncate values in the middle of a UTF-8 multi-octet sequence.

6.1.3. relatedTo

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

Relates the object to other JSCalendar objects. This is represented as a map of the UIDs of the related objects to information about the relation.

If an object is split to make a “this and future” change to a recurrence, the original object MUST be truncated to end at the previous occurrence before this split, and a new object created to represent all the occurrences after the split. A `next` relation MUST be set on the original object’s relatedTo property for the UID of the new object. A `first` relation for the UID of the first object in the series MUST be set on the new object. Clients can then follow these UIDs to get the complete set of objects if the user wishes to modify them all at once.

6.1.4. prodId

Type: `String` (optional).

The identifier for the product that last updated the JSCalendar object. This should be set whenever the data in the object is modified (i.e., whenever the “updated” property is set).

The vendor of the implementation MUST ensure that this is a globally unique identifier, using some technique such as an FPI value, as defined in ISO.9070.1991.

This property SHOULD NOT be used to alter the interpretation of a JSCalendar object beyond the semantics specified in this document. For example, it is not to be used to further the understanding of non-standard properties, a practice that is known to cause long-term interoperability problems.

6.1.5. created

Type: `UTCDateTime` (optional).

The date and time this object was initially created.

6.1.6. updated

Type: `UTCDateTime` (mandatory).

The date and time the data in this object was last modified (or its creation date/time if not modified since).

6.1.7. sequence

Type: `UnsignedInt` (optional, default: 0).

Initially zero, this MUST be incremented by one every time a change is made to the object, except if the change only modifies the `participants` property (see Clause 6.5.5).

This is used as part of iTIP IETF RFC 5546 to know which version of the object a scheduling message relates to.

6.1.8. method

Type: `String` (optional).

The iTIP IETF RFC 5546 method, in lowercase. This MUST only be present if the JSCalendar object represents an iTIP scheduling message.

6.2. What and Where Properties

6.2.1. title

Type: `String` (optional, default: empty String).

A short summary of the object.

6.2.2. description

Type: `String` (optional, default: empty String).

A longer-form text description of the object. The content is formatted according to the `descriptionContentType` property.

6.2.3. descriptionContentType

Type: `String` (optional, default: `text/plain`).

Describes the media type IETF RFC 6838 of the contents of the `description` property. Media types MUST be sub-types of type `text`, and SHOULD be `text/plain` or `text/html` MEDIATYPES. They MAY include parameters and the `charset` parameter value MUST be `utf-8`, if specified. Descriptions of type `text/html` MAY contain `cid` URLs IETF RFC 2392 to reference links in the calendar object by use of the `cid` property of the Link object.

6.2.4. showWithoutTime

Type: `Boolean` (optional, default: false).

Indicates that the time is not important to display to the user when rendering this calendar object. An example of this is an event that conceptually occurs all day or across multiple days, such as “New Year’s Day” or “Italy Vacation”. While the time component is important for free-

busy calculations and checking for scheduling clashes, calendars may choose to omit displaying it and/or display the object separately to other objects to enhance the user's view of their schedule.

Such events are also commonly known as "all-day" events.

6.2.5. locations

Type: `Id[Location]` (optional).

A map of location ids to Location objects, representing locations associated with the object.

A Location object has the following properties. It MUST have at least one property other than the `relativeTo` property.

- @type: `String` (mandatory)
  Specifies the type of this object. This MUST be `Location`.
- name: `String` (optional)
  The human-readable name of the location.
- description: `String` (optional)
  Human-readable, plain-text instructions for accessing this location. This may be an address, set of directions, door access code, etc.
- locationTypes: `String[Boolean]` (optional)
  A set of one or more location types that describe this location. All types MUST be from the Location Types Registry LOCATIONTYPES as defined in IETF RFC 4589. The set is represented as a map, with the keys being the location types. The value for each key in the map MUST be true.
- relativeTo: `String` (optional)
  Specifies the relation between this location and the time of the JSCalendar object. This is primarily to allow events representing travel to specify the location of departure (at the start of the event) and location of arrival (at the end); this is particularly important if these locations are in different time zones, as a client may wish to highlight this information for the user.
  This MUST be one of the following values, another value registered in the IANA JSCalendar Enum Values registry, or a vendor-specific value (see Clause 5.3). Any value the client or server doesn't understand should be treated the same as if this property is omitted.
  - `start`: The event/task described by this JSCalendar object occurs at this location at the time the event/task starts.
  - end: The event/task described by this JSCalendar object occurs at this location at the time the event/task ends.

- timeZone: `TimeZoneId` (optional)
  A time zone for this location.
- coordinates: `String` (optional)
  A geo : URI IETF RFC 5870 for the location.
- links: `Id[Link]` (optional)
  A map of link ids to Link objects, representing external resources associated with this location, for example a vCard or image. If there are no links, this MUST be omitted (rather than specified as an empty set).

6.3. virtualLocations

Type: `Id[VirtualLocation]` (optional).

A map of virtual location ids to VirtualLocation objects, representing virtual locations, such as video conferences or chat rooms, associated with the object.

A VirtualLocation object has the following properties.

- @type: `String` (mandatory)
  Specifies the type of this object. This MUST be `VirtualLocation`.
- name: `String` (optional, default: empty String)

The human-readable name of the virtual location.
- description: `String` (optional)
  Human-readable plain-text instructions for accessing this virtual location. This may be a conference access code, etc.
- uri: `String` (mandatory)
  A URI IETF RFC 3986 that represents how to connect to this virtual location.
  This may be a telephone number (represented using the `tel:` scheme, e.g., `tel:+1-555-555-5555`) for a teleconference, a web address for online chat, or any custom URI.
- features: `String[Boolean]` (optional)
  A set of features supported by this virtual location. The set is represented as a map, with the keys being the feature. The value for each key in the map MUST be true.
  The feature MUST be one of the following values, another value registered in the IANA JSCalendar Enum Values registry, or a vendor-specific value (see Clause 5.3). Any value the client or server doesn't understand should be treated the same as if this feature is omitted.
  - `audio`: audio conferencing
  - `chat`: chat or instant messaging
  - `feed`: blog or atom feed
  - `moderator`: provides moderator-specific features
  - phone: phone conferencing
  - `screen`: screen sharing
  - `video`: video conferencing

6.3.1. links

Type: `Id[Link]` (optional).

A map of link ids to Link objects, representing external resources associated with the object.

Links with a rel of `enclosure` MUST be considered by the client to be attachments for download.

Links with a rel of `describedby` MUST be considered by the client to be alternative representations of the description.

Links with a rel of `icon` MUST be considered by the client to be images that it may use when presenting the calendar data to a user. The `display` property may be set to indicate the purpose of the respective image.

6.3.2. locale

Type: `String` (optional).

The language tag as defined in IETF RFC 5646 that best describes the locale used for the text in the calendar object, if known.

6.3.3. keywords

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

A set of keywords or tags that relate to the object. The set is represented as a map, with the keys being the keywords. The value for each key in the map MUST be true.

6.3.4. categories

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

A set of categories that relate to the calendar object. The set is represented as a map, with the keys being the categories specified as URIs. The value for each key in the map MUST be true.

In contrast to keywords, categories typically are structured. For example, a vendor owning the domain `example.com` might define the categories `http://example.com/categories/sports/american-football` and `http://example.com/categories/music/r-b`.

6.3.5. color

Type: `String` (optional).

A color clients MAY use when displaying this calendar object. The value is a color name taken from the set of names defined in Section 4.3 of COLORS, or an RGB value in hexadecimal notation, as defined in Section 4.2.1 of COLORS.

6.4. Recurrence Properties

Some events and tasks occur at regular or irregular intervals. Rather than having to copy the data for every occurrence there can be a master event with rules to generate recurrences, and/or overrides that add extra dates or exceptions to the rules.

The recurrence set is the complete set of instances for an object. It is generated by considering the following properties in order, all of which are optional:

1) The recurrenceRules property (Clause 6.4.2) generates a set of extra date-times on which the object occurs.
2) The excludedRecurrenceRules property (Clause 6.4.3) generates a set of date-times that are to be removed from the previously generated set of date-times on which the object occurs.
3) The recurrenceOverrides property (Clause 6.4.4) defines date-times which are added or excluded to form the final set. (This property may also contain changes to the object to apply to particular instances.)

6.4.1. recurrenceId

Type: `LocalDateTime` (optional).

If present, this JSCalendar object represents one occurrence of a recurring JSCalendar object. If present the `recurrenceRules` and `recurrenceOverrides` properties MUST NOT be present.

The value is a date-time either produced by the `recurrenceRules` of the master event, or added as a key to the `recurrenceOverrides` property of the master event.

6.4.2. recurrenceRules

Type: `RecurrenceRule[]` (optional).

Defines a set of recurrence rules (repeating patterns) for recurring calendar objects.

A Event recurs by applying the recurrence rules to the `start` date-time.

A Task recurs by applying the recurrence rules to the `start` date-time, if defined, otherwise it recurs by the due date-time, if defined. If the task defines neither a `start` nor due date-time, it MUST NOT define a `recurrenceRules` property.

If multiple recurrence rules are given, each rule is to be applied and then the union of the results used, ignoring any duplicates.

A RecurrenceRule object is a JSON object mapping of a RECUR value type in iCalendar IETF RFC 5545 IETF RFC 7529 and has the same semantics. It has the following properties:

— @type: `String` (mandatory)
Specifies the type of this object. This MUST be `RecurrenceRule`.

— frequency: `String` (mandatory)
The time span covered by each iteration of this recurrence rule (see Clause 6.4.2.1 for full semantics). This MUST be one of the following values:
  — `yearly`
  — `monthly`
  — `weekly`
  — `daily`
  — `hourly`
  — `minutely`
  — `secondly`

  This is the FREQ part from iCalendar, converted to lowercase.
— interval: `UnsignedInt` (optional, default: 1)
The interval of iteration periods at which the recurrence repeats. If included, it MUST be an integer >= 1.
This is the INTERVAL part from iCalendar.
— rscale: `String` (optional, default: “gregorian”)
The calendar system in which this recurrence rule operates, in lowercase. This MUST be either a CLDR-registered calendar system name CLDR, or a vendor-specific value (see Clause 5.3).
This is the RSCALE part from iCalendar RSCALE IETF RFC 7529, converted to lowercase.
— skip: `String` (optional, default: “omit”)
The behaviour to use when the expansion of the recurrence produces invalid dates. This property only has an effect if the frequency is “yearly” or “monthly”. It MUST be one of the following values:
  — `omit`
  — `backward`
  — `forward`

  This is the SKIP part from iCalendar RSCALE IETF RFC 7529, converted to lowercase.
— firstDayOfWeek: `String` (optional, default: “mo”)
The day on which the week is considered to start, represented as a lowercase abbreviated two-letter English day of the week. If included, it MUST be one of the following values:
  — `mo`
  — `tu`
  — `we`
  — `th`
  — `fr`
  — `sa`
  — `su`

  This is the WKST part from iCalendar.
— byDay: `NDay[]` (optional)
Days of the week on which to repeat. An `NDay` object has the following properties:
  — @type: `String` (mandatory)
  Specifies the type of this object. This MUST be `NDay`.
  — day: `String` (mandatory)
  A day of the week on which to repeat; the allowed values are the same as for the `firstDayOfWeek` RecurrenceRule property.
  This is the day-of-the-week of the BYDAY part in iCalendar, converted to lowercase.
  — nthOfPeriod: `Int` (optional)
  If present, rather than representing every occurrence of the weekday defined in the day property, it represents only a specific instance within the recurrence period. The value can be positive or negative, but MUST NOT be zero. A negative integer means nth last of period, with -1 being the last day.
  This is the ordinal part of the BYDAY value in iCalendar (e.g., 1 or -3).

— byMonthDay: `Int[]` (optional)
Days of the month on which to repeat. Valid values are between 1 and the maximum number of days any month may have in the calendar given by the “rscale” property, and the negative

values of these numbers. For example, in the Gregorian calendar valid values are 1 to 31 and -31 to -1. Negative values offset from the end of the month. The array MUST have at least one entry if included.
This is the BYMONTHDAY part in iCalendar.

— byMonth: `String[]` (optional)
The months in which to repeat. Each entry is a string representation of a number, starting from "1" for the first month in the calendar (e.g., "1" means January with the Gregorian calendar), with an optional "L" suffix (see IETF RFC 7529) for leap months (this MUST be uppercase, e.g., "3L"). The array MUST have at least one entry if included.
This is the BYMONTH part from iCalendar.

— byYearDay: `Int[]` (optional)
The days of the year on which to repeat. Valid values are between 1 and the maximum number of days any year may have in the calendar given by the "rscale" property, and the negative values of these numbers. For example, in the Gregorian calendar valid values are 1 to 366 and -366 to -1. Negative values offset from the end of the year. The array MUST have at least one entry if included.
This is the BYYEARDAY part from iCalendar.

— byWeekNo: `Int[]` (optional)
Weeks of the year in which to repeat. Valid values are between 1 and the maximum number of weeks any year may have in the calendar given by the "rscale" property, and the negative values of these numbers. For example, in the Gregorian calendar valid values are 1 to 53 and -53 to -1. The array MUST have at least one entry if included.
This is the BYWEEKNO part from iCalendar.

— byHour: `UnsignedInt[]` (optional)
The hours of the day in which to repeat. Valid values are 0 to 23. The array MUST have at least one entry if included. This is the BYHOUR part from iCalendar.

— byMinute: `UnsignedInt[]` (optional)
The minutes of the hour in which to repeat. Valid values are 0 to 59. The array MUST have at least one entry if included.
This is the BYMINUTE part from iCalendar.

— bySecond: `UnsignedInt[]` (optional)
The seconds of the minute in which to repeat. Valid values are 0 to 60. The array MUST have at least one entry if included.
This is the BYSECOND part from iCalendar.

— bySetPosition: `Int[]` (optional)
The occurrences within the recurrence interval to include in the final results. Negative values offset from the end of the list of occurrences. The array MUST have at least one entry if included. This is the BYSETPOS part from iCalendar.

— count: `UnsignedInt` (optional)
The number of occurrences at which to range-bound the recurrence. This MUST NOT be included if an `until` property is specified.
This is the COUNT part from iCalendar.

— until: `LocalDateTime` (optional)
The date-time at which to finish recurring. The last occurrence is on or before this date-time. This MUST NOT be included if a `count` property is specified. Note: if not specified otherwise for a specific JSCalendar object, this date is to be interpreted in the time zone specified in the JSCalendar object's `timeZone` property.
This is the UNTIL part from iCalendar.

6.4.2.1. Interpreting recurrence rules

A recurrence rule specifies a set of date-times for recurring calendar objects. A recurrence rule has the following semantics. Note, wherever "year", "month" or "day of month" is used, this is within the calendar system given by the "rscale" property, which defaults to "gregorian" if omitted.

1) A set of candidates is generated. This is every second within a period defined by the frequency property value:
   — `yearly`: every second from midnight on the 1st day of a year (inclusive) to midnight the 1st day of the following year (exclusive).

If skip is not “omit”, the calendar system has leap months and there is a byMonth property, generate candidates for the leap months even if they don’t occur in this year.
If skip is not “omit” and there is a byMonthDay property, presume each month has the maximum number of days any month may have in this calendar system when generating candidates, even if it’s more than this month actually has.

- `monthly`: every second from midnight on the 1st day of a month (inclusive) to midnight on the 1st of the following month (exclusive).
  If skip is not “omit” and there is a byMonthDay property, presume the month has the maximum number of days any month may have in this calendar system when generating candidates, even if it’s more than this month actually has.
- `weekly`: every second from midnight (inclusive) on the first day of the week (as defined by the firstDayOfWeek property, or Monday if omitted), to midnight 7 days later (exclusive).
- `daily`: every second from midnight at the start of the day (inclusive) to midnight at the end of the day (exclusive).
- `hourly`: every second from the beginning of the hour (inclusive) to the beginning of the next hour (exclusive).
- `minutely`: every second from the beginning of the minute (inclusive) to the beginning of the next minute (exclusive).
- `secondly`: the second itself, only.

2) Each date-time candidate is compared against all of the byX properties of the rule except bySetPosition. If any property in the rule does not match the date-time, the date-time is eliminated. Each byX property is an array; the date-time matches the property if it matches any of the values in the array. The properties have the following semantics:
   - byMonth: the date-time is in the given month.
   - byWeekNo: the date-time is in the nth week of the year. Negative numbers mean the nth last week of the year. This corresponds to weeks according to week numbering as defined in ISO.8601.2004, with a week defined as a seven day period, starting on the firstDayOfWeek property value or Monday if omitted. Week number one of the calendar year is the first week that contains at least four days in that calendar year.
     If the date-time is not valid (this may happen when generating candidates with a skip property in effect), it is always eliminated by this property.
   - byYearDay: the date-time is on the nth day of year. Negative numbers mean the nth last day of the year.
     If the date-time is not valid (this may happen when generating candidates with a skip property in effect), it is always eliminated by this property.
   - byMonthDay: the date-time is on the given day of the month. Negative numbers mean the nth last day of the month.
   - byDay: the date-time is on the given day of the week. If the day is prefixed by a number, it is the nth occurrence of that day of the week within the month (if frequency is monthly) or year (if frequency is yearly). Negative numbers means nth last occurrence within that period.
   - byHour: the date-time has the given hour value.
   - byMinute: the date-time has the given minute value.
   - bySecond: the date-time has the given second value.
3) If a skip property is defined and is not “omit”, there may be candidates that do not correspond to valid dates (e.g., 31st February in the Gregorian calendar). In this case, the properties MUST be considered in the order above and:
   a) After applying the byMonth filter, if the candidate’s month is invalid for the given year, increment it (if skip is “forward”) or decrement it (if skip is “backward”) until a valid month is found, incrementing/decrementing the year as well if passing through the beginning/end of the year. This only applies to calendar systems with leap months.
   b) After applying the byMonthDay filter, if the day of the month is invalid for the given month and year, change the date to the first day of the next month (if skip is “forward”) or the last day of the current month (if skip is “backward”).
   c) If any valid date produced after applying the skip is already a candidate, eliminate the duplicate. (For example after adjusting, 30th February and 31st February would both become the same “real” date, so one is eliminated as a duplicate.)
4) If a bySetPosition property is included, this is now applied to the ordered list of remaining dates. This property specifies the indexes of date-times to keep; all others should be eliminated. Negative numbers are indexes from the end of the list, with -1 being the last item.

5) Any date-times before the start date of the event are eliminated (see below for why this might be needed).
6) If a skip property is included and is not “omit”, eliminate any date-times that have already been produced by previous iterations of the algorithm. (This is not possible if skip is “omit”.)
7) If further dates are required (we have not reached the until date, or count limit) skip the next (interval — 1) sets of candidates, then continue from step 1.

When determining the set of occurrence dates for an event or task, the following extra rules must be applied:

1) The initial date-time to which the rule is applied (the `start` date-time for events; the `start` or due date-time for tasks) is always the first occurrence in the expansion (and is counted if the recurrence is limited by a “count” property), even if it would normally not match the rule.
2) The first set of candidates to consider is that which would contain the initial date-time. This means the first set may include candidates before the initial date-time; such candidates are eliminated from the results in step (4) as outlined before.
3) The following properties MUST be implicitly added to the rule under the given conditions:
   — If frequency is not `secondly` and no bySecond property: Add a bySecond property with the sole value being the seconds value of the initial date-time.
   — If frequency is not `secondly` or `minutely`, and no byMinute property: Add a byMinute property with the sole value being the minutes value of the initial date-time.
   — If frequency is not `secondly`, `minutely` or `hourly` and no byHour property: Add a byHour property with the sole value being the hours value of the initial date-time.
   — If frequency is `weekly` and no byDay property: Add a byDay property with the sole value being the day-of-the-week of the initial date-time.
   — If frequency is `monthly` and no byDay property and no byMonthDay property: Add a byMonthDay property with the sole value being the day-of-the-month of the initial date-time.
   — If frequency is `yearly` and no byYearDay property:
      — If there are no byMonth or byWeekNo properties, and either there is a byMonthDay property or there is no byDay property: Add a byMonth property with the sole value being the month of the initial date-time.
      — If there is no byMonthDay, byWeekNo or byDay properties: Add a byMonthDay property with the sole value being the day-of-the-month of the initial date-time.
      — If there is a byWeekNo property and no byMonthDay or byDay properties: Add a byDay property with the sole value being the day-of-the-week of the initial date-time.

6.4.3. excludedRecurrenceRules

Type: `RecurrenceRule[]` (optional).

Defines a set of recurrence rules (repeating patterns) for date-times on which the object will not occur. The rules are interpreted the same as for the “recurrenceRules” property (see Clause 6.4.2), with the exception that the initial date-time to which the rule is applied (the “start” date-time for events; the “start” or “due” date-time for tasks) is only considered part of the expansion if it matches the rule. The resulting set of date-times are then removed from those generated by the recurrenceRules property, as described in Clause 6.4.

6.4.4. recurrenceOverrides

Type: `LocalDateTime[PatchObject]` (optional).

Maps recurrence ids (the date-time produced by the recurrence rule) to the overridden properties of the recurrence instance.

If the recurrence id does not match a date-time from the recurrence rule (or no rule is specified), it is to be treated as an additional occurrence (like an RDATE from iCalendar). The patch object may often be empty in this case.

If the patch object defines the `excluded` property of an occurrence to be true, this occurrence is omitted from the final set of recurrences for the calendar object (like an EXDATE from iCalendar). Such a patch object MUST NOT patch any other property.

By default, an occurrence inherits all properties from the main object except the start (or due) date-time, which is shifted to match the recurrence id LocalDateTime. However, individual properties of the occurrence can be modified by a patch, or multiple patches. It is valid to patch the `start` property value, and this patch takes precedence over the value generated from the recurrence id. Both the recurrence id as well as the patched `start` date-time may occur before the original JSCalendar object's `start` or due date.

A pointer in the PatchObject MUST be ignored if it starts with one of the following prefixes:

— @type
— excludedRecurrenceRules
— method
— privacy
— prodId
— recurrenceId
— recurrenceOverrides
— recurrenceRules
— relatedTo
— replyTo
— timeZones
— uid

6.4.5. excluded

Type: `Boolean` (optional, default: false).

Defines if this object is an overridden, excluded instance of a recurring JSCalendar object (see Clause 6.4.4). If this property value is true, this calendar object instance MUST be removed from the occurrence expansion. The absence of this property, or the presence of its default value false, indicates that this instance MUST be included in the occurrence expansion.

6.5. Sharing and Scheduling Properties

6.5.1. priority

Type: `Int` (optional, default: 0).

Specifies a priority for the calendar object. This may be used as part of scheduling systems to help resolve conflicts for a time period.

The priority is specified as an integer in the range 0 to 9. A value of 0 specifies an undefined priority, for which the treatment will vary by situation. A value of 1 is the highest priority. A value of 2 is the second highest priority. Subsequent numbers specify a decreasing ordinal priority. A value of 9 is the lowest priority. Other integer values are reserved for future use.

6.5.2. freeBusyStatus

Type: `String` (optional, default: `busy`).

Specifies how this calendar object should be treated when calculating free-busy state. This MUST be one of the following values, another value registered in the IANA JSCalendar Enum Values registry, or a vendor-specific value (see Clause 5.3):

— `free`: The object should be ignored when calculating whether the user is busy.

— `busy`: The object should be included when calculating whether the user is busy.

6.5.3. privacy

Type: `String` (optional, default: `public`).

Calendar objects are normally collected together and may be shared with other users. The privacy property allows the object owner to indicate that it should not be shared, or should only have the time information shared but the details withheld. Enforcement of the restrictions indicated by this property are up to the API via which this object is accessed.

This property MUST NOT affect the information sent to scheduled participants; it is only interpreted by protocols that share the calendar objects belonging to one user with other users.

The value MUST be one of the following values, another value registered in the IANA JSCalendar Enum Values registry, or a vendor-specific value (see Clause 5.3). Any value the client or server doesn't understand should be preserved but treated as equivalent to `private`.

— `public`: The full details of the object are visible to those whom the object's calendar is shared with.
— `private`: The details of the object are hidden; only the basic time and metadata is shared. The following properties MAY be shared, any other properties MUST NOT be shared:
  — @type
  — created
  — due
  — duration
  — estimatedDuration
  — freeBusyStatus
  — privacy
  — recurrenceOverrides. Only patches which apply to another permissible property are allowed to be shared.
  — sequence
  — showWithoutTime
  — start
  — timeZone
  — timeZones
  — uid
  — updated

— `secret`: The object is hidden completely (as though it did not exist) when the calendar this object is in is shared.

6.5.4. replyTo

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

Represents methods by which participants may submit their response to the organizer of the calendar object. 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. Future methods may be defined in future specifications and registered with IANA; a calendar client MUST ignore any method it does not understand, but MUST preserve the method key and URI. This property MUST be omitted if no method is defined (rather than being specified as an empty object).

The following methods are defined:

— `imip`: The organizer accepts an iMIP IETF RFC 6047 response at this email address. The value MUST be a `mailto:` URI.

— `web`: Opening this URI in a web browser will provide the user with a page where they can submit a reply to the organizer. The value MUST be a URL using the `https:` scheme.
— `other`: The organizer is identified by this URI but the method for submitting the response is undefined.

6.5.5. participants

Type: `Id[Participant]` (optional).

A map of participant ids to participants, describing their participation in the calendar object.

If this property is set and any participant has a sendTo property, then the `replyTo` property of this calendar object MUST define at least one reply method.

A Participant object has the following properties:

— @type: `String` (mandatory)
Specifies the type of this object. This MUST be `Participant`.
— name: `String` (optional)
The display name of the participant (e.g., “Joe Bloggs”).
— email: `String` (optional)
The email address for the participant.
— description: `String` (optional).
A plain text description of this participant. For example, this may include more information about their role in the event or how best to contact them.
— sendTo: `String[String]` (optional)
Represents methods by which the participant may receive the invitation and updates to the calendar object.
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. Future methods may be defined in future specifications and registered with IANA; a calendar client MUST ignore any method it does not understand, but MUST preserve the method key and URI. This property MUST be omitted if no method is defined (rather than being specified as an empty object).
The following methods are defined:
  — `imip`: The participant accepts an iMIP IETF RFC 6047 request at this email address. The value MUST be a `mailto:` URI. It MAY be different from the value of the participant’s `email` property.
  — `other`: The participant is identified by this URI but the method for submitting the invitation is undefined.

— kind: `String` (optional)
What kind of entity this participant is, if known.
This MUST be one of the following values, another value registered in the IANA JSCalendar Enum Values registry, or a vendor-specific value (see Clause 5.3). Any value the client or server doesn’t understand should be treated the same as if this property is omitted.
  — `individual`: a single person
  — `group`: a collection of people invited as a whole
  — `location`: a physical location that needs to be scheduled, e.g., a conference room
  — `resource`: a non-human resource other than a location, such as a projector

— roles: `String[Boolean]` (mandatory)
A set of roles that this participant fulfills.
At least one role MUST be specified for the participant. The keys in the set MUST be one of the following values, another value registered in the IANA JSCalendar Enum Values registry, or a vendor-specific value (see Clause 5.3):
  — `owner`: The participant is an owner of the object. This signifies they have permission to make changes to it that affect the other participants. Non-owner participants may only

change properties that just affect themselves (for example setting their own alerts or changing their rsvp status).
  - `attendee`: The participant is expected to be present at the event.
  - `optional`: The participant's involvement with the event is optional. This is expected to be primarily combined with the "attendee" role.
  - `informational`: The participant is copied for informational reasons, and is not expected to attend.
  - `chair`: The participant is in charge of the event/task when it occurs.
  - `contact`: The participant is someone that may be contacted for information about the event.

  The value for each key in the map MUST be true. It is expected that no more than one of the roles "attendee" and "informational" be present; if more than one are given, "attendee" takes precedence over "informational". Roles that are unknown to the implementation MUST be preserved.
- locationId: `String` (optional)
  The location at which this participant is expected to be attending.
  If the value does not correspond to any location id in the `locations` property of the JSCalendar object, this MUST be treated the same as if the participant's locationId were omitted.
- language: `String` (optional)
  The language tag as defined in IETF RFC 5646 that best describes the participant's preferred language, if known.
- participationStatus: `String` (optional, default: `needs-action`)
  The participation status, if any, of this participant.
  The value MUST be one of the following values, another value registered in the IANA JSCalendar Enum Values registry, or a vendor-specific value (see Clause 5.3):
  - `needs-action`: No status yet set by the participant.
  - `accepted`: The invited participant will participate.
  - `declined`: The invited participant will not participate.
  - `tentative`: The invited participant may participate.
  - `delegated`: The invited participant has delegated their attendance to another participant, as specified in the delegatedTo property.

- participationComment: `String` (optional)
  A note from the participant to explain their participation status.
- expectReply: `Boolean` (optional, default: false)
  If true, the organizer is expecting the participant to notify them of their participation status.
- scheduleAgent: `String` (optional, default: `server`)
  Who is responsible for sending scheduling messages with this calendar object to the participant.
  The value MUST be one of the following values, another value registered in the IANA JSCalendar Enum Values registry, or a vendor-specific value (see Clause 5.3):
  - `server`: The calendar server will send the scheduling messages.
  - `client`: The calendar client will send the scheduling messages.
  - none: No scheduling messages are to be sent to this participant.

- scheduleForceSend: `Boolean` (optional, default: false)
  A client may set the property on a participant to true to request that the server send a scheduling message to the participant when it would not normally do so (e.g. if no significant change is made the object or the scheduleAgent is set to client). The property MUST NOT be stored in the JSCalendar object on the server or appear in a scheduling message.
- scheduleSequence: `UnsignedInt` (optional, default: 0)
  The sequence number of the last response from the participant. If defined, this MUST be a non-negative integer.
  This can be used to determine whether the participant has sent a new response following significant changes to the calendar object, and to determine if future responses are responding to a current or older view of the data.
- scheduleStatus: `String[]` (optional)

A list of status codes, returned from the processing of the most recent scheduling message sent to this participant. The status codes MUST be valid `statcode` values as defined in the ABNF in IETF RFC 5545, Section 3.8.8.3.
Servers MUST only add or change this property when they send a scheduling message to the participant. Clients SHOULD NOT change or remove this property if it was provided by the server. Clients MAY add, change, or remove the property for participants where the client is handling the scheduling.
This property MUST NOT be included in scheduling messages.

— scheduleUpdated: `UTCDateTime` (optional)
The timestamp for the most recent response from this participant.
This is the `updated` property of the last response when using iTIP. It can be compared to the `updated` property in future responses to detect and discard older responses delivered out of order.

— invitedBy: `String` (optional)
The participant id of the participant who invited this one, if known.

— delegatedTo: `String[Boolean]` (optional)
A set of participant ids that this participant has delegated their participation to. Each key in the set MUST be the id of a participant. The value for each key in the map MUST be true. If there are no delegates, this MUST be omitted (rather than specified as an empty set).

— delegatedFrom: `String[Boolean]` (optional)
A set of participant ids that this participant is acting as a delegate for. Each key in the set MUST be the id of a participant. The value for each key in the map MUST be true. If there are no delegators, this MUST be omitted (rather than specified as an empty set).

— memberOf: `String[Boolean]` (optional)
A set of group participants that were invited to this calendar object, which caused this participant to be invited due to their membership in the group(s). Each key in the set MUST be the id of a participant. The value for each key in the map MUST be true. If there are no groups, this MUST be omitted (rather than specified as an empty set).

— links: `Id[Link]` (optional)
A map of link ids to Link objects, representing external resources associated with this participant, for example a vCard or image. If there are no links, this MUST be omitted (rather than specified as an empty set).

— progress: `String` (optional; only allowed for participants of a Task).
Represents the progress of the participant for this task. It MUST NOT be set if the `participationStatus` of this participant is any value other than `accepted`. See Clause 7.2.5 for allowed values and semantics.

— progressUpdated: `UTCDateTime` (optional; only allowed for participants of a Task).
Specifies the date-time the progress property was last set on this participant. See Clause 7.2.6 for allowed values and semantics.

— percentComplete: `UnsignedInt` (optional; only allowed for participants of a Task).
Represents the percent completion of the participant for this task. The property value MUST be a positive integer between 0 and 100.

6.5.6. requestStatus

Type: `String` (optional).

A request status as returned from processing the most recent scheduling request for this JSCalendar object. The allowed values are defined by the ABNF definitions of `statcode`, `statdesc` and `extdata` in IETF RFC 5545, Section 3.8.8.3 and the following ABNF IETF RFC 5234:

```
reqstatus = statcode ";" statdesc [";" extdata]
```

Servers MUST only add or change this property when they performed a scheduling action. Clients SHOULD NOT change or remove this property if it was provided by the server. Clients MAY add, change, or remove the property when the client is handling the scheduling.

This property MUST only be included in scheduling messages according to the rules as defined for the REQUEST-STATUS iCalendar property in IETF RFC 5546.

6.6. Alerts Properties

6.6.1. useDefaultAlerts

Type: `Boolean` (optional, default: false).

If true, use the user's default alerts and ignore the value of the `alerts` property. Fetching user defaults is dependent on the API from which this JSCalendar object is being fetched, and is not defined in this specification. If an implementation cannot determine the user's default alerts, or none are set, it MUST process the alerts property as if `useDefaultAlerts` is set to false.

6.6.2. alerts

Type: `Id[Alert]` (optional).

A map of alert ids to Alert objects, representing alerts/reminders to display or send to the user for this calendar object.

An Alert Object has the following properties:

— @type: `String` (mandatory)
Specifies the type of this object. This MUST be `Alert`.
— trigger: `OffsetTrigger`|`AbsoluteTrigger`|`UnknownTrigger` (mandatory)
Defines when to trigger the alert. New types may be defined in future documents.
An `OffsetTrigger` object has the following properties:
  — @type: `String` (mandatory)
  Specifies the type of this object. This MUST be `OffsetTrigger`.
  — offset: `SignedDuration` (mandatory).
  Defines the offset at which to trigger the alert relative to the time property defined in the `relativeTo` property of the alert. Negative durations signify alerts before the time property, positive durations signify alerts after.
  — relativeTo: `String` (optional, default: `start`)
  Specifies the time property that the alert offset is relative to. The value MUST be one of:
    — `start`: triggers the alert relative to the start of the calendar object
    — end: triggers the alert relative to the end/due time of the calendar object

  An `AbsoluteTrigger` object has the following properties:
  — @type: `String` (mandatory)
  Specifies the type of this object. This MUST be `AbsoluteTrigger`.
  — when: `UTCDateTime` (mandatory).
  Defines a specific UTC date-time when the alert is triggered.

  An `UnknownTrigger` object is an object that contains a `@type` property whose value is not recognized (i.e., not `OffsetTrigger` or `AbsoluteTrigger`), plus zero or more other properties. This is for compatibility with client extensions and future specifications. Implementations SHOULD NOT trigger for trigger types they do not understand, but MUST preserve them.
— acknowledged: `UTCDateTime` (optional)
This records when an alert was last acknowledged. This is set when the user has dismissed the alert; other clients that sync this property SHOULD automatically dismiss or suppress duplicate alerts (alerts with the same alert id that triggered on or before this date-time).
For a recurring calendar object, setting the `acknowledged` property MUST NOT add a new override to the `recurrenceOverrides` property. If the alert is not already overridden, the acknowledged property MUST be set on the alert in the master event/task.
Certain kinds of alert action may not provide feedback as to when the user sees them, for example email based alerts. For those kinds of alerts, this property MUST be set immediately when the alert is triggered and the action successfully carried out.
— relatedTo: `String[Relation]` (optional)

Relates this alert to other alerts in the same JSCalendar object. If the user wishes to snooze an alert, the application MUST create an alert to trigger after snoozing. This new snooze alert MUST set a parent relation to the identifier of the original alert.

— action: `String` (optional, default: `display`)
Describes how to alert the user.
The value MUST be at most one of the following values, a value registered in the IANA JSCalendar Enum Values registry, or a vendor-specific value (see Clause 5.3):
  — `display`: The alert should be displayed as appropriate for the current device and user context.
  — `email`: The alert should trigger an email sent out to the user, notifying about the alert. This action is typically only appropriate for server implementations.

6.7. Multilingual Properties

6.7.1. localizations

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

A map of language tags IETF RFC 5646 to patch objects, which localize the calendar object into the locale of the respective language tag.

See the description of PatchObject PatchObject for the structure of the PatchObject. The patches are applied to the top-level calendar object. In addition, the `locale` property of the patched object is set to the language tag. All pointers for patches MUST end with one of the following suffixes; any patch that does not follow this MUST be ignored unless otherwise specified in a future RFC:

— title
— description
— name

A patch MUST NOT have the prefix `recurrenceOverrides`; any localization of the override MUST be a patch to the localizations property inside the override instead.

For example, a patch to `locations/abcd1234/title` is permissible, but a patch to `uid` or `recurrenceOverrides/2020-01-05T14:00:00/title` is not.

Note that this specification does not define how to maintain validity of localized content. For example, a client application changing a JSCalendar object's title property might also need to update any localizations of this property. Client implementations SHOULD provide the means to manage localizations, but how to achieve this is specific to the application's workflow and requirements.

6.8. Time Zone Properties

6.8.1. timeZone

Type: `TimeZoneId|null` (optional, default: null).

Identifies the time zone the object is scheduled in, or null for floating time. This is either a name from the IANA Time Zone Database TZDB or the TimeZoneId of a custom time zone from the `timeZones` property (Clause 6.8.2). If omitted, this MUST be presumed to be null (i.e., floating time).

6.8.2. timeZones

Type: `TimeZoneId[TimeZone]` (optional).

Maps identifiers of custom time zones to their time zone definitions. The following restrictions apply for each key in the map:

— To avoid conflict with names in the IANA Time Zone Database TZDB, it MUST start with the `/` character.
— It MUST be a valid `paramtext` value as specified in IETF RFC 5545, Section 3.1.
— At least one other property in the same JSCalendar object MUST reference a time zone using this identifier (i.e., orphaned time zones are not allowed).

An identifier need only be unique to this JSCalendar object. It MAY differ from the `tzId` property value of the TimeZone object it maps to.

A JSCalendar object may be part in a hierarchy of other JSCalendar objects (say, a Event is an entry in a Group). In this case, the set of time zones is the sum of the time zone definitions of this object and its parent objects. If multiple time zones with the same identifier exist, then the definition closest to the calendar object in relation to its parents MUST be used. (In context of Event, a time zone definition in its timeZones property has precedence over a definition of the same id in the Group). Time zone definitions in any children of the calendar object MUST be ignored.

A TimeZone object maps a VTIMEZONE component from iCalendar IETF RFC 5545 and the semantics are as defined there. A valid time zone MUST define at least one transition rule in the `standard` or `daylight` property. Its properties are:

— @type: `String` (mandatory)
Specifies the type of this object. This MUST be `TimeZone`.
— tzId: `String` (mandatory).
The TZID property from iCalendar. Note that this implies that the value MUST be a valid `paramtext` value as specified in IETF RFC 5545, Section 3.1.
— updated: `UTCDateTime` (optional)
The LAST-MODIFIED property from iCalendar.
— url: `String` (optional)
The TZURL property from iCalendar.
— validUntil: `UTCDateTime` (optional)
The TZUNTIL property from iCalendar specified in IETF RFC 7808.
— aliases: `String[Boolean]` (optional)
Maps the TZID-ALIAS-OF properties from iCalendar specified in IETF RFC 7808 to a JSON set of aliases. The set is represented as an object, with the keys being the aliases. The value for each key in the map MUST be true.
— standard: `TimeZoneRule[]` (optional)
The STANDARD sub-components from iCalendar. The order MUST be preserved during conversion.
— daylight: `TimeZoneRule[]` (optional).
The DAYLIGHT sub-components from iCalendar. The order MUST be preserved during conversion.

A TimeZoneRule object maps a STANDARD or DAYLIGHT sub-component from iCalendar, with the restriction that at most one recurrence rule is allowed per rule. It has the following properties:

— @type: `String` (mandatory)
Specifies the type of this object. This MUST be `TimeZoneRule`.
— start: `LocalDateTime` (mandatory)
The DTSTART property from iCalendar.
— offsetFrom: `String` (mandatory)
The TZOFFSETFROM property from iCalendar.
— offsetTo: `String` (mandatory)
The TZOFFSETTO property from iCalendar.
— recurrenceRules: `RecurrenceRule[]` (optional)
The RRULE property mapped as specified in Clause 6.4.2. During recurrence rule evaluation, the `until` property value MUST be interpreted as a local time in the UTC time zone.
— recurrenceOverrides: `LocalDateTime[PatchObject]` (optional)

Maps the RDATE properties from iCalendar. The set is represented as an object, with the keys being the recurrence dates. The patch object MUST be the empty JSON object ({}).

— names: `String[Boolean]` (optional)
Maps the TZNAME properties from iCalendar to a JSON set. The set is represented as an object, with the keys being the names, excluding any `tznparam` component from iCalendar. The value for each key in the map MUST be true.

— comments: `String[]` (optional).
Maps the COMMENT properties from iCalendar. The order MUST be preserved during conversion.

7. Type-specific JSCalendar Properties

7.1. Event Properties

In addition to the common JSCalendar object properties Clause 6 a Event has the following properties:

7.1.1. start

Type: `LocalDateTime` (mandatory).

The date/time the event starts in the event's time zone (as specified in the `timeZone` property, see Clause 6.8.1).

7.1.2. duration

Type: `Duration` (optional, default: `PT0S`).

The zero or positive duration of the event in the event's start time zone. The end time of an event can be found by adding the duration to the event's start time.

A Event MAY involve start and end locations that are in different time zones (e.g., a trans-continental flight). This can be expressed using the `relativeTo` and `timeZone` properties of the Event's Location objects (see Clause 6.2.5).

7.1.3. status

Type: `String` (optional, default: `confirmed`).

The scheduling status (Clause 6.5) of a Event. If set, it MUST be one of the following values, another value registered in the IANA JSCalendar Enum Values registry, or a vendor-specific value (see Clause 5.3):

— `confirmed`: Indicates the event is definitely happening.
— `cancelled`: Indicates the event has been cancelled.
— `tentative`: Indicates the event may happen.

7.2. Task Properties

In addition to the common JSCalendar object properties Clause 6 a Task has the following properties:

7.2.1. due

Type: `LocalDateTime` (optional).

The date/time the task is due in the task's time zone.

7.2.2. start

Type: `LocalDateTime` (optional).

The date/time the task should start in the task's time zone.

7.2.3. estimatedDuration

Type: `Duration` (optional).

Specifies the estimated positive duration of time the task takes to complete.

7.2.4. percentComplete

Type: `UnsignedInt` (optional).

Represents the percent completion of the task overall. The property value MUST be a positive integer between 0 and 100.

7.2.5. progress

Type: `String` (optional).

Defines the progress of this task. If omitted, the default progress (Clause 6.5) of a Task is defined as follows (in order of evaluation):

— `completed`: if the `progress` property value of all participants is `completed`.
— `failed`: if at least one `progress` property value of a participant is `failed`.
— `in-process`: if at least one `progress` property value of a participant is `in-process`.
— `needs-action`: If none of the other criteria match.

If set, it MUST be one of the following values, another value registered in the IANA JSCalendar Enum Values registry, or a vendor-specific value (see Clause 5.3):

— `needs-action`: Indicates the task needs action.
— `in-process`: Indicates the task is in process.
— `completed`: Indicates the task is completed.
— `failed`: Indicates the task failed.
— `cancelled`: Indicates the task was cancelled.

7.2.6. progressUpdated

Type: `UTCDateTime` (optional).

Specifies the date/time the `progress` property of either the task overall (Clause 7.2.5) or a specific participant (Clause 6.5.5) was last updated.

If the task is recurring and has future instances, a client may want to keep track of the last progress update timestamp of a specific task recurrence, but leave other instances unchanged. One way to achieve this is by overriding the progressUpdated property in the task `recurrenceOverrides` property. However, this could produce a long list of timestamps for regularly recurring tasks. An alternative approach is to split the Task into a current, single instance of Task with this instance progress update time and a future recurring instance. See also Clause 6.1.3 on splitting.

7.3. Group Properties

Group supports the following common JSCalendar Clause 6 properties:

— @type
— uid
— prodId
— created
— updated
— title
— description
— descriptionContentType
— links
— locale
— keywords
— categories
— color
— timeZones

In addition, the following Group-specific properties are supported:

7.3.1. entries

Type: `(Task|Event)[]` (mandatory).

A collection of group members. Implementations MUST ignore entries of unknown type.

7.3.2. source

Type: `String` (optional).

The source from which updated versions of this group may be retrieved from. The value MUST be a URI.

8. Examples

The following examples illustrate several aspects of the JSCalendar data model and format. The examples may omit mandatory or additional properties, which is indicated by a placeholder property with key …. While most of the examples use calendar event objects, they are also illustrative for tasks.

8.1. Simple event

This example illustrates a simple one-time event. It specifies a one-time event that begins on January 15, 2020 at 1pm New York local time and ends after 1 hour.

```
{
  "@type": "Event",
  "uid": "a8df6573-0474-496d-8496-033ad45d7fea",
  "updated": "2020-01-02T18:23:04Z",
  "title": "Some event",
  "start": "2020-01-15T13:00:00",
  "timeZone": "America/New_York",
  "duration": "PT1H"
```



```
}
```

8.2. Simple task

This example illustrates a simple task for a plain to-do item.

```
{
  "@type": "Task",
  "uid": "2a358cee-6489-4f14-a57f-c104db4dc2f2",
  "updated": "2020-01-09T14:32:01Z",
  "title": "Do something"
}
```

8.3. Simple group

This example illustrates a simple calendar object group that contains an event and a task.

```
{
  "@type": "Group",
  "uid": "bf0ac22b-4989-4caf-9ebd-54301b4ee51a",
  "updated": "2020-01-15T18:00:00Z",
  "name": "A simple group",
  "entries": [{
    "@type": "Event",
    "uid": "a8df6573-0474-496d-8496-033ad45d7fea",
    "updated": "2020-01-02T18:23:04Z",
    "title": "Some event",
    "start": "2020-01-15T13:00:00",
    "timeZone": "America/New_York",
    "duration": "PT1H"
  },
  {
    "@type": "Task",
    "uid": "2a358cee-6489-4f14-a57f-c104db4dc2f2",
    "updated": "2020-01-09T14:32:01Z",
    "title": "Do something"
  }]
}
```

8.4. All-day event

This example illustrates an event for an international holiday. It specifies an all-day event on April 1 that occurs every year since the year 1900.

```
{
  "...": "",
  "title": "April Fool's Day",
  "showWithoutTime": true,
  "start": "1900-04-01T00:00:00",
  "duration": "P1D",
  "recurrenceRules": [{
    "@type": "RecurrenceRule",
    "frequency": "yearly"
  }]
```

```
}
```

8.5. Task with a due date

This example illustrates a task with a due date. It is a reminder to buy groceries before 6pm Vienna local time on January 19, 2020. The calendar user expects to need 1 hour for shopping.

```
{
  "...": "",
  "title": "Buy groceries",
  "due": "2020-01-19T18:00:00",
  "timeZone": "Europe/Vienna",
  "estimatedDuration": "PT1H"
}
```

8.6. Event with end time zone

This example illustrates the use of end time zones by use of an international flight. The flight starts on April 1, 2020 at 9am in Berlin local time. The duration of the flight is scheduled at 10 hours 30 minutes. The time at the flight's destination is in the same time zone as Tokyo. Calendar clients could use the end time zone to display the arrival time in Tokyo local time and highlight the time zone difference of the flight. The location names can serve as input for navigation systems.

```
{
  "...": "",
  "title": "Flight XY51 to Tokyo",
  "start": "2020-04-01T09:00:00",
  "timeZone": "Europe/Berlin",
  "duration": "PT10H30M",
  "locations": {
    "1": {
      "@type": "Location",
      "rel": "start",
      "name": "Frankfurt Airport (FRA)"
    },
    "2": {
      "@type": "Location",
      "rel": "end",
      "name": "Narita International Airport (NRT)",
      "timeZone": "Asia/Tokyo"
    }
  }
}
```

8.7. Floating-time event (with recurrence)

This example illustrates the use of floating time. Since January 1, 2020, a calendar user blocks 30 minutes every day to practice Yoga at 7am local time, in whatever time zone the user is located on that date.

```
{
  "...": "",
  "title": "Yoga",
  "start": "2020-01-01T07:00:00",
  "duration": "PT30M",
  "recurrenceRules": [{
    "@type": "RecurrenceRule",
    "frequency": "daily"
  }]
```



```
}
```

8.8. Event with multiple locations and localization

This example illustrates an event that happens at both a physical and a virtual location. Fans can see a live concert on premises or online. The event title and descriptions are localized.

```
{
  "...": "",
  "title": "Live from Music Bowl: The Band",
  "description": "Go see the biggest music event ever!",
  "locale": "en",
  "start": "2020-07-04T17:00:00",
  "timeZone": "America/New_York",
  "duration": "PT3H",
  "locations": {
    "loc1": {
      "@type": "Location",
      "name": "The Music Bowl",
      "description": "Music Bowl, Central Park, New York",
      "coordinates": "geo:40.7829,-73.9654"
    }
  },
  "virtualLocations": {
    "vloc1": {
      "@type": "VirtualLocation",
      "name": "Free live Stream from Music Bowl",
      "uri": "https://stream.example.com/the_band_2020"
    }
  },
  "localizations": {
    "de": {
      "title": "Live von der Music Bowl: The Band!",
      "description": "Schau dir das größte Musikereignis an!",
      "virtualLocations/vloc1/name": "Gratis Live-Stream aus der Music Bowl"
    }
  }
}
```

8.9. Recurring event with overrides

This example illustrates the use of recurrence overrides. A math course at a University is held for the first time on January 8, 2020 at 9am London time and occurs every week until June 24, 2020. Each lecture lasts for one hour and 30 minutes and is located at the Mathematics department. This event has exceptional occurrences: at the last occurrence of the course is an exam, which lasts for 2 hours and starts at 10am. Also, the location of the exam differs from the usual location. On April 1 no course is held. On January 7 at 2pm is an optional introduction course, that occurs before the first regular lecture.

```
{
  "...": "",
  "title": "Calculus I",
  "start": "2020-01-08T09:00:00",
  "timeZone": "Europe/London",
  "duration": "PT1H30M",
  "locations": {
    "mlab": {
      "@type": "Location",
      "title": "Math lab room 1",
```



```
      "description": "Math Lab I, Department of Mathematics"
    }
  },
  "recurrenceRules": [{
    "@type": "RecurrenceRule",
    "frequency": "weekly",
    "until": "2020-06-24T09:00:00"
  }],
  "recurrenceOverrides": {
    "2020-01-07T14:00:00": {
      "title": "Introduction to Calculus I (optional)"
    },
    "2020-04-01T09:00:00": {
      "excluded": true
    },
    "2020-06-25T09:00:00": {
      "title": "Calculus I Exam",
      "start": "2020-06-25T10:00:00",
      "duration": "PT2H",
      "locations": {
        "auditorium": {
          "@type": "Location",
          "title": "Big Auditorium",
          "description": "Big Auditorium, Other Road"
        }
      }
    }
  }
}
```

8.10. Recurring event with participants

This example illustrates scheduled events. A team meeting occurs every week since January 8, 2020 at 9am Johannesburg time. The event owner also chairs the event. Participants meet in a virtual meeting room. An attendee has accepted the invitation, but on March 4, 2020 he is unavailable and declined participation for this occurrence.

```
{
  "...": "",
  "title": "FooBar team meeting",
  "start": "2020-01-08T09:00:00",
  "timeZone": "Africa/Johannesburg",
  "duration": "PT1H",
  "virtualLocations": {
    "1": {
      "@type": "VirtualLocation",
      "name": "ChatMe meeting room",
      "uri": "https://chatme.example.com?id=1234567&pw=a8a24627b63d396e"
    }
  },
  "recurrenceRules": [{
    "@type": "RecurrenceRule",
    "frequency": "weekly"
  }],
  "replyTo": {
    "imip": "mailto:f245f875-7f63-4a5e-a2c8@schedule.example.com"
  },
  "participants": {
    "dG9tQGZvb2Jhci5xlLmNvbQ": {
      "@type": "Participant",
```



```
      "name": "Tom Tool",
      "email": "tom@foobar.example.com",
      "sendTo": {
        "imip": "mailto:tom@calendar.example.com"
      },
      "participationStatus": "accepted",
      "roles": {
        "attendee": true
      }
    },
    "em9lQGZvb2GFtcGxlLmNvbQ": {
      "@type": "Participant",
      "name": "Zoe Zelda",
      "email": "zoe@foobar.example.com",
      "sendTo": {
        "imip": "mailto:zoe@foobar.example.com"
      },
      "participationStatus": "accepted",
      "roles": {
        "owner": true,
        "attendee": true,
        "chair": true
      }
    }
  },
  "recurrenceOverrides": {
    "2020-03-04T09:00:00": {
      "participants/dG9tQGZvb2Jhci5xlLmNvbQ/participationStatus":
                                                              "declined"
    }
  }
}{
  "...": "",
  "title": "FooBar team meeting",
  "start": "2020-01-08T09:00:00",
  "timeZone": "Africa/Johannesburg",
  "duration": "PT1H",
  "virtualLocations": {
    "1": {
      "@type": "VirtualLocation",
      "name": "ChatMe meeting room",
      "uri": "https://chatme.example.com?id=1234567&pw=a8a24627b63d396e"
    }
  },
  "recurrenceRules": [{
    "@type": "RecurrenceRule",
    "frequency": "weekly"
  }],
  "replyTo": {
    "imip": "mailto:f245f875-7f63-4a5e-a2c8@schedule.example.com"
  },
  "participants": {
    "dG9tQGZvb2Jhci5xlLmNvbQ": {
      "@type": "Participant",
      "name": "Tom Tool",
      "email": "tom@foobar.example.com",
      "sendTo": {
        "imip": "mailto:tom@calendar.example.com"
      },
      "participationStatus": "accepted",
```



```
      "roles": {
        "attendee": true
      }
    },
    "em9lQGZvb2GFtcGxlLmNvbQ": {
      "@type": "Participant",
      "name": "Zoe Zelda",
      "email": "zoe@foobar.example.com",
      "sendTo": {
        "imip": "mailto:zoe@foobar.example.com"
      },
      "participationStatus": "accepted",
      "roles": {
        "owner": true,
        "attendee": true,
        "chair": true
      }
    }
  },
  "recurrenceOverrides": {
    "2020-03-04T09:00:00": {
      "participants/dG9tQGZvb2Jhci5xlLmNvbQ/participationStatus":
                                                              "declined"
    }
  }
}
```

9. Security Considerations

Calendaring and scheduling information is very privacy-sensitive. It can reveal the social network of a user; location information of this user and those in their social network; identity and credentials information; and the patterns of behavior of the user in both the physical and cyber realm. Additionally, calendar events and tasks can could influence the physical location of a user or their cyber behavior within a known time window. Its transmission and storage must be done carefully to protect it from possible threats, such as eavesdropping, replay, message insertion, deletion, modification, and on-path attacks.

The data being stored and transmitted may be used in systems with real world consequences. For example, a home automation system may turn an alarm on and off. Or a coworking space may charge money to the organiser of an event that books one of their meeting rooms. Such systems must be careful to authenticate all data they receive to prevent them from being subverted, and ensure the change comes from an authorized entity.

This document just defines the data format; such considerations are primarily the concern of the API or method of storage and transmission of such files.

9.1. Expanding Recurrences

A recurrence rule may produce infinite occurrences of an event. Implementations MUST handle expansions carefully to prevent accidental or deliberate resource exhaustion.

Conversely, a recurrence rule may be specified that does not expand to anything. It is not always possible to tell this through static analysis of the rule, so implementations MUST be careful to avoid getting stuck in infinite loops, or otherwise exhausting resources while searching for the next occurrence.

Events recur in the event's time zone. If the user is in a different time zone, daylight saving transitions may cause an event that normally occurs at, for example, 9am to suddenly shift an hour earlier. This may be used in an attempt to cause a participant to miss an important meeting. User

agents must be careful to translate date-times correctly between time zones and may wish to call out unexpected changes in the time of a recurring event.

9.2. JSON Parsing

The Security Considerations of IETF RFC 8259 apply to the use of JSON as the data interchange format.

As for any serialization format, parsers need to thoroughly check the syntax of the supplied data. JSON uses opening and closing tags for several types and structures, and it is possible that the end of the supplied data will be reached when scanning for a matching closing tag; this is an error condition, and implementations need to stop scanning at the end of the supplied data.

JSON also uses a string encoding with some escape sequences to encode special characters within a string. Care is needed when processing these escape sequences to ensure that they are fully formed before the special processing is triggered, with special care taken when the escape sequences appear adjacent to other (non-escaped) special characters or adjacent to the end of data (as in the previous paragraph).

If parsing JSON into a non-textual structured data format, implementations may need to allocate storage to hold JSON string elements. Since JSON does not use explicit string lengths, the risk of denial of service due to resource exhaustion is small, but implementations may still wish to place limits on the size of allocations they are willing to make in any given context, to avoid untrusted data causing excessive memory allocation.

9.3. URI Values

Several JSCalendar properties contain URIs as values, and processing these properties requires extra care. IETF RFC 3986, Section 7 discusses security risks related to URIs.

Fetching remote resources carries inherent risks. Connections must only be allowed on well known ports, using allowed protocols (generally just HTTP/HTTPS on their default ports). The URL must be resolved externally and not allowed to access internal resources. Connecting to an external source reveals IP (and therefore generally location) information.

A maliciously constructed JSCalendar object may contain a very large number of URIs. In the case of published calendars with a large number of subscribers, such objects could be widely distributed. Implementations should be careful to limit the automatic fetching of linked resources to reduce the risk of this being an amplification vector for a denial-of-service attack.

9.4. Spam

Calendar systems may receive JSCalendar files from untrusted sources, in particular as attachments to emails. This can be a vector for an attacker to inject spam into a user's calendar. This may confuse, annoy, and mislead users, or overwhelm their calendar with bogus events, preventing them from seeing legitimate ones.

Heuristic, statistical or machine-learning-based filters can be effective in filtering out spam. Authentication mechanisms such as DKIM IETF RFC 6376 can help establish the source of messages and associate the data with existing relationships (such as an address book contact). Misclassifications are always possible, however, and providing a mechanism for users to quickly correct this is advised.

Confusable unicode characters may be used to trick a user into trusting a JSCalendar file that appears to come from a known contact but is actually from a similar-looking source controlled by an attacker.

9.5. Duplication

It is important for calendar systems to maintain the UID of an event when updating it to avoid unexpected duplication of events. Consumers of the data may not remove the previous version of the event if it has a different UID. This can lead to a confusing situation for the user, with many variations of the event and no indication of which one is correct. Care must be taken by consumers of the data to remove old events where possible to avoid an accidental denial-of-service attack due to the volume of data.

9.6. Time Zones

Events recur in a particular time zone. When this differs from the user's current time zone, it may unexpectedly cause an occurrence to shift in time for that user due to a daylight savings change in the event's time zone. A maliciously crafted event could attempt to confuse users with such an event to ensure a meeting is missed.

10. IANA Considerations

10.1. Media Type Registration

This document defines a media type for use with JSCalendar data formatted in JSON.

Table 1

Type name:	application
Subtype name:	jscalendar+json
Required parameters:	type

The `type` parameter conveys the type of the JSCalendar data in the body part. The allowed parameter values correspond to the `@type` property of the JSON-formatted JSCalendar object in the body:

— `jsevent`: the `@type` property value MUST be `Event`
— `jstask`: the `@type` property value MUST be `Task`
— `jsgroup`: the `@type` property value MUST be `Group`

No other parameter values are allowed. The parameter MUST NOT occur more than once.

Optional parameters	none
Encoding considerations	Same as encoding considerations of application/json as specified in IETF RFC 8259, Section 11.
Security considerations	See Clause 9 of this document.
Interoperability considerations	While JSCalendar is designed to avoid ambiguities as much as possible, when converting objects from other calendar formats to/from JSCalendar it is possible that differing representations for the same logical data might arise, or ambiguities in interpretation. The semantic equivalence of two JSCalendar objects may be determined differently by different applications, for example where URL values differ in case between the two objects.
Published specification	This specification.

Applications that use this media type	Applications that currently make use of the text/calendar and application/calendar+json media types can use this as an alternative. Similarly, applications that use the application/json media type to transfer calendaring data can use this to further specify the content.
Fragment identifier considerations	A JSON Pointer fragment identifier may be used, as defined in IETF RFC 6901, Section 6.
Additional information	

Magic number(s)	N/A
File extensions(s)	N/A
Macintosh file type code(s)	N/A
Person & email address to contact for further information	calsify@ietf.org
Intended usage	COMMON
Restrictions on usage	N/A
Author	See the “Author’s Address” section of this document.
Change controller	IETF

10.2. Creation of “JSCalendar Properties” Registry

The IANA will create the “JSCalendar Properties” registry to allow interoperability of extensions to JSCalendar objects.

This registry follows the Expert Review process (IETF RFC 8126, Section 4.5). If the “intended use” field is `common`, sufficient documentation is required to enable interoperability. Preliminary community review for this registry is optional but strongly encouraged.

A registration can have an intended use of `common`, `reserved`, or `obsolete`. The IANA will list common-use registrations prominently and separately from those with other intended use values.

A `reserved` registration reserves a property name without assigning semantics to avoid name collisions with future extensions or protocol use.

An `obsolete` registration denotes a property that is no longer expected to be added by up-to-date systems. A new property has probably been defined covering the obsolete property’s semantics.

The JSCalendar property registration procedure is not a formal standards process but rather an administrative procedure intended to allow community comment and sanity checking without excessive time delay. It is designed to encourage vendors to document and register new properties they add for use cases not covered by the original specification, leading to increased interoperability.

10.3. Preliminary Community Review

Notice of a potential new registration SHOULD be sent to the Calext mailing list calsify@ietf.org>; for review. This mailing list is appropriate to solicit community feedback on a proposed new property.

Properties registrations must be marked with their intended use: “common”, “reserved” or “obsolete”.

The intent of the public posting to this list is to solicit comments and feedback on the choice of the property name, the unambiguity of the specification document, and a review of any interoperability or security considerations. The submitter may submit a revised registration proposal or abandon the registration completely at any time.

10.4. Submit Request to IANA

Registration requests can be sent to <iana@iana.org>.

10.5. Designated Expert Review

The primary concern of the designated expert (DE) is preventing name collisions and encouraging the submitter to document security and privacy considerations. For a common-use registration, the DE is expected to confirm that suitable documentation, as described in IETF RFC 8126, Section 4.6, is available to ensure interoperability. That documentation will usually be in an RFC, but simple definitions are likely to use a web/wiki page, and if a sentence or two is deemed sufficient it could be described in the registry itself. The DE should also verify that the property name does not conflict with work that is active or already published within the IETF. A published specification is not required for reserved or obsolete registrations.

The DE will either approve or deny the registration request and publish a notice of the decision to the Calext WG mailing list or its successor, as well as inform IANA. A denial notice must be justified by an explanation, and, in the cases where it is possible, concrete suggestions on how the request can be modified so as to become acceptable should be provided.

10.6. Change Procedures

Once a JSCalendar property has been published by the IANA, the change controller may request a change to its definition. The same procedure that would be appropriate for the original registration request is used to process a change request.

JSCalendar property registrations may not be deleted; properties that are no longer believed appropriate for use can be declared obsolete by a change to their “intended use” field; such properties will be clearly marked in the lists published by the IANA.

Significant changes to a JSCalendar property’s definition should be requested only when there are serious omissions or errors in the published specification, as such changes may cause interoperability issues. When review is required, a change request may be denied if it renders entities that were valid under the previous definition invalid under the new definition.

The owner of a JSCalendar property may pass responsibility to another person or agency by informing the IANA; this can be done without discussion or review.

10.7. JSCalendar Properties Registry Template

— Property Name: The name of the property. The property name MUST NOT already be registered for any of the object types listed in the “Property Context” field of this registration. Other object types MAY already have registered a different property with the same name, however the same name SHOULD only be used when the semantics are analogous.
— Property Type: The type of this property, using type signatures as specified in Type Signatures. The property type MUST be registed in the Type Registry.
— Property Context: A comma-separated list of JSCalendar object types this property is allowed on.
— Reference or Description: A brief description or RFC number and section reference where the property is specified (omitted for “reserved” property names).
— Intended Use: Common, reserved, or obsolete.

— Change Controller: Who may request a change to this entry's definition (`IETF` for IETF-stream RFCs).

10.8. Initial Contents for the JSCalendar Properties Registry

The following table lists the initial entries of the JSCalendar Properties registry. All properties are for common-use. All RFC section references are for this document. The change controller for all these properties is "IETF".

Table 2

Property Name	Property Type	Property Context	Reference or Description
@type	String	Event, Task, Group, AbsoluteTrigger, Alert, Link, Location, NDay, OffsetTrigger, Participant, RecurrenceRule, Relation, TimeZone, TimeZoneRule, VirtualLocation	Clause 6.1.1, Clause 6.6.2, Link, Clause 6.2.5, Clause 6.5.5, Clause 6.4.2, Clause 6.1.3, Clause 6.8.2, Clause 6.3
acknowledged	UTCDateTime	Alert	Clause 6.6.2
action	String	Alert	Clause 6.6.2
alerts	Id[Alert]	Event, Task	Clause 6.6.2
aliases	String[Boolean]	TimeZone	Clause 6.8.2
byDay	NDay[]	RecurrenceRule	Clause 6.4.2
byHour	UnsignedInt[]	RecurrenceRule	Clause 6.4.2
byMinute	UnsignedInt[]	RecurrenceRule	Clause 6.4.2
byMonth	String[]	RecurrenceRule	Clause 6.4.2
byMonthDay	Int[]	RecurrenceRule	Clause 6.4.2
bySecond	UnsignedInt[]	RecurrenceRule	Clause 6.4.2
bySetPosition	Int[]	RecurrenceRule	Clause 6.4.2
byWeekNo	Int[]	RecurrenceRule	Clause 6.4.2
byYearDay	Int[]	RecurrenceRule	Clause 6.4.2
categories	String[Boolean]	Event, Task, Group	Clause 6.3.4
cid	String	Link	Link
color	String	Event, Task, Group	Clause 6.3.5

Property Name	Property Type	Property Context	Reference or Description
comments	String[]	TimeZoneRule	Clause 6.8.2
contentType	String	Link	Link
coordinates	String	Location	Clause 6.2.5
count	UnsignedInt	RecurrenceRule	Clause 6.4.2
created	UnsignedInt	RecurrenceRule	Clause 6.4.2
day	String	NDay	Clause 6.4.2
daylight	TimeZoneRule[]	TimeZone	Clause 6.8.2
delegatedFrom	String[Boolean]	Participant	Clause 6.5.5
delegatedTo	String[Boolean]	Participant	Clause 6.5.5
description	String	Event, Task, Location, Participant, VirtualLocation	Clause 6.2.2, Clause 6.2.5, Clause 6.5.5, Clause 6.3
descriptionContentType	String	Event, Task	Clause 6.2.3
display	String	Link	Link
due	LocalDateTime	Task	Clause 7.2.1
duration	Duration	Event	Clause 7.1.2
email	String	Participant	Clause 6.5.5
entries	(Task|Event)[]	Group	Clause 7.3.1
estimatedDuration	Duration	Task	Clause 7.2.3
excluded	Boolean	Event, Task	Clause 6.4.5
excludedRecurrenceRules	RecurrenceRule[]	Event, Task	Clause 6.4.3
expectReply	Boolean	Participant	Clause 6.5.5
features	String[Boolean]	VirtualLocation	Clause 6.3
firstDayOfWeek	String	RecurrenceRule	Clause 6.4.2
freeBusyStatus	String	Event, Task	Clause 6.5.2
frequency	String	RecurrenceRule	Clause 6.4.2
href	String	Link	Link
interval	UnsignedInt	RecurrenceRule	Clause 6.4.2
invitedBy	String	Participant	Clause 6.5.5
keywords	String[Boolean]	Event, Task, Group	Clause 6.3.3

Property Name	Property Type	Property Context	Reference or Description
kind	String	Participant	Clause 6.5.5
language	String	Participant	Clause 6.5.5
links	Id[Link]	Group, Event, Task, Location, Participant	Clause 6.3.1, Clause 6.2.5, Clause 6.5.5
locale	String	Group, Event, Task	Clause 6.3.2
localizations	String[PatchObject]	Event, Task	Clause 6.7.1
locationId	String	Participant	Clause 6.5.5
locations	Id[Location]	Event, Task	Clause 6.2.5
locationTypes	String[Boolean]	Location	Clause 6.2.5
memberOf	String[Boolean]	Participant	Clause 6.5.5
method	String	Event, Task	Clause 6.1.8
name	String	Location, VirtualLocation, Participant	Clause 6.2.5, Clause 6.3, Clause 6.5.5
names	String[Boolean]	TimeZoneRule	Clause 6.8.2
nthOfPeriod	Int	NDay	Clause 6.4.2
offset	SignedDuration	OffsetTrigger	Clause 6.6.2
offsetFrom	UTCDateTime	TimeZoneRule	Clause 6.8.2
offsetTo	UTCDateTime	TimeZoneRule	Clause 6.8.2
participants	Id[Participant]	Event, Task	Clause 6.5.5
participationComment	String	Participant	Clause 6.5.5
participationStatus	String	Participant	Clause 6.5.5
percentComplete	UnsignedInt	Task, Participant	Clause 7.2.4
priority	Int	Event, Task	Clause 6.5.1
privacy	String	Event, Task	Clause 6.5.3
prodId	String	Event, Task, Group	Clause 6.1.4
progress	String	Task, Participant	Clause 7.2.5

Property Name	Property Type	Property Context	Reference or Description
progressUpdated	UTCDateTime	Task, Participant	Clause 7.2.6
recurrenceId	LocalDateTime	Event, Task	Clause 6.4.1
recurrenceOverrides	LocalDateTime[PatchObject]	Event, Task, TimeZoneRule	Clause 6.4.4, Clause 6.8.2
recurrenceRules	RecurrenceRule[]	Event, Task, TimeZoneRule	Clause 6.4.2, Clause 6.8.2
rel	String	Link	Link
relatedTo	String[Relation]	Event, Task, Alert	Clause 6.1.3, Clause 6.6.2
relation	String[Boolean]	Relation	Relation
relativeTo	String	OffsetTrigger, Location	Clause 6.6.2, Clause 6.2.5
replyTo	String[String]	Event, Task	Clause 6.5.4
requestStatus	String	Event, Task	Clause 6.5.6
roles	String[Boolean]	Participant	Clause 6.5.5
rscale	String	RecurrenceRule	Clause 6.4.2
standard	TimeZoneRule[]	TimeZone	Clause 6.8.2
start	LocalDateTime	TimeZoneRule	Clause 6.8.2
scheduleAgent	String	Participant	Clause 6.5.5
scheduleForceSend	Boolean	Participant	Clause 6.5.5
scheduleSequence	UnsignedInt	Participant	Clause 6.5.5
scheduleStatus	String[]	Participant	Clause 6.5.5
scheduleUpdated	UTCDateTime	Participant	Clause 6.5.5
sendTo	String[String]	Participant	Clause 6.5.5
sequence	UnsignedInt	Event, Task	Clause 6.1.7
showWithoutTime	Boolean	Event, Task	Clause 6.2.4
size	UnsignedInt	Link	Link
skip	String	RecurrenceRule	Clause 6.4.2
source	String	Group	Clause 7.3.2
start	LocalDateTime	Event, Task	Clause 7.1.1,

Property Name	Property Type	Property Context	Reference or Description
status	String	Event	Clause 7.2.2
timeZone	TimeZoneId|null	Event, Task, Location	Clause 7.1.3
timeZones	TimeZoneId[TimeZone]	Event, Task	Clause 6.8.1, Clause 6.2.5
title	String	Event, Task, Group, Link	Clause 6.8.2
trigger	OffsetTrigger|AbsoluteTrigger|UnknownTrigger	Alert	Clause 6.2.1
tzId	String	TimeZone	Clause 6.6.2
uid	String	Event, Task, Group	Clause 6.8.2
until	LocalDateTime	RecurrenceRule	Clause 6.1.2
updated	UTCDateTime	Event, Task, Group	Clause 6.4.2
uri	String	VirtualLocation	Clause 6.1.6
url	String	TimeZone	Clause 6.3
useDefaultAlerts	Boolean	Event, Task	Clause 6.8.2
validUntil	UTCDateTime	TimeZone	Clause 6.6.1
virtualLocations	Id[VirtualLocation]	Event, Task	Clause 6.8.2
when	UTCDateTime	AbsoluteTrigger	Clause 6.3
			Clause 6.6.2

10.9. Creation of "JSCalendar Types" Registry

The IANA will create the "JSCalendar Types" registry to avoid name collisions and provide a complete reference for all data types used for JSCalendar property values. The registration process is the same as for the JSCalendar Properties registry, as defined in Clause 10.2.

10.9.1. JSCalendar Types Registry Template

— Type Name: The name of the type.
— Reference or Description: A brief description or RFC number and section reference where the Type is specified (may be omitted for "reserved" type names).
— Intended Use: Common, reserved, or obsolete.
— Change Controller: Who may request a change to this entry's definition (`IETF` for IETF-stream RFCs).

10.9.2. Initial Contents for the JSCalendar Types Registry

The following table lists the initial entries of the JSCalendar Types registry. All properties are for common-use. All RFC section references are for this document. The change controller for all these properties is "IETF".

Table 3

Type Name	Reference or Description
Alert	Clause 6.6.2
Boolean	Type Signatures
Duration	Duration
Id	Id
Int	Int
LocalDateTime	LocalDateTime
Link	Link
Location	Clause 6.2.5
NDay	Clause 6.4.2
Number	Type Signatures
Participant	Clause 6.5.5
PatchObject	PatchObject
RecurrenceRule	Clause 6.4.2
Relation	Relation
SignedDuration	SignedDuration
String	Type Signatures
TimeZone	Clause 6.8.2
TimeZoneId	TimeZoneId
TimeZoneRule	Clause 6.8.2
UnsignedInt	UnsignedInt
UTCDateTime	UTCDateTime
VirtualLocation	Clause 6.3

10.10. Creation of "JSCalendar Enum Values" Registry

The IANA will create the "JSCalendar Enum Values" registry to allow interoperable extension of semantics for properties with enumerable values. Each such property will have a subregistry of allowed values. The registration process for a new enum value or adding a new enumerable property is the same as for the JSCalendar Properties registry, as defined in Clause 10.2.

10.10.1. JSCalendar Enum Property Template

This template is for adding a subregistry for a new enumerable property to the JSCalendar Enum registry.

— Property Name: the name(s) of the property or properties where these values may be used. This MUST be registered in the JSCalendar Properties registry.
— Context: the list of allowed object types where the property or properties may appear, as registered in the JSCalendar Properties registry. This disambiguates where there may be two distinct properties with the same name in different contexts.
— Change Controller: (`IETF` for properties defined in IETF-stream RFCs).
— Initial Contents: The initial list of defined values for this enum, using the template defined in Clause 10.10.2. A subregistry will be created with these values for this property name/context tuple.

10.10.2. JSCalendar Enum Value Template

This template is for adding a new enum value to a subregistry in the JSCalendar Enum registry.

— Enum Value: The verbatim value of the enum.
— Reference or Description: A brief description or RFC number and section reference for the semantics of this value.

10.10.3. Initial Contents for the JSCalendar Enum Values registry

For each subregistry created in this section, all RFC section references are for this document.

Property Name: action

Context: Alert

Change Controller: IETF

Initial Contents: **Table 4**

Enum Value	Reference or Description
display	Clause 6.6.2
email	Clause 6.6.2

Property Name: display

Context: Link

Change Controller: IETF

Initial Contents: **Table 5**

Enum Value	Reference or Description
badge	Link
graphic	Link
fullsize	Link
thumbnail	Link

Property Name: features

Context: VirtualLocation

Change Controller: IETF

Initial Contents: **Table 6**

Enum Value	Reference or Description
audio	Clause 6.3
chat	Clause 6.3
feed	Clause 6.3
moderator	Clause 6.3
phone	Clause 6.3
screen	Clause 6.3
video	Clause 6.3

Property Name: freeBusyStatus

Context: Event, Task

Change Controller: IETF

Initial Contents: **Table 7**

Enum Value	Reference or Description
free	Clause 6.5.2
busy	Clause 6.5.2

Property Name: kind

Context: Participant

Change Controller: IETF

Initial Contents

Table 8

Enum Value	Reference or Description
individual	Clause 6.5.5
group	Clause 6.5.5
resource	Clause 6.5.5
location	Clause 6.5.5

Property Name: participationStatus

Context: Participant

Change Controller: IETF

Initial Contents

Table 9

Enum Value	Reference or Description
needs-action	Clause 6.5.5
accepted	Clause 6.5.5
declined	Clause 6.5.5
tentative	Clause 6.5.5
delegated	Clause 6.5.5

Property Name: privacy

Context: Event, Task

Change Controller: IETF

Initial Contents

Table 10

Enum Value	Reference or Description
public	Clause 6.5.3
private	Clause 6.5.3
secret	Clause 6.5.3

Property Name: progress

Context: Task, Participant

Change Controller: IETF

Initial Contents

Table 11

Enum Value	Reference or Description
needs-action	Clause 7.2.5
in-process	Clause 7.2.5
completed	Clause 7.2.5
failed	Clause 7.2.5
cancelled	Clause 7.2.5

Property Name: relation

Context: Relation

Change Controller: IETF

Initial Contents

Table 12

Enum Value	Reference or Description
first	Relation
next	Relation
child	Relation
parent	Relation

Property Name: relativeTo

Context	OffsetTrigger, Location
Change Controller	IETF
Initial Contents	**Table 13**

Enum Value	Reference or Description
start	Clause 6.6.2
end	Clause 6.6.2

Property Name	roles
Context	Participant
Change Controller	IETF
Initial Contents	**Table 14**

Enum Value	Reference or Description
owner	Clause 6.5.5
attendee	Clause 6.5.5
optional	Clause 6.5.5
informational	Clause 6.5.5
chair	Clause 6.5.5
contact	Clause 6.5.5

Property Name	scheduleAgent
Context	Participant
Change Controller	IETF
Initial Contents	**Table 15**

Enum Value	Reference or Description
server	Clause 6.5.5
client	Clause 6.5.5
none	Clause 6.5.5

Property Name	status
Context	Event
Change Controller	IETF
Initial Contents	**Table 16**

Enum Value	Reference or Description
confirmed	Clause 7.1.3
cancelled	Clause 7.1.3
tentative	Clause 7.1.3

11. Acknowledgments

The authors would like to thank the members of CalConnect for their valuable contributions. This specification originated from the work of the API technical committee of CalConnect, the Calendaring and Scheduling Consortium.

Bibliography

[1] IETF RFC 6376, D. CROCKER, T. HANSEN and M. KUCHERAWY (eds.). *DomainKeys Identified Mail (DKIM) Signatures*. 2011. RFC Publisher. https://www.rfc-editor.org/info/rfc6376.

[2] IETF RFC 7265, P. KEWISCH, C. DABOO and M. DOUGLASS. *jCal: The JSON Format for iCalendar*. 2014. RFC Publisher. https://www.rfc-editor.org/info/rfc7265.

[3] IETF RFC 7986, C. DABOO. *New Properties for iCalendar*. 2016. RFC Publisher. https://www.rfc-editor.org/info/rfc7986.

[4] LOCATIONTYPES, IANA Location Types Registry, https://www.iana.org/assignments/location-type-registry/location-type-registry.xhtml

[5] LINKRELS, IANA Link Relation Types, https://www.iana.org/assignments/link-relations/link-relations.xhtml

[6] MEDIATYPES, IANA Media Types, https://www.iana.org/assignments/media-types/media-types.xhtml

