

Calendaring extensions
Internet-Draft
Obsoletes: 8984 (if approved)
Intended status: Standards Track
Expires: 19 April 2026

N. Jenkins
R. Stepanek
Fastmail
16 October 2025

JSCalendar: A JSON Representation of Calendar Data
draft-ietf-calext-jscalendarbis-10

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. It also aims to be unambiguous, extendable, and simple to process. In contrast to the jCal format, which is also based on JSON, JSCalendar is not a direct mapping from iCalendar but defines the data model independently and expands semantics where appropriate.

Note

This note is to be removed before publishing as an RFC.

Differences from RFC 8984 are documented in Appendix A.

Status of This Memo

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

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

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

This Internet-Draft will expire on 19 April 2026.

Copyright Notice

Copyright (c) 2025 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction	5
1.1. Motivation and Relation to iCalendar and jCal	5
1.2. Notational Conventions	6
1.3. Type and Property Notation	7
1.3.1. Type Signatures	7
1.3.2. Property Attributes	7
1.3.3. The @type Property	8
1.4. Data Types	8
1.4.1. Id	9
1.4.2. Int	9
1.4.3. UnsignedInt	9
1.4.4. UTCDateTime	9
1.4.5. LocalDateTime	10
1.4.6. Duration	10
1.4.7. SignedDuration	12
1.4.8. TimeZoneId	12
1.4.9. PatchObject	12
1.4.10. Relation	13
1.4.11. Link	14
2. JSCalendar Objects	15
2.1. Event	15
2.2. Task	15
2.3. Group	15
3. Structure of JSCalendar Objects	16
3.1. Object Type	16
3.2. Normalization and Equivalence	16
3.3. Vendor-Specific Property Extensions, Values, and Types	17
4. Common JSCalendar Properties	17
4.1. Metadata Properties	17
4.1.1. uid	17
4.1.2. relatedTo	18
4.1.3. prodId	18
4.1.4. created	18
4.1.5. updated	19
4.1.6. sequence	19
4.1.7. method	19
4.2. What and Where Properties	19

4.2.1.	title	19
4.2.2.	description	19
4.2.3.	descriptionContentType	19
4.2.4.	showWithoutTime	20
4.2.5.	locations	20
4.2.6.	mainLocationId	21
4.2.7.	virtualLocations	21
4.2.8.	links	22
4.2.9.	locale	22
4.2.10.	keywords	22
4.2.11.	categories	23
4.2.12.	color	23
4.3.	Recurrence Properties	23
4.3.1.	recurrenceId	23
4.3.2.	recurrenceIdTimeZone	24
4.3.3.	recurrenceRule	24
4.3.4.	recurrenceOverrides	32
4.4.	Sharing and Scheduling Properties	33
4.4.1.	priority	33
4.4.2.	freeBusyStatus	34
4.4.3.	privacy	34
4.4.4.	organizerCalendarAddress	35
4.4.5.	participants	36
4.5.	Alerts Properties	40
4.5.1.	alerts	40
4.6.	Time Zone Properties	42
4.6.1.	timeZone	42
5.	Type-Specific JSCalendar Properties	42
5.1.	Event Properties	42
5.1.1.	start	42
5.1.2.	duration	42
5.1.3.	endTimeZone	43
5.1.4.	status	43
5.2.	Task Properties	43
5.2.1.	due	43
5.2.2.	start	43
5.2.3.	estimatedDuration	44
5.2.4.	percentComplete	44
5.2.5.	progress	44
5.3.	Group Properties	45
5.3.1.	entries	45
5.3.2.	source	45
6.	Examples	46
6.1.	Simple Event	46
6.2.	Simple Task	46
6.3.	Simple Group	46
6.4.	All-Day Event	47
6.5.	Task with a Due Date	47

6.6.	Event with End Time Zone	48
6.7.	Floating-Time Event (with Recurrence)	48
6.8.	Event with Physical and Virtual Location	49
6.9.	Recurring Event with Overrides	50
6.10.	Making a "This and Future" Change	51
6.11.	Recurring Event with Participants	54
7.	Security Considerations	56
7.1.	Expanding Recurrences	56
7.2.	JSON Parsing	57
7.3.	URI Values	57
7.4.	Spam	58
7.5.	Duplication	58
7.6.	Time Zones	58
8.	IANA Considerations	58
8.1.	Media Type Registration	59
8.2.	"JSCalendar Properties" Registry	60
8.2.1.	Registry Policy and Change Procedures	60
8.2.2.	"JSCalendar Properties" Registry Template	62
8.2.3.	Changes to the JSCalendar Properties Registry contents	63
8.3.	"JSCalendar Types" Registry	71
8.3.1.	"JSCalendar Types" Registry Template	71
8.3.2.	Changes to the JSCalendar Types Registry contents	72
8.4.	"JSCalendar Enum Values" Registry	74
8.4.1.	"JSCalendar Enum Values" Registry Property Template	74
8.4.2.	"JSCalendar Enum Values" Registry Value Template	75
8.4.3.	Changes to the JSCalendar Enum Values Registry contents	75
9.	References	78
9.1.	Normative References	78
9.2.	Informative References	81
Appendix A.	Differences from RFC 8984	82
A.1.	Applied Errata	82
A.2.	Obsolete Properties	82
A.3.	Reserved Properties	85
A.4.	Updated Properties	87
A.5.	New Properties	90
A.6.	Obsolete Types	92
A.7.	Updated Types	92
Acknowledgments	92
Authors' Addresses	92

1. 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 modeled 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 [RFC5545] [RFC7986] 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 Internet JSON (I-JSON) format [RFC7493], which is a strict subset of the JSON data interchange format [RFC8259]. 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.

1.1. Motivation and Relation to iCalendar and jCal

The iCalendar data format [RFC5545], a widely deployed interchange format for calendaring and scheduling data, has served calendaring vendors for a long time but contains some ambiguities and pitfalls that cannot be overcome without backward-incompatible changes.

Sources of implementation errors include the following:

- * iCalendar defines various formats for local times, UTC, 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 interpretations, 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 [RFC7265], 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 time zone'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.

1.2. Notational Conventions

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

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 Section 1 of [RFC8259].

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.

1.3. Type and Property Notation

This section defines the notation for JSCalendar properties and types.

1.3.1. 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`: This is the JSON string type.

`Number`: This is the JSON number type.

`Boolean`: This is the JSON boolean type.

`A`: The value is of type A.

`A[B]`: The keys are all of type A and the values are all of type B for a JSON object.

`A[]`: There is 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; their representations are defined elsewhere in this document.

1.3.2. Property Attributes

Object properties may also have a set of attributes defined along with the type signature. These have the following meanings:

`mandatory`: The property **MUST** be set for an instance of this object to be valid.

`optional`: The property can, but need not, be set for an instance of this object to be valid.

`default`: This is followed by a JSON value. That value will be used for this property if it is omitted.

`defaultType`: This is followed by the name of a JSCalendar object type. A property value of JSCalendar object type is expected to be of this named type, in case it omits the `@type` property.

1.3.3. The @type Property

Every JSCalendar object has the following JSON object member:

@type: String This is the JSCalendar type of a JSON object. It MUST match the type name of the JSCalendar object of which the JSON object is an instance of.

The purpose of the @type property is to help implementations identify which JSCalendar object type a given JSON object represents. Implementations MUST validate that JSON objects with this property conform to the specification of the JSCalendar object type of that name.

In many cases, the @type property value is implied by where the object occurs in JSCalendar data. Assuming that both A and B are JSCalendar object types:

- * An object that is set as the value for a property with type signature A MAY have the @type property set. If the @type property is not set, then its value is implied to be A.
- * An object that is set as the value for a property with type signature A|B (defaultType: A) MAY have the @type property set if it is an instance of A. It MUST have the @type property set if it is an instance of B. If instead the defaultType attribute is not defined, then the @type property MUST also be set for A.
- * An object that is not the value of a property, such as the topmost object in JSON data (directly or as a member of an array), MUST have the @type property set.

An object definition MAY require the @type property to be set regardless of the above definitions, e.g. this document requires the @type property to be set for the Event, Task and Group object types.

1.4. Data Types

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

1.4.1. Id

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

In many places in 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.

1.4.2. Int

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

1.4.3. UnsignedInt

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

1.4.4. UTCDateTime

This is a string in the date-time [RFC3339] 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.

For example, 2010-10-10T10:10:10Z is conformant, but 2010-10-10T10:10:10.0Z or 2010-10-10T10:10:10.123Z are invalid.

1.4.5. LocalDateTime

This is a date-time string with no time zone/offset information. It is otherwise in the same format as UTCDateTime. For example, 2006-01-02T15:04:05 is valid. The time zone to associate with the LocalDateTime comes from the timeZone property of the JSCalendar object (see Section 4.6.1). 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 daylight savings time (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), 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 Section 1.4.6) 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.

1.4.6. Duration

Where Duration is given as a type, it means a length of time represented by a subset of the ISO 8601 duration format, as specified by the following ABNF [RFC5234]:

```
dur-second = 1*DIGIT "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)
```

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 seven days.
2. If a time zone applies to the `LocalDateTime`, convert it to a `UTCDateTime` following the semantics in Section 1.4.5.
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 Section 1.4.5.
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 Section 1.4.5 to convert to UTC and back to get a valid `LocalDateTime`.

These semantics match the iCalendar DURATION value type ([RFC5545], Section 3.3.6).

1.4.7. 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 indicates a time at or after the associated time.

1.4.8. TimeZoneId

Where TimeZoneId is given as a data type, it means a String that is a time zone name in the IANA Time Zone Database [TZDB]. The zone rules of the respective IANA time zone records apply.

1.4.9. 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 the JSON Pointer format [RFC6901]. The paths have an implicit leading /, so each key is prefixed with / before applying the JSON Pointer evaluation algorithm.

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

1. The pointer MUST NOT reference inside an array (i.e., you MUST NOT insert/delete from an array; the array MUST be replaced in its entirety instead).
2. All parts prior to the last (i.e., the value after the final slash) MUST already exist on the object being patched.
3. There MUST NOT be two patches in the PatchObject where the pointer of one is the prefix of the pointer of the other, e.g., 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 Section 1.3.3). An @type property in a patch MUST be handled as any other patched property value.

Implementations MUST reject a PatchObject in its entirety if any of its patches are 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.

1.4.10. 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

This specifies the type of this object. This MUST be Relation, if set.

relation: String[Boolean] (optional, default: empty Object)

This describes how the linked object is related to the linking object. The relation is defined as a set of relation types. Keys in the set MUST be one of the following values, specified in the property definition where the Relation object is used, a value registered in the IANA "JSCalendar Enum Values" registry, or a vendor-specific value (see Section 3.3):

first: The linked object is the first in a series the linking object is part of.

next: The linked object is 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. The empty Object value represents a "parent" relation, unless defined differently for a specific property.

1.4.11. Link

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

@type: String

This specifies the type of this object. This MUST be Link, if set.

href: String (mandatory)

This is a URI[RFC3986] from which the resource may be fetched.

This MAY be a data: URL [RFC2397], but it is recommended that the file be hosted on a server to avoid embedding arbitrarily large data in JSCalendar object instances.

contentType: String (optional)

This is the media type [RFC6838] of the resource, if known.

size: UnsignedInt (optional)

This is 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)

This identifies the relation of the linked resource to the object. If set, the value MUST be a link relation type as defined in Section 2.1 of [RFC8288] .

display: String[Boolean] (optional)

This is a set of intended purposes of a link to an image. The keys MUST be one of the following values, another value registered in the IANA "JSCalendar Enum Values" registry, or a vendor-specific value (see Section 3.3):

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

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

title: String (optional)

This is a human-readable, plain-text description of the resource.

2. JSCalendar Objects

This section describes the calendar object types specified by JSCalendar.

2.1. Event

Media type: application/jscalendar+json;type=event

An 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.

The @type (Section 1.3.3) property is mandatory, it MUST be set to value Event.

2.2. Task

Media type: application/jscalendar+json;type=task

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

The @type (Section 1.3.3) property is mandatory, it MUST be set to value Task.

2.3. Group

Media type: application/jscalendar+json;type=group

A Group is a collection of Event (Section 2.1) and/or Task (Section 2.2) objects. Typically, objects are grouped by topic (e.g., by keywords) or calendar membership.

The @type (Section 1.3.3) property is mandatory, it MUST be set to value Group.

3. Structure of JSCalendar Objects

A JSCalendar object is a JSON object [RFC8259], which MUST be valid I-JSON (a stricter subset of JSON) [RFC7493]. 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.

3.1. 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 (Section 1.4.9).

3.2. 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 Section 6 of [RFC3986]).

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.

3.3. 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 an `@type` property, or it MUST explicitly be specified to not require a type designator. The type name MUST be prefixed with a domain name controlled by the vendor.

Some 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.

4. 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 Section 5 describe the set of supported properties per type.

4.1. Metadata Properties

4.1.1. uid

Type: String (mandatory)

This is a globally unique identifier used to associate objects representing the same event, task, group, or other object across different systems, calendars, and views. For recurring events and tasks, the UID is associated with the base object and therefore is the same for all occurrences; the combination of the UID with a `recurrenceId` identifies a particular instance.

The generator of the identifier MUST guarantee that the identifier is unique. [RFC9562] describes a range of established algorithms to generate universally unique identifiers (UUIDs). UUID version 4, described in Section 4.4 of [RFC9562], is RECOMMENDED.

For compatibility with UIDs [RFC5545], 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.

4.1.2. relatedTo

Type: String[Relation] (optional)

This relates the object to other JSCalendar objects. Each key in the map is the uid property value of a related object. The value defines the type of 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 is created to represent all the occurrences after the split. A next relation MUST be set on the original object's relatedTo property, keyed by the uid of the new object. A first relation MUST be set on the new object, keyed by the uid of the first object in the series. Clients can then follow these uid keys to get the complete set of objects if the user wishes to modify them all at once.

4.1.3. prodId

Type: String (optional)

This is 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 a Formal Public Identifier (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 nonstandard properties, a practice that is known to cause long-term interoperability problems.

4.1.4. created

Type: UTCDateTime (optional)

This is the date and time this object was initially created.

4.1.5. updated

Type: `UTCDateTime` (mandatory)

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

4.1.6. sequence

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

This property indicates the revision of the calendar object. For scheduled calendar objects, its value must be incremented according to the rules of the scheduling protocol, e.g. [jmap-calendars] or [RFC5546].

4.1.7. method

Type: `String` (optional)

This is the iTIP [RFC5546] method, in lowercase. This MUST only be present if the JSCalendar object represents an iTIP scheduling message.

4.2. What and Where Properties

4.2.1. title

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

This is a short summary of the object.

4.2.2. description

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

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

4.2.3. descriptionContentType

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

This describes the media type [RFC6838] of the contents of the `description` property. Media types MUST be subtypes 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.

4.2.4. showWithoutTime

Type: Boolean (optional, default: false)

This 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.

4.2.5. locations

Type: Id[Location] (optional)

This is 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 @type property.

@type: String

This specifies the type of this object. This MUST be Location, if set.

name: String (optional)

This is the human-readable name or short description of the location, such as an address.

locationTypes: String[Boolean] (optional)

This is 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 [RFC4589]. 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.

coordinates: String (optional)

This is a geo: URI [RFC5870] for the location.

links: Id[Link] (optional)

This is 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).

4.2.6. mainLocationId

Type: String (optional)

This indicates which of the multiple entries in the locations property can be considered the main location for the event or task. A client implementation MAY choose to display this location more prominently. The main location is undefined if this property is not set. If this property is set, then its value MUST match a key in the locations property and the name property of that main Location object MUST be set.

4.2.7. virtualLocations

Type: Id[VirtualLocation] (optional)

This is 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

This specifies the type of this object. This MUST be VirtualLocation, if set.

name: String (optional, default: empty String)

This is the human-readable name or short description of the virtual location, such as an access code.

uri: String (mandatory)

This is a URI[RFC3986] 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 Section 3.3). 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

4.2.8. links

Type: Id[Link] (optional)

This is 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 this image.

4.2.9. locale

Type: String (optional)

This is the language tag, as defined in [RFC5646], that best describes the locale used for the text in the calendar object, if known.

4.2.10. keywords

Type: String[Boolean] (optional)

This is 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.

4.2.11. categories

Type: String[Boolean] (optional)

This is 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 are typically 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`.

4.2.12. color

Type: String (optional)

This is 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 CSS Color Module Level 3 [COLORS] or an RGB value in hexadecimal notation, as defined in Section 4.2.1 of CSS Color Module Level 3.

4.3. 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 base event with a rule to generate recurrences and/or overrides that add extra dates or exceptions to the rule.

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 `recurrenceRule` property (Section 4.3.3) generates a set of extra date-times on which the object occurs.
2. The `recurrenceOverrides` property (Section 4.3.4) defines date-times that are added or excluded to form the final set. (This property may also contain changes to the object to apply to particular instances.)

4.3.1. recurrenceId

Type: LocalDateTime (optional)

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

The value is a date-time either produced by the recurrenceRule of the base event or added as a key to the recurrenceOverrides property of the base event.

4.3.2. recurrenceIdTimeZone

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

Identifies the time zone of the main JSCalendar object, of which this JSCalendar object is a recurrence instance. It MUST NOT be set if the recurrenceId property is not set.

4.3.3. recurrenceRule

Type: RecurrenceRule (optional)

This a recurrence rule (a repeating pattern) for recurring calendar objects.

An Event recurs by applying the recurrence rule to the start date-time.

A Task recurs by applying the recurrence rule 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 recurrenceRule property.

A RecurrenceRule object is a JSON object mapping of a RECUR value type in iCalendar [RFC5545] [RFC7529] and has the same semantics. It has the following properties:

@type: String

This specifies the type of this object. This MUST be RecurrenceRule, if set.

frequency: String (mandatory)

This is the time span covered by each iteration of this recurrence rule (see Section 4.3.3.1 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)

This is 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")

This is 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 Section 3.3).

This is the `RSCALE` part from iCalendar `RSCALE` [RFC7529], converted to lowercase.

`skip`: `String` (optional, default: "omit")

This is the behavior 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` [RFC7529], converted to lowercase.

`firstDayOfWeek`: `String` (optional, default: "mo")

This is the day on which the week is considered to start, represented as a lowercase, abbreviated, and 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)

These are days of the week on which to repeat. An NDay object has the following properties:

@type: String

This specifies the type of this object. This MUST be NDay, if set.

day: String (mandatory)

This is 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 the nth-last occurrence within that period (i.e., -1 is the last occurrence, -2 the one before that, etc.).

This is the ordinal part of the BYDAY value in iCalendar (e.g., 1 or -3).

byMonthDay: Int[] (optional)

These are the 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)

These are 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 [RFC7529]) 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)

These are 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)

These are the 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)

These are 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)

These are 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: UInt[]` (optional)

These are 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)

These are 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: UInt` (optional)

These are 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)

These are 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 that 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.

4.3.3.1. Interpreting Recurrence Rules

A recurrence rule specifies a set of date-times for recurring calendar objects. A recurrence rule has the following semantics. Note that 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 first day of a year (inclusive) to midnight the first 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 first day of a month (inclusive) to midnight on the first 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 seven 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: only the second itself.

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 mean the 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.

If a skip property is defined and is not "omit", there may be candidates that do not correspond to valid dates (e.g., February 31st in the Gregorian calendar). In this case, the properties MUST be considered in the order above, and:

1. 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.
2. 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").
3. If any valid date produced after applying the skip is already a candidate, eliminate the duplicate. (For example, after adjusting, February 30th and February 31st would both become the same "real" date, so one is eliminated as a duplicate.)

3. 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 indexed from the end of the list, with -1 being the last item, -2 the second from last, etc.
4. Any date-times before the start date of the event are eliminated (see below for why this might be needed).
5. 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".)
6. 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 or 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 of the list above.
3. The following properties MUST be implicitly added to the rule under the given conditions:
 - * If frequency is not secondly and there is 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 there is 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 there is 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 there is 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 there is 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 there is 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 are 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.

4.3.4. 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 occurrence generated by the recurrence id shall be omitted from the final set of recurrences (like an EXDATE from iCalendar), then the patch object MUST be a JSON object with a single member. The member name MUST be "excluded", the member value MUST be true. The JSON object MUST NOT contain any other members.

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`
- * `method`
- * `organizerCalendarAddress`
- * `participants/*/calendarAddress`
- * `privacy`
- * `prodId`
- * `recurrenceId`
- * `recurrenceIdTimeZone`
- * `recurrenceOverrides`
- * `recurrenceRule`
- * `relatedTo`
- * `uid`

4.4. Sharing and Scheduling Properties

4.4.1. `priority`

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

This 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.

4.4.2. freeBusyStatus

Type: String (optional, default: busy)

This 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 Section 3.3):

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.

4.4.3. 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 is 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 Section 3.3). 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 are shared. The following properties MAY be shared; any other properties MUST NOT be shared:

- * @type
- * created
- * due
- * duration
- * estimatedDuration
- * freeBusyStatus
- * privacy
- * recurrenceId
- * recurrenceIdTimeZone
- * recurrenceOverrides (Only patches that apply to another permissible property are allowed to be shared.)
- * recurrenceRule
- * sequence
- * showWithoutTime
- * start
- * timeZone
- * uid
- * updated

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

4.4.4. organizerCalendarAddress

Type: String (optional)

This is a URI as defined by [RFC3986] or any other IANA-registered form for a URI. It is the same as the CAL-ADDRESS value of an iCalendar ORGANIZER property [RFC5545] (Section 3.8.4.3) — it globally identifies a particular organizer, even across different calendaring objects.

4.4.5. participants

Type: Id[Participant] (optional)

This is a map of participant ids to participants, describing their participation in the calendar object.

If this property is set and any participant has a calendarAddress property set, then the organizerCalendarAddress property of this calendar object MUST be set.

A Participant object has the following properties:

@type: String

This specifies the type of this object. This MUST be Participant, if set.

name: String (optional)

This is the display name of the participant (e.g., "Joe Bloggs").

email: String (optional)

This is the email address to use to contact the participant or, for example, match with an address book entry. If set, the value MUST be a valid addr-spec value as defined in Section 3.4.1 of [RFC5322] .

description: String (optional)

This is a description of this participant. For example, this may include more information about their role in the event or how best to contact them.

descriptionContentType: String (optional)

This describes the media type of the contents of the description property. Its requirements are specified in Section 4.2.3. If this property is set, then the description property MUST be set.

`calendarAddress`: String (optional)

This is a URI as defined by [RFC3986] or any other IANA-registered form for a URI. It is the same as the `CAL-ADDRESS` value of an iCalendar `ATTENDEE` property [RFC5545] (Section 3.8.4.1) or `ORGANIZER` property [RFC5545] (Section 3.8.4.3) — it globally identifies a particular participant, even across different calendaring objects.

`kind`: String (optional)

This is what kind of entity this participant is, if known. If this property is set, then the `calendarAddress` property **MUST** be set. 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 Section 3.3). 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] (optional)

This is a set of roles that this participant fulfills. If this property is set, then the `calendarAddress` property **MUST** be set. 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 Section 3.3):

`owner`: The participant is an owner of the calendar object. This signifies they can make changes that affect all participants (for example, rescheduling the calendar object, adding and removing participants and roles). The presence of this role only is indicative, its semantics are subject to the calendaring exchange protocol being used. See [jmap-calendars] for an example for making use of this role.

`optional`: The participant's involvement with the event is optional.

`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.

required: The participant is required to be present at the event.

The value for each key in the map MUST be true. It is expected that no more than one of the roles "required", "optional", "informational" and "chair" be present; if more than one are given, they take precedence as follows: "chair" over "required", either of them over "optional", all of them over "informational". Roles that are unknown to the implementation MUST be preserved.

participationStatus: String (optional, default: needs-action)
This is the participation status, if any, of this participant. If this property is set, then the calendarAddress property MUST be set.

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 Section 3.3):

needs-action: No status has yet been 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.

expectReply: Boolean (optional, default: false)
If true, the organizer is expecting the participant to notify them of their participation status. If this property is set, then the calendarAddress property MUST be set.

sentBy: String (optional)
This is the email address in the "From" header of the email that last updated this participant via iMIP. If this property is set, then the calendarAddress property MUST be set. This SHOULD only be set if the email address is different to that in the mailto URI of this participant's calendarAddress property (i.e., the response was received from a different address to that which the invitation was sent to). If set, the value MUST be a valid addr-spec value as defined in Section 3.4.1 of [RFC5322] .

`delegatedTo: String[Boolean]` (optional)

This is the set of participants that this participant has delegated their participation to. If this property is set, then the `calendarAddress` property MUST be set. Each key in the set MUST be a URI according to the definition of the `calendarAddress` property. 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)

This is a set of participants that this participant is acting as a delegate for. If this property is set, then the `calendarAddress` property MUST be set. Each key in the set MUST be a URI according to the definition of the `calendarAddress` property. 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: Id[Boolean]` (optional)

This is 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). If this property is set, then the `calendarAddress` property MUST be set. Each key in the set MUST be a URI according to the definition of the `calendarAddress` property. 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)

This is 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)

This represents the progress of the participant for this task. If this property is set, then the `calendarAddress` property MUST be set and the `participationStatus` of this participant MUST be accepted. The property value MUST be one of the following values, another value registered in the IANA "JSCalendar Enum Values" registry, or a vendor-specific value (see Section 3.3):

`in-process:` The participant is in process of contributing to the task.

`completed:` The participant successfully completed contributing to the task.

failed: The participant could not complete their contribution to the task.

percentComplete: UnsignedInt (optional; only allowed for participants of a Task)

This represents the percent completion of the participant for this task. The property value MUST be a positive integer between 0 and 100.

4.5. Alerts Properties

4.5.1. alerts

Type: Id[Alert] (optional)

This is 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

This specifies the type of this object. This MUST be Alert, if set.

trigger: OffsetTrigger|AbsoluteTrigger|UnknownTrigger (mandatory, defaultType: OffsetTrigger)

This defines when to trigger the alert. New types may be defined in future documents.

An OffsetTrigger object has the following properties:

@type: String

This specifies the type of this object. This MUST be OffsetTrigger, if set.

offset: SignedDuration (mandatory)

This 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 the time property.

relativeTo: String (optional, default: start)

This specifies the time property that the alert offset is relative to. The value MUST be one of the following:

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

This specifies the type of this object. This MUST be AbsoluteTrigger.

when: UTCDateTime (mandatory)

This defines a specific UTC date-time when the alert is triggered.

An UnknownTrigger object is an object that contains an @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 base 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 is successfully carried out.

relatedTo: String[Relation] (optional)

This relates this alert to other alerts in the same JSCalendar object. Each key in the map is the key of an Alert object in the alerts property. The value defines the type of the relation. In addition to the relation values defined in Section 1.4.10, the following key is allowed:

snooze: The linked alert is snoozed by this alert.

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 "snooze" relation to the identifier of the original alert.

action: String (optional, default: display)
This 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 Section 3.3):

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 them of the alert. This action is typically only appropriate for server implementations.

4.6. Time Zone Properties

4.6.1. timeZone

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

This identifies the time zone the object is scheduled in or is null for floating time. If omitted, this MUST be presumed to be null (i.e., floating time).

5. Type-Specific JSCalendar Properties

5.1. Event Properties

In addition to the common JSCalendar object properties (Section 4), an Event has the following properties:

5.1.1. start

Type: LocalDateTime (mandatory)

This is the date/time the event starts in the event's time zone (as specified in the timeZone property, see Section 4.6.1).

5.1.2. duration

Type: Duration (optional, default: PT0S)

This is 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.

5.1.3. endTimeZone

Type: TimeZoneId (optional)

This identifies the time zone in which this event ends, for cases where the start and time zones of the event differ (e.g., a transcontinental flight). If this property is not set, then the event starts and ends in the same time zone. This property **MUST NOT** be set if the timeZone property value is null or not set.

5.1.4. status

Type: String (optional, default: confirmed)

This is the scheduling status (Section 4.4) of an 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 Section 3.3):

confirmed: indicates the event is definitely happening

cancelled: indicates the event has been cancelled

tentative: indicates the event may happen

5.2. Task Properties

In addition to the common JSCalendar object properties (Section 4), a Task has the following properties.

If the timeZone property value is not null or the showWithoutTime property value is "true", then at least one of the due and start properties **MUST** be set.

5.2.1. due

Type: LocalDateTime (optional)

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

5.2.2. start

Type: LocalDateTime (optional)

This the date/time the task should start in the task's time zone. This MUST be set if the recurrenceRule or recurrenceId properties are set.

5.2.3. estimatedDuration

Type: Duration (optional)

This specifies the estimated positive duration of time the task takes to complete.

5.2.4. percentComplete

Type: UnsignedInt (optional)

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

5.2.5. progress

Type: String (optional)

This defines the progress of this task. If omitted, the default progress (Section 4.4) 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 Section 3.3):

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

5.3. Group Properties

Group supports the following common JSCalendar properties (Section 4):

- * @type
- * uid
- * prodId
- * created
- * updated
- * title
- * description
- * descriptionContentType
- * links
- * locale
- * keywords
- * categories
- * color

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

5.3.1. entries

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

This is a collection of group members. Implementations MUST ignore entries of unknown type.

5.3.2. source

Type: String (optional)

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

6. 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.

6.1. Simple Event

This example illustrates a simple one-time event. It specifies a one-time event that begins on January 15, 2020 at 1 pm 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"
}
```

6.2. 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"
}
```

6.3. 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",
  "title": "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"
  }
]
```

6.4. 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",
  "recurrenceRule": {
    "frequency": "yearly"
  }
}
```

6.5. Task with a Due Date

This example illustrates a task with a due date. It is a reminder to buy groceries before 6 pm 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"
}
```

6.6. 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 9 am 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. The `mainLocationId` property indicates the start location.

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

6.7. 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 7 am local time in whatever time zone the user is located on that date.

```
{
  "...": "",
  "title": "Yoga",
  "start": "2020-01-01T07:00:00",
  "duration": "PT30M",
  "recurrenceRule": {
    "frequency": "daily"
  }
}
```

6.8. Event with Physical and Virtual Location

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. In addition to the main event location, the event contains an additional location for a nearby parking garage.

```
{
  "...": "",
  "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",
  "mainLocationId": "c0503d30-8c50-4372-87b5-7657e8e0fedd",
  "locations": {
    "c0503d30-8c50-4372-87b5-7657e8e0fedd": {
      "name": "The Music Bowl",
      "description": "Music Bowl, Central Park, New York",
      "coordinates": "geo:40.7829,-73.9654"
    },
    "ee42e41e-1046-4489-9760-c0b85f0dc176": {
      "name": "BAZ Parking, 9 West 57th Street, New York",
      "coordinates": "geo:40.7637,-73.9748",
      "locationTypes": {
        "parking": true
      }
    }
  },
  "virtualLocations": {
    "vloc1": {
      "name": "Free live Stream from Music Bowl",
      "uri": "https://stream.example.com/the_band_2020"
    }
  }
}
```

6.9. 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 9 am 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 10 am. Also, the location of the exam differs from the usual location. On April 1, no course is held. On January 7 at 2 pm, there is an optional introduction course, which occurs before the first regular lecture.

```

{
  "...": "",
  "title": "Calculus I",
  "start": "2020-01-08T09:00:00",
  "timeZone": "Europe/London",
  "duration": "PT1H30M",
  "locations": {
    "mlab": {
      "name": "Math lab room 1",
      "description": "Math Lab I, Department of Mathematics"
    }
  },
  "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": {
          "name": "Big Auditorium",
          "description": "Big Auditorium, Other Road"
        }
      }
    }
  }
}

```

6.10. Making a "This and Future" Change

Sometimes, you may want to make a change to a recurring event that applies from a specific instance onwards. This cannot be represented as a single JSCalendar object. Instead, you must duplicate the event, modifying the recurrence rule of the original so it finishes before the split point, and the duplicate so it starts at the split point. A "next" and "first" relation must be set on the new objects respectively, as per Section 4.1.2.

This example shows two JSCalendar objects, representing an event that used to happen at 2pm on a Tuesday in Room 101, but moved in March to 3pm on a Wednesday in Room 202.

```
{
  "...": "",
  "uid": "715ed4c5-3cf5-427f-927c-db40cdd63894",
  "relatedTo": {
    "32859916-af7a-4599-82ed-32a4315b4fe7": {
      "relation": {
        "next": true
      }
    }
  },
  "title": "Departmental meeting",
  "start": "2025-01-07T14:00:00",
  "timeZone": "Australia/Melbourne",
  "duration": "PT1H",
  "locations": {
    "room": {
      "name": "Room 101"
    }
  },
  "recurrenceRule": {
    "frequency": "weekly",
    "until": "2025-02-25T14:00:00"
  }
}
{
  "...": "",
  "uid": "32859916-af7a-4599-82ed-32a4315b4fe7",
  "relatedTo": {
    "715ed4c5-3cf5-427f-927c-db40cdd63894": {
      "relation": {
        "first": true
      }
    }
  },
  "title": "Departmental meeting",
  "start": "2025-03-05T15:00:00",
  "timeZone": "Australia/Melbourne",
  "duration": "PT1H",
  "locations": {
    "room": {
      "name": "Room 202"
    }
  },
  "recurrenceRule": {
    "frequency": "weekly"
  }
}
```

6.11. Recurring Event with Participants

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

```
{
  "...": "",
  "title": "FooBar team meeting",
  "start": "2020-01-08T09:00:00",
  "timeZone": "Africa/Johannesburg",
  "duration": "PT1H",
  "virtualLocations": {
    "0": {
      "name": "ChatMe meeting room",
      "uri": "https://chatme.example.com?id=1234567&pw=a8a24627b63d"
    }
  },
  "recurrenceRule": {
    "frequency": "weekly"
  },
  "organizerCalendarAddress":
    "mailto:f245f875-7f63-4a5e-a2c8@schedule.example.com",
  "participants": {
    "dG9tQGZvb2Jhci5x1LmNvbQ": {
      "name": "Tom Tool",
      "email": "tom@foobar.example.com",
      "calendarAddress": "mailto:tom@calendar.example.com",
      "participationStatus": "accepted",
    },
    "em9lQGZvb2GFtcGx1LmNvbQ": {
      "name": "Zoe Zelda",
      "calendarAddress": "mailto:zoe@foobar.example.com",
      "participationStatus": "accepted",
      "roles": {
        "owner": true,
        "chair": true
      }
    }
  },
  "recurrenceOverrides": {
    "2020-03-04T09:00:00": {
      "participants/dG9tQGZvb2Jhci5x1LmNvbQ/participationStatus":
        "declined"
    }
  }
}
```

7. Security Considerations

Calendar 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 patterns of behavior of the user in both the physical and cyber realm. Additionally, calendar events and tasks can 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 organizer 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 only defines the data format; such considerations are primarily the concern of the API or method of storage and transmission of such files.

7.1. 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, 9 am 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.

7.2. JSON Parsing

The security considerations of [RFC8259] 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.

7.3. URI Values

Several JSCalendar properties contain URIs as values, and processing these properties requires extra care. Section 7 of [RFC3986] 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 often 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.

7.4. 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 DomainKeys Identified Mail (DKIM) [RFC6376], can help establish the source of messages and associate the data with existing relationships (such as an address book contact). However, misclassifications are always possible 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.

7.5. Duplication

It is important for calendar systems to maintain the UID of an event when updating it to avoid an 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.

7.6. 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.

8. IANA Considerations

IANA has created the "JSCalendar Properties", "JSCalendar Types" and "JSCalendar Enum Values" registries, originally defined in [RFC8984]. This document updates some of the registry definitions and registry contents. The following sections redefine all IANA considerations, even if they are unchanged.

8.1. Media Type Registration

[RFC8984] defined a media type for use with JSCalendar data formatted in JSON. This section restates the original definition unchanged.

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:

event: The @type property value MUST be Event.

task: The @type property value MUST be Task.

group: 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: This is the same as the encoding considerations of application/json, as specified in Section 11 of [RFC8259].

Security considerations: See Section 7 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 or ambiguities in interpretation might arise. 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: RFC 8984

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 [RFC6901], 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

8.2. "JSCalendar Properties" Registry

IANA has created the "JSCalendar Properties" registry to allow interoperability of extensions to JSCalendar objects. IANA will set the Reference of the registry to this document, rather than obsoleted [RFC8984].

8.2.1. Registry Policy and Change Procedures

Note for IANA: This section leaves the original definition of [RFC8984] unchanged.

This registry follows the Expert Review process ([RFC8126], Section 4.5). If the "Intended Usage" 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 usage of common, reserved, or obsolete. IANA will list registrations with a common usage designation prominently and separately from those with other intended usage 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 check it is coherent 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.

8.2.1.1. 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.

Property 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.

8.2.1.2. Submit Request to IANA

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

8.2.1.3. 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 Section 4.6 of [RFC8126], 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.

8.2.1.4. Change Procedures

Once a JSCalendar property has been published by 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 usage" field; such properties will be clearly marked in the IANA registry.

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 IANA; this can be done without discussion or review.

8.2.2. "JSCalendar Properties" Registry Template

Note for IANA: This section leaves the original definition of [RFC8984] unchanged.

Property Name: This is 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: This is the type of this property, using type signatures, as specified in Section 1.3.1. The property type **MUST** be registered in the "JSCalendar Types" registry.

Property Context: This is a comma-separated list of JSCalendar object types this property is allowed on.

Reference or Description: This is a brief description or RFC number

and section reference where the property is specified (omitted for "reserved" property names).

Intended Usage: This may be "common", "reserved", or "obsolete".

Change Controller: This is who may request a change to this entry's definition (IETF for RFCs from the IETF stream).

8.2.3. Changes to the JSCalendar Properties Registry contents

8.2.3.1. Obsoleted Properties

IANA will change the Intended Status from "common" to "obsolete" for the following entries in the JSCalendar Properties Registry. For each entry, the Reference/Description column will refer to Appendix A.2 of this document.

- * aliases
- * cid
- * comments
- * daylight
- * excludedRecurrenceRules
- * language
- * locationId
- * names
- * offsetFrom
- * offsetTo
- * progressUpdated
- * recurrenceRules
- * standard
- * timeZones
- * tzId
- * url

- * validUntil

8.2.3.2. Reserved Properties

IANA will change the Intended Status from "common" to "reserved" for the following entries in the JSCalendar Properties Registry. For each entry, the Reference/Description column will refer to Appendix A.3 of this document.

- * excluded
- * invitedBy
- * localizations
- * participationComment
- * replyTo
- * requestStatus
- * scheduleAgent
- * scheduleForceSend
- * scheduleSequence
- * scheduleStatus
- * scheduleUpdated
- * sendTo
- * useDefaultAlerts

8.2.3.3. Removed Properties

IANA will remove the entry having Property Name "start" and Property Context "TimeZoneRule".

8.2.3.4. Updated Properties

IANA will update the following entries in the JSCalendar Properties Registry. For each entry, the Property Name and its changed fields are listed. Unchanged fields are omitted. Referenced section numbers refer to this document, unless otherwise noted.

Property Name @type

Property Context Event, Task, Group, AbsoluteTrigger, Alert, Link,
Location, NDay, OffsetTrigger, Participant, RecurrenceRule,
Relation, VirtualLocation

Reference/Description Section 1.3.3

Property Name acknowledged

Reference/Description Section 4.5.1

Property Name action

Reference/Description Section 4.5.1

Property Name alerts

Reference/Description Section 4.5.1

Property Name byDay

Reference/Description Section 4.3.3

Property Name byHour

Reference/Description Section 4.3.3

Property Name byMinute

Reference/Description Section 4.3.3

Property Name byMonth

Reference/Description Section 4.3.3

Property Name byMonthDay

Reference/Description Section 4.3.3

Property Name bySecond

Reference/Description Section 4.3.3

Property Name bySetPosition

Reference/Description Section 4.3.3

Property Name byWeekNo

Reference/Description Section 4.3.3

Property Name byYearDay

Reference/Description Section 4.3.3

Property Name calendarAddress

Reference/Description Section 4.4.5

Property Name categories

Reference/Description Section 4.2.11

Property Name color

Reference/Description Section 4.2.12

Property Name contentType

Reference/Description Section 1.4.11

Property Name coordinates

Reference/Description Section 4.2.5

Property Name count

Reference/Description Section 4.3.3

Property Name created

Reference/Description Section 4.1.4

Property Name day

Reference/Description Section 4.3.3

Property Name delegatedFrom

Reference/Description Section 4.4.5

Property Name delegatedTo

Reference/Description Section 4.4.5

Property Name description

Property Context Event, Task, Participant

Reference/Description Section 4.2.2, Section 4.4.5

Property Name descriptionContentType

Property Context Event, Task, Participant

Reference/Description Section 4.2.2, Section 4.4.5

Property Name display

Reference/Description Section 1.4.11

Property Name due

Reference/Description Section 5.2.1

Property Name duration

Reference/Description Section 5.1.2

Property Name email

Reference/Description Section 4.4.5

Property Name entries

Reference/Description Section 5.3.1

Property Name estimatedDuration

Reference/Description Section 5.2.3

Property Name expectReply
Reference/Description Section 4.4.5

Property Name features
Reference/Description Section 4.2.7

Property Name firstDayOfWeek
Reference/Description Section 4.3.3

Property Name freeBusyStatus
Reference/Description Section 4.4.2

Property Name frequency
Reference/Description Section 4.3.3

Property Name href
Reference/Description Section 1.4.11

Property Name interval
Reference/Description Section 4.3.3

Property Name keywords
Reference/Description Section 4.2.10

Property Name kind
Reference/Description Section 4.4.5

Property Name links
Reference/Description Section 4.2.8, Section 4.2.5, Section 4.4.5

Property Name locale
Reference/Description Section 4.2.9

Property Name locationTypes
Reference/Description Section 4.2.5

Property Name locations
Reference/Description Section 4.2.5

Property Name memberOf
Reference/Description Section 4.4.5

Property Name method
Reference/Description Section 4.1.7

Property Name name
Reference/Description Section 4.2.5, Section 4.2.7, Section 4.4.5

Property Name nthOfPeriod
Reference/Description Section 4.3.3

Property Name offset
Reference/Description Section 4.5.1

Property Name participants
Reference/Description Section 4.4.5

Property Name participationStatus
Reference/Description Section 4.4.5

Property Name percentComplete
Reference/Description Section 5.2.4, Section 4.4.5

Property Name priority
Reference/Description Section 4.4.1

Property Name privacy
Reference/Description Section 4.4.3

Property Name prodId
Reference/Description Section 4.1.3

Property Name progress
Reference/Description Section 5.2.5, Section 4.4.5

Property Name recurrenceId
Reference/Description Section 4.3.1

Property Name recurrenceIdTimeZone
Reference/Description Section 4.3.2

Property Name recurrenceOverrides
Property Content Event, Task
Reference/Description Section 4.3.4

Property Name rel
Reference/Description Section 1.4.11

Property Name relatedTo
Reference/Description Section 4.1.2, Section 4.5.1

Property Name relation
Reference/Description Section 1.4.10

Property Name relativeTo
Property Context OffsetTrigger

Reference/Description Section 4.5.1

Property Name roles
Reference/Description Section 4.4.5

Property Name rscale
Reference/Description Section 4.3.3

Property Name sentBy
Property Context Participant
Reference/Description Section 4.4.5

Property Name sequence
Reference/Description Section 4.1.6

Property Name showWithoutTime
Reference/Description Section 4.2.4

Property Name size
Reference/Description Section 1.4.11

Property Name skip
Reference/Description Section 4.3.3

Property Name source
Reference/Description Section 5.3.2

Property Name start
Reference/Description Section 5.1.1, Section 5.2.2

Property Name status
Reference/Description Section 5.1.4

Property Name timeZone
Property Context Event, Task
Reference/Description Section 4.6.1

Property Name title
Reference/Description Section 4.2.1

Property Name trigger
Reference/Description Section 4.5.1

Property Name uid
Reference/Description Section 4.1.1

Property Name until
Reference/Description Section 4.3.3

Property Name updated
Reference/Description Section 4.1.5

Property Name uri
Reference/Description Section 4.2.7

Property Name virtualLocations
Reference/Description Section 4.2.7

Property Name when
Reference/Description Section 4.5.1

8.2.3.5. Added Properties

IANA will add the following entries in the JSCalendar Properties Registry. Referenced section numbers refer to this document, unless otherwise noted.

Property Name calendarAddress
Property Type String
Property Context Participant
Reference/Description Section 4.4.5
Intended Usage common
Is Per-User No

Property Name description
Property Type Not applicable
Property Context Location, VirtualLocation
Reference/Description Appendix A.3
Intended Usage reserved
Is Per-User No

Property Name endTimeZone
Property Type String
Property Context Event
Reference/Description Section 5.1.3
Intended Usage common
Is Per-User No

Property Name mainLocationId
Property Type String
Property Context Event, Task
Reference/Description Section 4.2.6
Intended Usage common
Is Per-User No

Property Name organizerCalendarAddress
Property Type String

Property Context Event, Task
Reference/Description Section 4.4.4
Intended Usage common
Is Per-User No

Property Name recurrenceRule
Property Type RecurrenceRule
Property Context Event, Task
Reference/Description Section 4.3.3
Intended Usage common
Is Per-User No

Property Name relativeTo
Property Context Location
Reference/Description Appendix A.2
Intended Usage obsolete
Is Per-User No

Property Name sentBy
Property Type Not applicable
Property Context Event, Task
Reference/Description Appendix A.3
Intended Usage reserved
Is Per-User No

Property Name timeZone
Property Context Location
Reference/Description Appendix A.2
Intended Usage obsolete
Is Per-User No

8.3. "JSCalendar Types" Registry

IANA has created the "JSCalendar Types" registry to avoid name collisions and provide a complete reference for all data types used for JSCalendar property values. IANA will set the Reference of the registry to this document, rather than obsoleted [RFC8984].

The registration process is the same as for the "JSCalendar Properties" registry, as defined in Section 8.2.1.

8.3.1. "JSCalendar Types" Registry Template

Note for IANA: This section leaves the original definition of [RFC8984] unchanged.

Type Name: This is the name of the type.

Reference or Description: This is a brief description or RFC number and section reference where the Type is specified (may be omitted for "reserved" type names).

Intended Use: This may be "common", "reserved", or "obsolete".

Change Controller: This is who may request a change to this entry's definition (IETF for RFCs from the IETF stream)

8.3.2. Changes to the JSCalendar Types Registry contents

8.3.2.1. Obsoleted Types

IANA will change the Intended Status from "common" to "obsolete" for the following entries in the JSCalendar Types Registry. For each entry, the Reference/Description column will refer to Appendix A.6 of this document.

- * TimeZone

- * TimeZoneRule

8.3.2.2. Updated Types

IANA will update the following entries in the JSCalendar Types Registry. For each entry, the Type Name and its changed fields are listed. Unchanged fields are omitted. Referenced section numbers refer to this document, unless otherwise noted.

Type Name Alert
Reference/Description Section 4.5.1

Type Name Boolean
Reference/Description Section 1.3.1

Type Name Duration
Reference/Description Section 1.4.6

Type Name Id
Reference/Description Section 1.4.1

Type Name Int
Reference/Description Section 1.4.2

Type Name LocalDateTime
Reference/Description Section 1.4.5

Type Name Link

Reference/Description Section 1.4.11

Type Name Location
Reference/Description Section 4.2.5

Type Name NDay
Reference/Description Section 4.3.3

Type Name Number
Reference/Description Section 1.3.1

Type Name Participant
Reference/Description Section 4.4.5

Type Name PatchObject
Reference/Description Section 1.4.9

Type Name RecurrenceRule
Reference/Description Section 4.3.3

Type Name Relation
Reference/Description Section 1.4.10

Type Name SignedDuration
Reference/Description Section 1.4.7

Type Name String
Reference/Description Section 1.3.1

Type Name TimeZoneId
Reference/Description Section 1.4.8

Type Name UnsignedInt
Reference/Description Section 1.4.3

Type Name UTCDateTime
Reference/Description Section 1.4.4

8.3.2.3. Added Types

IANA will add the following entries in the JSCalendar Types Registry. Referenced section numbers refer to this document, unless otherwise noted.

Type Name Event
Reference/Description Section 2.1
Intended Usage common
Change Controller IETF

Type Name Group
Reference/Description Section 2.3
Intended Usage common
Change Controller IETF

Type Name Task
Reference/Description Section 2.2
Intended Usage common
Change Controller IETF

Type Name OffsetTrigger
Reference/Description Section 4.5.1
Intended Usage common
Change Controller IETF

Type Name AbsoluteTrigger
Reference/Description Section 4.5.1
Intended Usage common
Change Controller IETF

Type Name UnknownTrigger
Reference/Description Section 4.5.1
Intended Usage common
Change Controller IETF

8.4. "JSCalendar Enum Values" Registry

IANA has created the "JSCalendar Enum Values" registry to allow interoperable extension of semantics for properties with enumerable values. Each such property has a subregistry of allowed values. IANA will set the Reference of the registry to this document, rather than obsoleted [RFC8984].

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 Section 8.2.1.

8.4.1. "JSCalendar Enum Values" Registry Property Template

Note for IANA: This section leaves the original definition of [RFC8984] unchanged.

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

Property Name: These are the name(s) of the property or properties where these values may be used. This MUST be registered in the "JSCalendar Properties" registry.

Context: This is 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 RFCs from the IETF stream).

Initial Contents: This is the initial list of defined values for this enum, using the template defined in Section 8.4.2. A subregistry will be created with these values for this property name/context tuple.

8.4.2. "JSCalendar Enum Values" Registry Value Template

Note for IANA: This section adds the "Intended Use" column to the template.

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

Enum Value: This is the verbatim value of the enum.

Reference or Description: This is a brief description or RFC number and section reference for the semantics of this value.

Intended Use: This may be "common", "reserved", or "obsolete".

8.4.3. Changes to the JSCalendar Enum Values Registry contents

8.4.3.1. Updated References

This document adds the "Intended Use" column to the registry template (Section 8.4.2). IANA will set the value for each existing entry to "common", except for the special case of the "attendee" roles enum described in Section 8.4.3.3.4.

IANA will set the Reference of the registry and its sub-registries to a reference to this document. It will set the Reference column of the property names to a reference to this document.

IANA will set the Reference columns of the Enum Values sub-registries as follows:

Registry Name	Reference
JSCalendar Enum Values for action (Context: Alert)	Section 4.5.1
JSCalendar Enum Values for display (Context: Link)	Section 1.4.11
JSCalendar Enum Values for features (Context: VirtualLocation)	Section 4.2.7
JSCalendar Enum Values for freeBusyStatus (Context: Event, Task)	Section 4.4.2
JSCalendar Enum Values for kind (Context: Participant)	Section 4.4.5
JSCalendar Enum Values for participationStatus (Context: Participant)	Section 4.4.5
JSCalendar Enum Values for privacy (Context: Event, Task)	Section 4.4.3
JSCalendar Enum Values for roles (Context: Participant)	Section 4.4.5
JSCalendar Enum Values for status (Context: Event)	Section 5.1.4

Table 1: Updates references of the "JSCalendar Enum Values" registry"

8.4.3.2. Removed Registries

IANA will remove the sub-registry "JSCalendar Enum Values for scheduleAgent (Context: Participant)".

8.4.3.3. Updated Registries

8.4.3.3.1. JSCalendar Enum Values for progress (Context: Task, Participant)

IANA will rename the sub-registry to "JSCalendar Enum Values for progress (Context: Task)". It will set the Reference column of the enum values to a reference to Section 5.2.5 of this document.

8.4.3.3.2. JSCalendar Enum Values for relativeTo (Context: OffsetTrigger, Location)

IANA will rename the sub-registry to "JSCalendar Enum Values for relativeTo (Context: OffsetTrigger)". It will set the Reference column of the enum values to a reference to Section 4.5.1 of this document.

8.4.3.3.3. JSCalendar Enum Values for relation (Context: Relation)

IANA will set the Reference column of the enum values to a reference to Section 1.4.10 of this document.

It will add the following Enum Value entry:

```
Enum Value    snooze
Reference/Description  Section 4.5.1
Change Controller    IETF
```

8.4.3.3.4. JSCalendar Enum Values for roles (Context: Context: Participant)

IANA will set "Intended Use" column of the "attendee" value to "obsolete".

8.4.3.4. Added Registries

IANA will add the following sub-registries to the "JSCalendar Enum Values" registry.

8.4.3.4.1. JSCalendar Enum Values for progress (Context: Participant)

```
Property Name:    progress
Context:          Participant
Change Controller: IETF
Initial Contents:
```

Enum Value	Reference or Description
in-process	Section 4.4.5
completed	Section 4.4.5
failed	Section 4.4.5

Table 2: JSCalendar Enum Values for progress (Context: Participant)

9. References

9.1. Normative References

- [CLDR] "Unicode Common Locale Data Repository",
<<http://cldr.unicode.org/>>.
- [COLORS] elik, T., Lilley, C., and L. Baron, "CSS Color Module Level 3", W3C Recommendation, June 2018,
<<https://www.w3.org/TR/css-color-3/>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997,
<<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC2397] Masinter, L., "The "data" URL scheme", RFC 2397, DOI 10.17487/RFC2397, August 1998,
<<https://www.rfc-editor.org/info/rfc2397>>.
- [RFC2445] Dawson, F. and D. Stenerson, "Internet Calendaring and Scheduling Core Object Specification (iCalendar)", RFC 2445, DOI 10.17487/RFC2445, November 1998,
<<https://www.rfc-editor.org/info/rfc2445>>.
- [RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002,
<<https://www.rfc-editor.org/info/rfc3339>>.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005,
<<https://www.rfc-editor.org/info/rfc3986>>.
- [RFC4589] Schulzrinne, H. and H. Tschofenig, "Location Types Registry", RFC 4589, DOI 10.17487/RFC4589, July 2006,
<<https://www.rfc-editor.org/info/rfc4589>>.
- [RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006,
<<https://www.rfc-editor.org/info/rfc4648>>.
- [RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008,
<<https://www.rfc-editor.org/info/rfc5234>>.

- [RFC5322] Resnick, P., Ed., "Internet Message Format", RFC 5322, DOI 10.17487/RFC5322, October 2008, <<https://www.rfc-editor.org/info/rfc5322>>.
- [RFC5545] Desruisseaux, B., Ed., "Internet Calendaring and Scheduling Core Object Specification (iCalendar)", RFC 5545, DOI 10.17487/RFC5545, September 2009, <<https://www.rfc-editor.org/info/rfc5545>>.
- [RFC5546] Daboo, C., Ed., "iCalendar Transport-Independent Interoperability Protocol (iTIP)", RFC 5546, DOI 10.17487/RFC5546, December 2009, <<https://www.rfc-editor.org/info/rfc5546>>.
- [RFC5646] Phillips, A., Ed. and M. Davis, Ed., "Tags for Identifying Languages", BCP 47, RFC 5646, DOI 10.17487/RFC5646, September 2009, <<https://www.rfc-editor.org/info/rfc5646>>.
- [RFC5870] Mayrhofer, A. and C. Spanring, "A Uniform Resource Identifier for Geographic Locations ('geo' URI)", RFC 5870, DOI 10.17487/RFC5870, June 2010, <<https://www.rfc-editor.org/info/rfc5870>>.
- [RFC6638] Daboo, C. and B. Desruisseaux, "Scheduling Extensions to CalDAV", RFC 6638, DOI 10.17487/RFC6638, June 2012, <<https://www.rfc-editor.org/info/rfc6638>>.
- [RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <<https://www.rfc-editor.org/info/rfc6838>>.
- [RFC6901] Bryan, P., Ed., Zyp, K., and M. Nottingham, Ed., "JavaScript Object Notation (JSON) Pointer", RFC 6901, DOI 10.17487/RFC6901, April 2013, <<https://www.rfc-editor.org/info/rfc6901>>.
- [RFC7493] Bray, T., Ed., "The I-JSON Message Format", RFC 7493, DOI 10.17487/RFC7493, March 2015, <<https://www.rfc-editor.org/info/rfc7493>>.
- [RFC7529] Daboo, C. and G. Yakushev, "Non-Gregorian Recurrence Rules in the Internet Calendaring and Scheduling Core Object Specification (iCalendar)", RFC 7529, DOI 10.17487/RFC7529, May 2015, <<https://www.rfc-editor.org/info/rfc7529>>.

- [RFC7809] Daboo, C., "Calendaring Extensions to WebDAV (CalDAV): Time Zones by Reference", RFC 7809, DOI 10.17487/RFC7809, March 2016, <<https://www.rfc-editor.org/info/rfc7809>>.
- [RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <<https://www.rfc-editor.org/info/rfc8126>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <<https://www.rfc-editor.org/info/rfc8259>>.
- [RFC8288] Nottingham, M., "Web Linking", RFC 8288, DOI 10.17487/RFC8288, October 2017, <<https://www.rfc-editor.org/info/rfc8288>>.
- [RFC8607] Daboo, C., Quillaud, A., and K. Murchison, Ed., "Calendaring Extensions to WebDAV (CalDAV): Managed Attachments", RFC 8607, DOI 10.17487/RFC8607, June 2019, <<https://www.rfc-editor.org/info/rfc8607>>.
- [RFC8984] Jenkins, N. and R. Stepanek, "JSCalendar: A JSON Representation of Calendar Data", RFC 8984, DOI 10.17487/RFC8984, July 2021, <<https://www.rfc-editor.org/info/rfc8984>>.
- [RFC9073] Douglass, M., "Event Publishing Extensions to iCalendar", RFC 9073, DOI 10.17487/RFC9073, August 2021, <<https://www.rfc-editor.org/info/rfc9073>>.
- [RFC9074] Daboo, C. and K. Murchison, Ed., "\"VALARM\" Extensions for iCalendar", RFC 9074, DOI 10.17487/RFC9074, August 2021, <<https://www.rfc-editor.org/info/rfc9074>>.
- [RFC9253] Douglass, M., "Support for iCalendar Relationships", RFC 9253, DOI 10.17487/RFC9253, August 2022, <<https://www.rfc-editor.org/info/rfc9253>>.
- [RFC9562] Davis, K., Peabody, B., and P. Leach, "Universally Unique IDentifiers (UUIDs)", RFC 9562, DOI 10.17487/RFC9562, May 2024, <<https://www.rfc-editor.org/info/rfc9562>>.

[TZDB] IANA, "Time Zone Database",
<<https://www.iana.org/time-zones>>.

9.2. Informative References

[ical-tasks]

Douglass, M. and A. Apthorp, "Task Extensions to iCalendar", Work in Progress, Internet-Draft, draft-ietf-calext-ical-tasks, 7 July 2025, <<https://datatracker.ietf.org/doc/draft-ietf-calext-ical-tasks>>.

[ISO.9070.1991]

ISO/IEC, "Information technology -- SGML support facilities -- Registration procedures for public text owner identifiers", Edition 2, ISO/IEC 9070:1991, April 1991, <<https://www.iso.org/standard/16645.html>>.

[jmap-calendars]

Jenkins, N.M. and M. Douglass, "JMAP for Calendars", Work in Progress, Internet-Draft, draft-ietf-jmap-calendars, 8 October 2025, <<https://datatracker.ietf.org/doc/draft-ietf-jmap-calendars/>>.

[LOCATIONTYPES]

IANA, "Location Types Registry",
<<https://www.iana.org/assignments/location-type-registry>>.

[MEDIATYPES]

IANA, "Media Types",
<<https://www.iana.org/assignments/media-types>>.

[RFC6376] Crocker, D., Ed., Hansen, T., Ed., and M. Kucherawy, Ed., "DomainKeys Identified Mail (DKIM) Signatures", STD 76, RFC 6376, DOI 10.17487/RFC6376, September 2011, <<https://www.rfc-editor.org/info/rfc6376>>.

[RFC7265] Kewisch, P., Daboo, C., and M. Douglass, "jCal: The JSON Format for iCalendar", RFC 7265, DOI 10.17487/RFC7265, May 2014, <<https://www.rfc-editor.org/info/rfc7265>>.

[RFC7986] Daboo, C., "New Properties for iCalendar", RFC 7986, DOI 10.17487/RFC7986, October 2016, <<https://www.rfc-editor.org/info/rfc7986>>.

Appendix A. Differences from RFC 8984

This section documents all significant differences from RFC 8984. Insignificant differences, such as formatting, grammar or typos are not documented.

A.1. Applied Errata

All verified errata of RFC 8984 was applied to this document:

- * Errata 6872 (<https://www.rfc-editor.org/errata/eid6872>)
- * Errata 6873 (<https://www.rfc-editor.org/errata/eid6873>)
- * Errata 8028 (<https://www.rfc-editor.org/errata/eid8028>)

A.2. Obsolete Properties

The following properties became obsolete:

`*excludedRecurrenceRules*`:

This is incompatible with iCalendar, which deprecated the `EXRULE` property [RFC2445] (Section 4.8.5.2) in [RFC5545].

See Section 4.3.4 of [RFC8984].

`*recurrenceRules*`:

This is incompatible with the following definitions of the iCalendar `RRULE` property [RFC5545] (Section 3.8.5.3):

- * The `RRULE` property SHOULD NOT be specified more than once.
- * The recurrence set generated with multiple `RRULE` properties is undefined.

The newly defined single-valued `recurrenceRule` property (Section 4.3.3) replaces it.

See Section 4.3.3 of [RFC8984].

`*timeZones*`:

This property was obsoleted for the following reasons:

- * Custom time zones in calendaring data increase implementation complexity, introduce inefficiencies and are error-prone, as described for CalDAV and iCalendar in [RFC7809].

- * In practice, the only non-IANA time zone identifiers are Microsoft time zone names. These can be converted to IANA time zones, e.g. using the International Components for Unicode (<https://icu.unicode.org/>) software library.
- * The property contradicts the stated goal of JSCalendar to avoid ambiguities and pitfalls of iCalendar, defined in Section 1.1 of [RFC8984].

See Section 4.7.2 of [RFC8984].

***Link.cid*:**

This property was obsoleted for the following reasons:

- * It only is relevant for rich-text descriptions, but the related iCalendar STYLED-DESCRIPTION property [RFC9073] (Section 6.5) does not support referring to attached media.
- * It only is applicable for Link objects having a "data:" URI as href property value. But even the definition of the Link object recommends not to use "data:" URIs as values in the href property.
- * A Link with "data:" URI converts to an iCalendar BINARY value [RFC5545] (Section 3.3.1). But these are marginally supported by CalDAV clients, which tend to prefer using WebDAV managed attachments [RFC8607].

See Section 1.4.11 of [RFC8984].

***Location.{relativeTo|timeZone}*:**

These got replaced with the newly defined Event.endTimeZone property (Section 5.1.3):

- * They were meant to be equivalent to the iCalendar DTEND property [RFC5545] (Section 3.8.2.2). But the semantics were undefined if multiple Location objects had "relativeTo=end" and the "timeZone" property set.
- * Their semantics were undefined for Task objects.
- * They were incompatible with the iCalendar VLOCATION component [RFC9073] (Section 7.2), which neither defines how to set a time zone identifier or how the VLOCATION relates to start and end.

See Section 4.2.5 of [RFC8984].

***Participant.locationId*:**

This is incompatible with the iCalendar PARTICIPANT component [RFC9073] (Section 7.1):

- * The PARTICIPANT component may contain multiple VLOCATION components and LOCATION properties.
- * The PARTICIPANT component does not support referring to other VLOCATION components or LOCATION properties

The description of the Event object type in Section 2.1 reflects this:

- * It stated: "Multiple participants may partake in the event at multiple locations."
- * It now states: "Multiple participants may partake in the event."

See Section 4.4.6 of [RFC8984].

***Participant.language*:**

No equivalent element exists in iCalendar and its current single-valued definition is likely to conflict with any related future extension of the PARTICIPANT component [RFC9073] (Section 7.1).

See Section 4.4.6 of [RFC8984].

***{Participant|Task}.progressUpdated*:**

This became obsolete for the following reasons:

- * It is incompatible with the newly introduced iCalendar VSTATUS component [ical-tasks] (Section 12.1), which may occur multiple times in the PARTICIPANT and VTOD components.
- * The VSTATUS component introduces new semantics to a task's progress and status, which better be defined by a future JSCalendar extension for tasks.

See Section 4.4.6 of [RFC8984] and Section 5.2.6 of [RFC8984].

***{TimeZone|TimeZoneRule}.**:**

All properties defined solely for the TimeZone and TimeZoneRule object types became obsolete, because the "timeZones" property became obsolete.

See Section 4.7.2 of [RFC8984].

A.3. Reserved Properties

A.3.1. JMAP for Calendars

The following common use properties became reserved for JMAP for Calendars [jmap-calendars]:

***useDefaultAlerts*:**

No equivalent element exists in iCalendar and no consensus for default alarms in CalDAV and iCalendar was found at IETF as part of [RFC9074].

See Section 4.4.1 of [RFC8984].

***Participant.scheduleSequence*:**

This requires further work by IETF to update iTIP [RFC5546] with the PARTICIPANT component and JSCalendar. This property may later be redefined for common use.

See Section 4.4.6 of [RFC8984].

***Participant.scheduleUpdated*:**

This requires further work by IETF to update iTIP [RFC5546] with the PARTICIPANT component and JSCalendar. This property may later be redefined for common use.

See Section 4.4.6 of [RFC8984].

A.3.2. Scheduling Extensions for CalDAV and iTIP

The following common use properties became reserved for future JSCalendar extensions of Scheduling Extensions for CalDAV [RFC6638] and iTIP [RFC5546]:

***replyTo*:**

This requires further work by IETF. For compatibility with iCalendar, the newly introduced organizerCalendarAddress property (Section 4.4.4) replaces it.

See Section 4.4.4 of [RFC8984].

***requestStatus*:**

This mainly is applicable in scheduling over CalDAV.

See Section 4.4.7 of [RFC8984].

***{Event|Task}.sentBy*:**

No equivalent element exists in iCalendar and this property mainly is relevant in context of a combined calendaring and mail service. Note that the `Participant.sentBy` property does not become reserved: it is equivalent to the iCalendar SENT-BY [RFC5545] (Section 3.2.18) parameter.

See Section 4.4.5 of [RFC8984].

***Participant.invitedBy*:**

This requires further work by IETF. Specifically, the iTIP definitions of the REPLY method [RFC5546] (Section 3.2.3) and how to forward invitation requests (Section 4.2.8 of [RFC5546]) must be updated.

See Section 4.4.6 of [RFC8984].

***Participant.participationComment*:**

This requires further work by IETF to update iTIP [RFC5546] with the PARTICIPANT component or JSCalendar. This property may later be redefined for common use.

See Section 4.4.6 of [RFC8984].

***Participant.scheduleAgent*:**

This only is applicable in scheduling over CalDAV.

See Section 4.4.6 of [RFC8984].

***Participant.scheduleForceSend*:**

This only is applicable in scheduling over CalDAV.

See Section 4.4.6 of [RFC8984].

***Participant.scheduleStatus*:**

This only is applicable in scheduling over CalDAV.

See Section 4.4.6 of [RFC8984].

***Participant.sendTo*:**

This requires further work by IETF. For compatibility with iCalendar, the newly introduced `Participant.calendarAddress` property (Section 4.4.5) replaces it.

See Section 4.4.6 of [RFC8984].

A.3.3. Localization Extensions for iCalendar and JSCalendar

The following properties became reserved for future JSCalendar and iCalendar extensions for multi-lingual calendar data:

***localizations*:**

This requires further work by IETF. The "localizations" property primarily was introduced in JSCalendar for event publishing, but neither the Event Publishing extensions for iCalendar [RFC9073] nor other iCalendar extensions define how to localize iCalendar data. This property may later be redefined for common use.

See Section 4.6.1 of [RFC8984].

A.3.4. Reserved for Future RFCs

The following properties became reserved for some future JSCalendar extension RFCs:

***Location.description, VirtualLocation.description*:**

This requires further work by IETF. The VirtualLocation "description" property does not convert to any parameter of the CONFERENCE property. It would require to define a new parameter or to introduce a new component to augment the CONFERENCE property with a DESCRIPTION or STYLED-DESCRIPTION property. For consistency, the Location "description" property became reserved, too.

See Section 4.2.5 of [RFC8984] and Section 4.2.6 of [RFC8984].

A.3.5. Reserved for Internal Use

The following properties became reserved for internal use by JSCalendar:

***excluded*:**

This property name already was defined for internal use only in the recurrenceOverrides property, but registering it for common use had made it appear as being a regular property. Making this a reserved property is to help clarify its purpose. Section 4.3.4 has been updated accordingly.

See Section 4.3.4 of [RFC8984].

A.4. Updated Properties

The following property definitions were updated:

***@type*:**

The original definition required this property be set on every object type. The type and property notation in Section 1.3 now matches the one in JSContact, and the @type property is optional in the majority of cases.

***Alert.relatedTo*:**

The original definition instructed to set the "parent" relation on a snooze alert. It now instructs to set the newly defined "snooze" relation, for compatibility with the VALARM "SNOOZE" relationship type (Section 7 of [RFC9074]).

The original definition did not define that the map keys represent keys of Alert objects in the "alerts" property.

See Section 4.5.1.

***Alert.trigger*:**

JSCalendar now supports default types for properties of type A|B. The default type of this property now is OffsetTrigger.

See Section 4.5.1.

***Link.display*:**

The original definition only allowed one purpose to be set, which is incompatible with the multi-valued iCalendar DISPLAY parameter [RFC7986] (Section 6.1). It now supports setting multiple purposes.

Setting this property does not anymore require the "rel" property of the Link object be set to "icon".

See Section 1.4.11.

***Link.rel*:**

The original definition restricted the value of this property to registered link types. It now also allows extension relation types for compatibility with the LINKREL parameter [RFC9253] (Section 6.1).

See Section 1.4.11.

***Participant.delegatedFrom*, *Participant.delegatedTo*,
Participant.memberOf,**

The original definitions required their values to identify Participant objects. This was incompatible with iCalendar when the DELEGATED-TO, DELEGATED-FROM or MEMBER parameters contained a calendar address but no ATTENDEE property with that same calendar address. The new definition now requires the property values to be calendar addresses.

Also, they now require the calendarAddress property to be set.

See Section 4.4.5.

Participant.expectReply, *Participant.invitedBy*,
Participant.kind, *Participant.participationStatus*,
Participant.progress, *Participant.sentBy*:

The original definitions were incompatible with iCalendar: they did not require the (now reserved) sendTo property to be set, but their equivalent iCalendar parameters require an ATTENDEE property. They now require the calendarAddress property to be set.

See Section 4.4.5.

Participant.progress:

The original definition was incompatible with iCalendar. It allowed the progress value to be "cancelled", but this is not a supported PARTSTAT parameter value of an ATTENDEE property in a VTODO component. The new definition only supports the progress property values "in-process", "completed" and "failed".

See Section 4.4.5.

Participant.roles:

The original definition was incompatible with iCalendar:

- * It did not cover the "REQ-PARTICIPANT" value of the ROLE parameter [RFC5545] (Section 3.2.16). It now defines the new "required" role.
- * The "roles" property is multi-valued but the ROLE parameter is single-valued and only allowed to be set once on an ATTENDEE. For compatibility, a precedence of Participant roles got defined for converting roles to iCalendar.
- * It was mandatory for a Participant, but the (now reserved) sendTo property was not. This was incompatible with iCalendar, which required an ATTENDEE property to set the ROLE parameter. It now is optional, but requires the calendarAddress property to be set.

- * It defined a "contact" role, but this role later got redefined as an enumerated value of the PARTICIPANT-TYPE property [RFC9073] (Section 6.2). It now removed this role and leaves defining participant types to a future JSCalendar extension.
- * The "attendee" role got removed. There is no equivalent ROLE value in iCalendar and any Participant with a calendarAddress by definition is an ATTENDEE in iCalendar.
- * The "owner" role got redefined. It now makes clear that the semantics of the role depend on the calendaring exchange protocol.

See Section 4.4.5.

***Relation.relation*:**

The original definition specified the empty relation to represent an unspecified relationship. It now defines the empty relation to default to "parent", unless overridden, for compatibility with the RELATED-TO property [RFC5545] (Section 3.8.4.5).

See Section 1.4.10.

***Task.{due|start}*:**

The original definition defined these as optional, but this is incompatible with iCalendar: if the Task object's timeZone property is set, then it requires the DUE property [RFC5545] (Section 3.8.2.3) or DTSTART property [RFC5545] (Section 3.8.2.4) be set in the VTOD component. It now requires at least one of the "due" or "start" properties be set, if the "timeZone" property is set.

The original definition did not require the "start" property be set if the "recurrenceRule" property is set. But this is incompatible with the RRULE property [RFC5545] (Section 3.8.5.3), which requires the DTSTART property be set in the VTOD component. The "start" property now is required if the "recurrenceRule" or "recurrenceId" properties are set.

The "showWithoutTime" property having value "true" now requires at least one of the "due" or "start" properties be set.

See Section 5.2.

A.5. New Properties

The following new properties were defined:

***mainLocationId*:**

This got introduced for better interoperability with iCalendar, where the VEVENT and VTODO components allow at most one LOCATION property to be present. While VLOCATION components [RFC9073] (Section 7.2) may occur multiple times, implementations need to know which Location to choose for the LOCATION property.

See Section 4.2.6.

***organizerCalendarAddress*:**

This replaces the reserved replyTo property.

See Section 4.4.4.

***recurrenceRule*:**

This replaces the obsoleted recurrenceRules property.

See Section 4.3.3.

***Event.endTimeZone*:**

This replaces the obsoleted Location.relativeTo and Location.timeZone properties.

See Section 5.1.3.

***Location.descriptionContentType*:**

This is for compatibility with the iCalendar STYLED-DESCRIPTION property [RFC9073] (Section 6.5) of the VLOCATION component.

See Section 4.2.5.

***Participant.calendarAddress*:**

This replaces the reserved Participant.sendTo property.

See Section 4.4.5.

***Participant.descriptionContentType*:**

This is for compatibility with the iCalendar STYLED-DESCRIPTION property [RFC9073] (Section 6.5) of the PARTICIPANT component.

See Section 4.2.5.

***VirtualLocation.descriptionContentType*:**

This is for consistency with the definition of the Location object type. The newly defined VCONFERENCE component will be allowed to contain the STYLED-DESCRIPTION property [RFC9073] (Section 6.5).

See Section 4.2.7.

A.6. Obsolete Types

The following type definitions became obsolete:

***TimeZone, TimeZoneRule*:**

The "timeZones" property became obsolete.

See Section 4.7.2 of [RFC8984].

A.7. Updated Types

The following type definitions were updated:

***Duration, LocalDateTime, UTCDateTime*:**

The original definitions supported fractional seconds. This is incompatible with the iCalendar DATE-TIME and DURATION types defined in Section 3.3.5 of [RFC5545] and Section 3.3.6 of [RFC5545].

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.

Authors' Addresses

Neil Jenkins
Fastmail
Collins St. West
P.O. Box 234
Melbourne VIC 8007
Australia
Email: neilj@fastmailteam.com
URI: <https://www.fastmail.com>

Robert Stepanek
Fastmail
Collins St. West
P.O. Box 234
Melbourne VIC 8007
Australia
Email: rsto@fastmailteam.com
URI: <https://www.fastmail.com>