Internet Engineering Task Force (IETF)
Request for Comments: 7265
Category: Standards Track
ISSN: 2070-1721

P. Kewisch
Mozilla
C. Daboo
Apple, Inc.
M. Douglass
RPI

May 2014

jCal: The JSON Format for iCalendar

Abstract

This specification defines "jCal", a JSON format for iCalendar data. The iCalendar data format is a text format for capturing and exchanging information normally stored within a calendaring and scheduling application, for example, tasks and events. JSON is a lightweight, text-based, language-independent data interchange format commonly used in Internet applications.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc7265.

Table of Contents

1. Introduction

The iCalendar data format [RFC5545] is a widely deployed interchange format for calendaring and scheduling data. While many applications and services consume and generate calendar data, iCalendar is a specialized format that requires its own parser/generator. In contrast, JSON-based formats as defined in [RFC7159] are the native format for JavaScript widgets and libraries, and it is appropriate to have a standard form of calendar data that is easier to work with than iCalendar.

The purpose of this specification is to define "jCal", a JSON format for iCalendar data. jCal is defined as a straightforward mapping into JSON from iCalendar, so that iCalendar data can be converted to JSON, and then back to iCalendar, without losing any semantic meaning in the data. Anyone creating jCal calendar data according to this specification will know that their data can be converted to a valid iCalendar representation as well.

The key design considerations are essentially the same as those for [RFC6321], that is:

Round-tripping (converting an iCalendar instance to jCal and back) will give the same semantic result as the starting point. For example, all components, properties, and property parameters are guaranteed to be preserved.

Ordering of elements and case of property and parameter names will not necessarily be preserved.

The iCalendar data semantics are to be preserved, allowing a simple consumer to easily browse the data in jCal. A full understanding of iCalendar is still required in order to modify and/or fully comprehend the calendar data.

Extensions to the underlying iCalendar specification must not lead to requiring an update to jCal.

2. Conventions Used in This Document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119]. The underlying format used for jCal 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 [RFC7159].

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

3. Converting from iCalendar to jCal

This section describes how iCalendar data is converted to jCal using a simple mapping between the iCalendar data model and JSON elements. Aside from the formal description in this section, an informative ABNF is specified in Appendix A.

In [RFC5545], an iCalendar object comprises a set of "components", "properties", "parameters", and "values". The top level of iCalendar data typically contains a stream of iCalendar objects, each of which can be considered a "component". A "component" can contain other "components" or "properties". A "property" has a "value" and a set of zero or more "parameters". Each of these entities have a representation in jCal, defined in the following sections. The representation of an iCalendar object in JSON will be named "jCal object" throughout this document.

3.1. Pre-processing

iCalendar uses a line-folding mechanism to limit lines of data to a maximum line length (typically 75 octets) to ensure the maximum likelihood of preserving data integrity as it is transported via various means (e.g., email) -- see Section 3.1 of [RFC5545].

iCalendar data uses an "escape" character sequence for text values and property parameter values. See Sections 3.1 and 3.3 of [RFC5545] as well as [RFC6868].

There is a subtle difference in the number representations between JSON and iCalendar. While in iCalendar, a number may have leading zeros, as well as a leading plus sign; this is not the case in JSON. Numbers should be represented in whatever way needed for the underlying format.

When converting from iCalendar to jCal: First, iCalendar lines MUST be unfolded. Afterwards, any iCalendar escaping MUST be unescaped. Finally, JSON escaping, as described in Section 7 of [RFC7159], MUST be applied. The reverse order applies when converting from jCal to iCalendar, which is further described in Section 4.

iCalendar uses a base64 encoding for binary data. However, it does not restrict the encoding from being applied to non-binary value types. So, the following rules are applied when processing a property with the "ENCODING" property parameter set to "BASE64":

- If the property value type is "BINARY", the base64 encoding MUST be preserved.

- If the value type is not "BINARY", the "ENCODING" property parameter MUST be removed, and the value MUST be base64 decoded.

When base64 encoding is used, it MUST conform to Section 4 of [RFC4648], which is the base64 method used in [RFC5545].

One key difference in the formatting of values used in iCalendar and jCal is that in jCal, the specification uses date/time values aligned with the extended format of [ISO.8601.2004], which is more commonly used in Internet applications that make use of the JSON format. The sections of this document describing the various date and time formats contain more information on the use of the complete representation, reduced accuracy, or truncated representation.

3.2. iCalendar Stream and Objects (RFC 5545, Section 3.4)

At the top level of the iCalendar object model is an "iCalendar stream". This stream encompasses multiple "iCalendar objects". As the typical use case is transporting a single iCalendar object, there is no defined equivalent to an "iCalendar stream" in jCal. To transport multiple jCal objects in a stream, a simple JSON array can be used.

Example:

```
["vcalendar",
  [ /* Add jCal properties in place of this comment */ ],
  [ /* Add jCal components in place of this comment */ ]
]
```

3.3. Components (RFC 5545, Section 3.6)

Each iCalendar component, delimited by "BEGIN" and "END", will be converted to a fixed-length array with three fields that have a specific structure:

1. A string with the name of the iCalendar component, but in lowercase.

2. An array of jCal properties as described in Section 3.4.

3. An array of jCal components, representing the sub-components of the component in question.

This mapping applies to the top level iCalendar objects, as well as individual sub-components in the same way. The iCalendar to jCal component mapping is valid for both current iCalendar components and any new iCalendar components added in the future. Conversion is to be done in the same way.

While the grouping of properties and sub-components does not retain the original order specified in the iCalendar data, the semantics of a component are preserved.

Example:

```
["vevent",
  [ /* Add jCal properties in place of this comment */ ],
  [ /* Add jCal components in place of this comment */ ]
]
```

3.4. Properties (RFC 5545, Sections 3.7 and 3.8)

iCalendar properties, whether they apply to the "VCALENDAR" object or to a component, are handled in a consistent way in the jCal format.

In jCal, each individual iCalendar property MUST be represented by an array with three fixed elements, followed by one or more additional elements, depending on if the property is a multi-valued property as described in Section 3.1.2 of [RFC5545].

The array consists of the following fixed elements:

1. The name of the property, as a lowercase string. The iCalendar format specifies that property names are case insensitive and recommends that they be rendered in uppercase. In jCal, they MUST be in lowercase.

2. An object containing the parameters as described in Section 3.5. If the property has no parameters, an empty object is used to represent that.

3. The type identifier string of the value, in lowercase. Due to special casing of certain properties as described in Section 3.4.1, it is important that parsers check both the type identifier and the value data type and do not rely on assumptions based on the property name.

The remaining elements of the array are used for one or more values of the property. For single-valued properties, the array has exactly four elements; for multi-valued properties, as described in Section 3.1.2 of [RFC5545], each value is another element, and there can be any number of additional elements.

In the following example, the "categories" property is multi-valued and has two values, while the summary property is single-valued:

Example:

```
["vevent",
  [
    ["summary", {}, "text", "Meeting with Fred"],
    ["categories", {}, "text", "Meetings", "Work"]
    ...
  ],
  [ /* sub-components */ ]
]
```

3.4.1. Special Cases for Properties

This section describes some properties that have special handling when converting to jCal.

3.4.1.1. GEO Property (RFC 5545, Section 3.8.1.6)

In iCalendar, the "GEO" property value is defined as a semicolon-separated list of two "FLOAT" values, the first representing latitude and the second longitude.

In jCal, the value for the "geo" property value is represented as an array of two values. The first value of the property represents the latitude; the second value represents the longitude.

When converting from jCal to iCalendar, be careful to use a semicolon as the separator between the two values as required by [RFC5545].

When converting from jCal to iCalendar, the two values MUST be converted using a semicolon as the separator character.

Example

```
["vevent",
  [
    ["geo", {}, "float", [ 37.386013, -122.082932 ] ]
    ...
  ],
  ...
]
```

3.4.1.2. REQUEST-STATUS Property (RFC 5545, Section 3.8.8.3)

In iCalendar, the "REQUEST-STATUS" property value is defined as a semicolon-separated list of two or three "TEXT" values. The first represents a code, the second a description, and the third any additional data.

In jCal, the value for the "request-status" property value is represented as an array with two or three values. The first array element corresponds to the code, the second element corresponds to the description, and the third element corresponds to the additional data. Each value is represented using a string value. If there is no additional data in the iCalendar value, the last element of the array SHOULD NOT be present.

When converting from jCal to iCalendar, the two or three values MUST be converted using a semicolon as the separator character.

iCalendar Example:

```
BEGIN:VEVENT
...
REQUEST-STATUS:2.0;Success
REQUEST-STATUS:3.7;Invalid calendar user;ATTENDEE:
 mailto:jsmith@example.com
...
END:VEVENT
```

jCal Example:

```
["vevent":
  [
    ["request-status", {}, "text", ["2.0", "Success"] ],
    ["request-status", {}, "text",
       [
        "3.7",
        "Invalid calendar user",
        "ATTENDEE:mailto:jsmith@example.org"
       ]
    ],
    ...
  ],
  ...
]
```

3.5. Parameters (RFC 5545, Section 3.2)

Property parameters are represented as a JSON object where each key-value pair represents the iCalendar parameter name and its value. The name of the parameter MUST be in lowercase; the original case of the parameter value MUST be preserved. For example, the "PARTSTAT" property parameter is represented in jCal by the "partstat" key. Any new iCalendar parameters added in the future will be converted in the same way.

Example:

```
["vevent":
  [
    ["attendee",
     {
       "partstat": "ACCEPTED",
       "rsvp": "TRUE",
       "role": "REQ-PARTICIPANT"
     },
     "cal-address",
     "mailto:jsmith@example.org"
    ],
    ["summary", {}, "text", "Meeting"],
    ...
  ],
  ...
]
```

3.5.1. VALUE Parameter

iCalendar defines a "VALUE" property parameter (Section 3.2.20 of [RFC5545]). This property parameter MUST NOT be added to the parameters object. Instead, the value type is signaled through the type identifier in the third element of the array describing the property. When converting a property from iCalendar to jCal, the value type is determined as follows:

1. If the property has a "VALUE" parameter, that parameter’s value is used as the value type.

2. If the property has no "VALUE" parameter but has a default value type, the default value type is used.

3. If the property has no "VALUE" parameter and has no default value type, "unknown" is used.

Converting from jCal into iCalendar is done as follows:

1. If the property’s value type is "unknown", no "VALUE" parameter is included.

2. If the property’s value type is the default type for that property, no "VALUE" parameter is included.

3. Otherwise, a "VALUE" parameter is included, and the value type is used as the parameter value.

See Section 5 for information on handling unknown value types.

3.5.2. Multi-value Parameters

In [RFC5545], some parameters allow using a COMMA-separated list of values. To ease processing in jCal, the value of such parameters MUST be represented in an array containing the separated values. The array elements MUST be string values. Single-value parameters can be represented using either a single string value or an array with one string element. A jCal parser MUST be able to understand both value data types. Examples of such parameters are the iCalendar "DELEGATED-FROM" and "DELEGATED-TO" parameters; more such parameters may be added in extensions.

The iCalendar specification requires encapsulation between DQUOTE characters if a parameter value contains a colon, a semicolon, or a comma. These extra DQUOTE characters do not belong to the actual parameter value, and hence are not included when the parameter is converted to jCal.

Example 1:

```
["attendee",
 {
   "delegated-to": ["mailto:jdoe@example.org",
                    "mailto:jqpublic@example.org"]
 },
 "cal-address",
 "mailto:jsmith@example.org"
]
```

Example 2:

```
["attendee",
 {
   "delegated-to": "mailto:jdoe@example.org"
 },
 "cal-address",
 "mailto:jsmith@example.org"
]
```

3.6. Values (RFC 5545, Section 3.3)

The following subsections specify how iCalendar property value data types, which are defined in the subsections of [RFC5545], Section 3.3, are represented in jCal.

3.6.1. Binary (RFC 5545, Section 3.3.1)

Description: iCalendar "BINARY" property values are represented by a property with the type identifier "binary". The value element is a JSON string, encoded with base64 encoding as specified in Section 4 of [RFC4648].

Example:

```
["attach", {}, "binary", "SGVsbG8gV29ybGQh"]
```

3.6.2. Boolean (RFC 5545, Section 3.3.2)

Description: iCalendar "BOOLEAN" property values are represented by a property with the type identifier "boolean". The value is a JSON boolean value.

Example:

```
["x-non-smoking", {}, "boolean", true]
```

3.6.3. Calendar User Address (RFC 5545, Section 3.3.3)

Description: iCalendar "CAL-ADDRESS" property values are represented by a property with the type identifier "cal-address". The value is a JSON string with the URI as described in [RFC3986].

Example:

```
["attendee", {}, "cal-address", "mailto:kewisch@example.com"]
```

3.6.4. Date (RFC 5545, Section 3.3.4)

Description: iCalendar "DATE" property values are represented by a property with the type identifier "date". The value elements are JSON strings with the same date value specified by [RFC5545], but represented using the extended format of the complete representation specified in [ISO.8601.2004], Section 4.1.2.2. Other variations, for example, representation with reduced accuracy, MUST NOT be used.

ABNF Schema:

```
; year, month, and day rules are
; defined in [ISO.8601.2004], Section 2.2.
date = year "-" month "-" day ;YYYY-MM-DD
```

Example:

```
["dtstart", {}, "date", "2011-05-17"]
```

3.6.5. Date-Time (RFC 5545, Section 3.3.5)

Description: iCalendar "DATE-TIME" property values are represented by a property with the type identifier "date-time". The value elements are JSON strings with the same date value specified by [RFC5545], but represented using the extended format of the complete representation specified in [ISO.8601.2004], Section 4.3.2. Other variations, for example, representation with reduced accuracy, MUST NOT be used. The same restrictions apply with respect to leap seconds and time zone offsets as specified in [RFC5545], Section 3.3.5.

ABNF Schema:

```
; year, month, day, hour, minute, and second rules are
; defined in [ISO.8601.2004], Section 2.2.
; The zone identifier is described in [ISO.8601.2004], Section 4.3.2.
date-complete = year "-" month "-" day ;YYYY-MM-DD
time-complete =  hour ":" minute ":" second [zone] ; HH:MM:SS
datetime = date-complete "T" time-complete
```

Examples:

```
["dtstart", {}, "date-time", "2012-10-17T12:00:00"],
["dtstamp", {}, "date-time", "2012-10-17T12:00:00Z"],
["dtend",
 { "tzid": "Europe/Berlin" },
 "date-time",
 "2011-10-17T13:00:00"
]
```

3.6.6. Duration (RFC 5545, Section 3.3.6)

Description: iCalendar "DURATION" property values are represented by a property with the type identifier "duration". The value elements are JSON strings with the same duration value specified by [RFC5545].

Example:

```
["duration", {}, "duration", "P1D"]
```

3.6.7. Float (RFC 5545, Section 3.3.7)

Description: iCalendar "FLOAT" property values are represented by a property with the type identifier "float". The value elements are JSON primitive number values.

Example:

```
["x-grade", {}, "float", 1.3]
```

3.6.8. Integer (RFC 5545, Section 3.3.8)

Description: vCard "INTEGER" property values are represented by a property with the type identifier "integer". The value elements are JSON primitive number values that MUST resolve to an integer value in the range specified in [RFC5545], Section 3.3.8. Thus, a fractional and/or exponential part are only allowed under limited circumstances.

Examples:

```
["percent-complete", {}, "integer", 42]
```

3.6.9. Period of Time (RFC 5545, Section 3.3.9)

Description: iCalendar "PERIOD" property values are represented by a jCal property with the type identifier "period". The value element is an array of JSON strings, with the first element representing the start of the period and the second element representing the end of the period. As in [RFC5545], the start of the period is always formatted as a date-time value, and the end of the period MUST be either a date-time or duration value. Any date, date-time, or duration values contained in the period value MUST be formatted in accordance to the rules for date, date-time, or duration values specified in this document.

Example:

```
["freebusy",
 { "fbtype": "FREE" },
 "period",
 ["1997-03-08T16:00:00Z", "P1D"]
]
```

3.6.10. Recurrence Rule (RFC 5545, Section 3.3.10)

Description: iCalendar "RECUR" property values are represented by a property with the type identifier "recur". The value elements are objects describing the structured data as specified by [RFC5545]. Each rule part is described by the combination of key and value. The key specifies the name of the rule part and MUST be converted to lowercase. The value of the rule part MUST be mapped by the following rules:

* The value of the "freq" and "wkst" rule parts MUST be a string as specified in [RFC5545], with case preserved.

* The value of the "until" rule part MUST be a date or date-time value formatted in accordance to the rules for date or date-time specified in this document.

* The "count" and "interval" rule parts MUST be specified as a single JSON number value.

* The following rule parts can have one or more numeric values: "bysecond", "byminute", "byhour", "bymonthday", "byyearday", "byweekno", "bymonth", and "bysetpos". If a rule part contains multiple values, an array of numbers MUST be used for that rule part. Single-valued rule parts can be represented by either using a single number value, omitting the array completely, or using an array with one number element. A jCal parser MUST be able to understand both data types.

* Similarly, the "byday" rule part can have one or more string values. If it contains multiple values, an array of strings MUST be used. As before, a single-valued rule part can be represented using either a single string value or an array with one string element, both of which a jCal parser MUST be able to understand.

Example 1:

```
["rrule",
 {},
 "recur",
 {
   "freq": "YEARLY",
   "count": 5,
   "byday": [ "-1SU", "2MO" ],
   "bymonth": 10
 }
]
```

Example 2:

```
["rrule",
 {},
 "recur",
 {
   "freq": "MONTHLY",
   "interval": 2,
   "bymonthday": [ 1, 15, -1 ],
   "until": "2013-10-01"
 }
]
```

3.6.11. Text (RFC 5545, Section 3.3.11)

Description: iCalendar "TEXT" property values are represented by a property with the type identifier "text". The value elements are JSON strings.

Example:

```
["comment", {}, "text", "hello, world"]
```

3.6.12. Time (RFC 5545, Section 3.3.12)

Description: iCalendar "TIME" property values are represented by a property with the type identifier "time". The value elements are JSON strings with the same time value specified by [RFC5545], but represented using the extended format of the complete representation specified in [ISO.8601.2004], Section 4.2.2.2. Other variations, for example, representation with reduced accuracy, MUST NOT be used. The same restrictions apply with respect to leap seconds, time fractions, and time zone offsets as specified in [RFC5545], Section 3.3.12.

ABNF Schema:

```
; hour, minute, and second rules are
; defined in [ISO.8601.2004], Section 2.2.
; The zone identifier is described in [ISO.8601.2004], Section 4.3.2.
time-complete =  hour ":" minute ":" second [zone] ; HH:MM:SS
```

Example:

```
["x-time-local", {}, "time", "12:30:00"],
["x-time-utc", {}, "time", "12:30:00Z"],
["x-time-offset", { "tzid": "Europe/Berlin" }, "time", "12:30:00"]
```

3.6.13. URI (RFC 5545, Section 3.3.13)

Description: iCalendar "URI" property values are represented by a property with the type identifier "uri". The value elements are JSON strings representing the URI.

Example:

```
["tzurl", {}, "uri", "http://example.org/tz/Europe-Berlin.ics"]
```

3.6.14. UTC Offset (RFC 5545, Section 3.3.14)

Description: iCalendar "UTC-OFFSET" property values are represented by a property with the type identifier "utc-offset". The value elements are JSON strings with the same UTC offset value specified by [RFC5545], with the exception that the hour and minute components are separated by a ":" character, for consistency with the [ISO.8601.2004] time zone offset, extended format.

Example:

```
["tzoffsetfrom", {}, "utc-offset", "-05:00"],
["tzoffsetto", {}, "utc-offset", "+12:45"]
```

3.7. Extensions

iCalendar extension properties and property parameters (those with an "X-" prefix in their name) are handled in the same way as other properties and property parameters: the property is represented by an array, and the property parameter is represented by an object. The property or parameter name uses the same name as for the iCalendar extension, but in lowercase. For example, the "X-FOO" property in iCalendar turns into the "x-foo" jCal property. See Section 5 for how to deal with default values for unrecognized extension properties or property parameters.

4. Converting from jCal into iCalendar

Converting jCal to iCalendar reverses the process described in Section 3. This section describes a few additional requirements for conversion.

When converting component, property, and property parameter names, the names SHOULD be converted to uppercase. Although iCalendar names are case insensitive, common practice is to keep them all uppercase following the actual definitions in [RFC5545].

During conversion, JSON escaping MUST be unescaped. Afterwards, iCalendar escaping, as defined by [RFC5545] and [RFC6868], MUST be applied. Finally, long lines SHOULD be folded as described in [RFC5545], Section 3.1.

Non-binary value types MUST NOT be base64 encoded.

When converting to iCalendar, the VALUE parameter MUST be added to properties whose default value type is unknown, but do not have a jCal type identifier "unknown". The VALUE parameter MAY be omitted for properties using the default value type. The VALUE parameter MUST be omitted for properties that have the jCal type identifier "unknown".

5. Handling Unrecognized Properties or Parameters

In iCalendar, properties can have one or more value types as specified by their definition, with one of those values being defined as the default. When a property uses its default value type, the "VALUE" property parameter does not need to be specified on the property. For example, the default value type for "DTSTART" is "DATE-TIME", so "VALUE=DATE-TIME" need not be set as a property parameter. However, "DTSTART" also allows a "DATE" value to be specified, and if that is used, "VALUE=DATE" has to be set as a property parameter.

When new properties are defined or "X-" properties used, an iCalendar to jCal converter might not recognize them, and not know what the appropriate default value types are, yet they need to be able to preserve the values. A similar issue arises for unrecognized property parameters.

In jCal, a new "unknown" property value type is introduced. Its purpose is to allow preserving unknown property values when round-tripping between jCal and iCalendar. To avoid collisions, this specification reserves the UNKNOWN property value type in iCalendar. It MUST NOT be used in any iCalendar as specified by [RFC5545], nor any extensions to it. Thus, the type is registered to the iCalendar Value Data Types registry in Section 7.1.

5.1. Converting iCalendar into jCal

Any property that does not include a "VALUE" property parameter and whose default value type is not known, MUST be converted to a primitive JSON string. The content of that string is the unprocessed value text. Also, value type MUST be set to "unknown".

To correctly implement this format, it is critical that the type "unknown" be used if the default type is not known. If this requirement is ignored and, for example, "text" is used, additional escaping may occur, which breaks round-tripping values.

Any unrecognized property parameter MUST be converted to a string value, with its content set to the property parameter value text, and treated as if it were a "TEXT" value.

5.2. Converting jCal into iCalendar

In jCal, the value type is always explicitly specified. It is converted to iCalendar using the iCalendar VALUE parameter, except in the following two cases:

- If the value type specified in jCal matches the default value type in iCalendar, the VALUE parameter MAY be omitted.
- If the value type specified in jCal is set to "unknown", the VALUE parameter MUST NOT be specified. The value MUST be taken over in iCalendar without processing.

5.3. Examples

The following is an example of an unrecognized iCalendar property (that uses a "DATE-TIME" value as its default), and the equivalent jCal representation of that property.

iCalendar:

```
X-COMPLAINT-DEADLINE:20110512T120000Z
```

jCal:

```
["x-complaint-deadline", {}, "unknown", "20110512T120000Z"]
```

The following is an example of how to cope with jCal data where the parser was unable to identify the type. Note how the "unknown" value type is not added to the iCalendar data and escaping, aside from standard JSON string escaping, is not processed.

jCal:

```
["x-coffee-data", {}, "unknown", "Stenophylla;Guinea\\,Africa"]
```

iCalendar:

```
X-COFFEE-DATA:Stenophylla;Guinea\,Africa
```

The following is an example of a jCal property (where the corresponding iCalendar property uses an "INTEGER" value as its default) and the equivalent iCalendar representation of that property.

jCal:

```
["percent-complete", {}, "integer", 95]
```

iCalendar:

```
PERCENT-COMPLETE:95
```

The following is an example of an unrecognized iCalendar property parameter (that uses a "FLOAT" value as its default) specified on a recognized iCalendar property and the equivalent jCal representation of that property and property parameter.

iCalendar:

```
DTSTART;X-SLACK=30.3;VALUE=DATE:20110512
```

jCal:

```
["dtstart", { "x-slack": "30.3" }, "date", "2011-05-12"]
```

6. Security Considerations

This specification defines how iCalendar data can be "translated" between two different data formats -- the original text format and JSON -- with a one-to-one mapping to ensure all the semantic data in one format (properties, parameters, and values) are preserved in the other. It does not change the semantic meaning of the underlying data itself, or impose or remove any security considerations that apply to the underlying data.

The use of JSON as a format does have its own inherent security risks as discussed in Section 12 of [RFC7159]. Even though JSON is considered a safe subset of JavaScript, it should be kept in mind that a flaw in the parser processing JSON could still impose a threat, which doesn’t arise with conventional iCalendar data.

With this in mind, a parser for JSON data should be used for jCal that is aware of the security implications. For example, the use of JavaScript’s eval() function is considered an unacceptable security risk, as described in Section 12 of [RFC7159]. A native parser with full awareness of the JSON format should be preferred.

In addition, it is expected that this new format will result in iCalendar data being more widely disseminated (e.g., with use in web applications rather than just dedicated calendaring applications).

In all cases, application developers have to conform to the semantics of the iCalendar data as defined by [RFC5545] and associated extensions, and all of the security considerations described in Section 7 of [RFC5545], or any associated extensions, are applicable.

7. IANA Considerations

This document defines a MIME media type for use with iCalendar in JSON data. This media type SHOULD be used for the transfer of calendaring data in JSON.

Type name: application

Subtype name: calendar+json

Required parameters: none

Optional parameters: "method", "component", and "optinfo" as defined for the text/calendar media type in [RFC5545], Section 8.1.

Encoding considerations: Same as encoding considerations of application/json as specified in [RFC7159], Section 11.

Security considerations: See Section 6.

Interoperability considerations: This media type provides an alternative format for iCalendar data based on JSON.

Published specification: This specification.

Applications that use this media type: Applications that currently make use of the text/calendar media type 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: N/A

Additional information:

Deprecated alias names for this type: N/A

Magic number(s): N/A

File extension(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: There are no restrictions on where this media type can be used.

Author: See the "Authors' Addresses" section of this document.

Change controller: IETF

7.1. UNKNOWN iCalendar Value Data Type

IANA has added the following entry to the iCalendar Data Types registry:

Value name: UNKNOWN

Purpose: To allow preserving property values whose default value type is not known during round-tripping between jCal and iCalendar.

Format definition: N/A

Description: The UNKNOWN value data type is reserved for the exclusive use of the jCal format. Its use is described in Section 5 of this document.

Example: As this registration serves as a reservation of the UNKNOWN type so that it is not used in iCalendar, there is no applicable iCalendar example. Examples of its usage in jCal can be found in this document.

IANA has made the "Status" column for this entry in the registry say, "Reserved - Do not use" and has made the "Reference" column refer to Section 5 of this document.

8. Acknowledgments

The authors would like to thank the following for their valuable contributions: William Gill, Erwin Rehme, Dave Thewlis, Simon Perreault, Michael Angstadt, Peter Saint-Andre, Bert Greevenbosch, and Javier Godoy. This specification originated from the work of the XML-JSON technical committee of the Calendaring and Scheduling Consortium.

9. References

9.1. Normative References

[ISO.8601.2004] International Organization for Standardization, "Data elements and interchange formats -- Information interchange -- Representation of dates and times", ISO 8601, December 2004, <http://www.iso.org/iso/catalogue_detail?csnumber=40874>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006.

[RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

[RFC5545] Desruisseaux, B., "Internet Calendaring and Scheduling Core Object Specification (iCalendar)", RFC 5545, September 2009.

[RFC6321] Daboo, C., Douglass, M., and S. Lees, "xCal: The XML Format for iCalendar", RFC 6321, August 2011.

[RFC6868] Daboo, C., "Parameter Value Encoding in iCalendar and vCard", RFC 6868, February 2013.

[RFC7159] Bray, T., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, March 2014.

9.2. Informative References

[calconnect-artifacts]
The Calendaring and Scheduling Consortium, "Code Artifacts and Schemas", <http://www.calconnect.org/artifacts.shtml>.

Appendix A. ABNF Schema

Below is an ABNF schema as per [RFC5234] for iCalendar in JSON. ABNF symbols not described here are taken from [RFC7159]. The schema is non-normative and given for reference only.

Additional semantic restrictions apply, especially regarding the allowed properties and sub-components per component. Details on these restrictions can be found in this document and [RFC5545].

Additional schemas may be available on the Internet at [calconnect-artifacts].

```
; A jCal object is a component with the component-name "vcalendar".
; Restrictions to which properties and sub-components may be
; specified are to be taken from [RFC5545].
jcalobject = component

; A jCal component consists of the name string, properties array, and
; component array
component = begin-array
            DQUOTE component-name DQUOTE value-separator
            properties-array value-separator
            components-array
            end-array

components-array = begin-array
                   [ component *(value-separator component) ]
                   end-array

; A jCal property consists of the name string, parameters object,
; type string, and one or more values as specified in this document.
property = begin-array
           DQUOTE property-name DQUOTE value-separator
           params-object value-separator
           DQUOTE type-name DQUOTE
           property-value *(value-separator property-value)
           end-array
properties-array = begin-array
                   [ property *(value-separator property) ]
                   end-array

; Property values depend on the type-name. Aside from the value types
; mentioned here, extensions may make use of other JSON value types.
; The non-terminal symbol structured-prop-value covers the special
; cases for GEO and REQUEST-STATUS.
property-value = simple-prop-value / structured-prop-value
simple-prop-value = string / number / true / false
```

```
   structured-prop-value =
       begin-array
       [ structured-element *(value-separator structured-element) ]
       end-array
   structured-element = simple-prop-value

   ; The jCal params-object is a JSON object that follows the semantic
   ; guidelines described in this document.
   params-object = begin-object
                   [ params-member *(value-separator params-member) ]
                   end-object
   params-member = DQUOTE param-name DQUOTE name-separator param-value
   param-value = string / param-multi
   param-multi = begin-array
                 [ string *(value-separator string) ]
                 end-array

   ; The type MUST be a valid type as described by this document. New
   ; value types can be added by extensions.
   type-name = "binary" / "boolean" / "cal-address" / "date" /
               "date-time" / "duration" / "float" / "integer" /
               "period" / "recur" / "text" / "time" / "uri" /
               "utc-offset" / x-type


   ; Component, property, parameter, and type names MUST be lowercase.
   ; Additional semantic restrictions apply as described by this
   ; document and [RFC5545].
   component-name = lowercase-name
   property-name = lowercase-name
   param-name = lowercase-name
   x-type = lowercase-name
   lowercase-name = 1*(%x61-7A / DIGIT / "-")

   ; The following rules are defined in [RFC7159], as mentioned above:
   ;   begin-array / end-array
   ;   begin-object / end-object
   ;   name-separator / value-separator
   ;   string / number / true / false
```

Appendix B. Examples

This section contains two examples of iCalendar objects with their jCal representation.

B.1. Example 1

B.1.1. iCalendar Data

```
BEGIN:VCALENDAR
CALSCALE:GREGORIAN
PRODID:-//Example Inc.//Example Calendar//EN
VERSION:2.0
BEGIN:VEVENT
DTSTAMP:20080205T191224Z
DTSTART:20081006
SUMMARY:Planning meeting
UID:4088E990AD89CB3DBB484909
END:VEVENT
END:VCALENDAR
```

B.1.2. jCal Data

```
["vcalendar",
  [
    ["calscale", {}, "text", "GREGORIAN"],
    ["prodid", {}, "text", "-//Example Inc.//Example Calendar//EN"],
    ["version", {}, "text", "2.0"]
  ],
  [
    ["vevent",
      [
        ["dtstamp", {}, "date-time", "2008-02-05T19:12:24Z"],
        ["dtstart", {}, "date", "2008-10-06"],
        ["summary", {}, "text", "Planning meeting"],
        ["uid", {}, "text", "4088E990AD89CB3DBB484909"]
      ],
      []
    ]
  ]
]
```

B.2. Example 2

B.2.1. iCalendar Data

```
BEGIN:VCALENDAR
VERSION:2.0
PRODID:-//Example Corp.//Example Client//EN
BEGIN:VTIMEZONE
LAST-MODIFIED:20040110T032845Z
TZID:US/Eastern
BEGIN:DAYLIGHT
DTSTART:20000404T020000
RRULE:FREQ=YEARLY;BYDAY=1SU;BYMONTH=4
TZNAME:EDT
TZOFFSETFROM:-0500
TZOFFSETTO:-0400
END:DAYLIGHT
BEGIN:STANDARD
DTSTART:20001026T020000
RRULE:FREQ=YEARLY;BYDAY=-1SU;BYMONTH=10
TZNAME:EST
TZOFFSETFROM:-0400
TZOFFSETTO:-0500
END:STANDARD
END:VTIMEZONE
BEGIN:VEVENT
DTSTAMP:20060206T001121Z
DTSTART;TZID=US/Eastern:20060102T120000
DURATION:PT1H
RRULE:FREQ=DAILY;COUNT=5
RDATE;TZID=US/Eastern;VALUE=PERIOD:20060102T150000/PT2H
SUMMARY:Event #2
DESCRIPTION:We are having a meeting all this week at 12 pm fo
 r one hour\, with an additional meeting on the first day 2 h
 ours long.\nPlease bring your own lunch for the 12 pm meetin
 gs.
UID:00959BC664CA650E933C892C@example.com
END:VEVENT
BEGIN:VEVENT
DTSTAMP:20060206T001121Z
DTSTART;TZID=US/Eastern:20060104T140000
DURATION:PT1H
RECURRENCE-ID;TZID=US/Eastern:20060104T120000
SUMMARY:Event #2 bis
UID:00959BC664CA650E933C892C@example.com
END:VEVENT
END:VCALENDAR
```

B.2.2. jCal Data

```
   ["vcalendar",
     [
       ["prodid", {}, "text", "-//Example Corp.//Example Client//EN"],
       ["version", {}, "text", "2.0"]
     ],
     [
       ["vtimezone",
         [
           ["last-modified", {}, "date-time", "2004-01-10T03:28:45Z"],
           ["tzid", {}, "text", "US/Eastern"]
         ],
         [
           ["daylight",
             [
               ["dtstart", {}, "date-time", "2000-04-04T02:00:00"],
               ["rrule",
                 {},
                 "recur",
                 {
                   "freq": "YEARLY",
                   "byday": "1SU",
                   "bymonth": 4
                 }
               ],
               ["tzname", {}, "text", "EDT"],
               ["tzoffsetfrom", {}, "utc-offset", "-05:00"],
               ["tzoffsetto", {}, "utc-offset", "-04:00"]
             ],
             []
           ],
           ["standard",
             [
               ["dtstart", {}, "date-time", "2000-10-26T02:00:00"],
               ["rrule",
                 {},
                 "recur",
                 {
                   "freq": "YEARLY",
                   "byday": "1SU",
                   "bymonth": 10
                 }
               ],
               ["tzname", {}, "text", "EST"],
               ["tzoffsetfrom", {}, "utc-offset", "-04:00"],
               ["tzoffsetto", {}, "utc-offset", "-05:00"]
             ],
```

```
            []
          ]
        ]
      ],
      ["vevent",
        [
          ["dtstamp", {}, "date-time", "2006-02-06T00:11:21Z"],
          ["dtstart",
            { "tzid": "US/Eastern" },
            "date-time",
            "2006-01-02T12:00:00"
          ],
          ["duration", {}, "duration", "PT1H"],
          ["rrule", {}, "recur", { "freq": "DAILY", "count": 5 } ],
          ["rdate",
            { "tzid": "US/Eastern" },
            "period",
            "2006-01-02T15:00:00/PT2H"
          ],
          ["summary", {}, "text", "Event #2"],
          ["description",
           {},
           "text",
           // Note that comments and string concatenation are not
           // allowed per the JSON specification and is used here only
           // to avoid long lines.
           "We are having a meeting all this week at 12 pm for one " +
           "hour, with an additional meeting on the first day 2 " +
           "hours long.\nPlease bring your own lunch for the 12 pm " +
           "meetings."
          ],
          ["uid", {}, "text", "00959BC664CA650E933C892C@example.com"]
        ],
        []
      ],
      ["vevent",
        [
          ["dtstamp", {}, "date-time", "2006-02-06T00:11:21Z"],
          ["dtstart",
            { "tzid": "US/Eastern" },
            "date-time",
            "2006-01-02T14:00:00"
          ],
          ["duration", {}, "duration", "PT1H"],
          ["recurrence-id",
            { "tzid": "US/Eastern" },
            "date-time",
            "2006-01-04T12:00:00"
```

```
            ],
            ["summary", {}, "text", "Event #2"],
            ["uid", {}, "text", "00959BC664CA650E933C892C@example.com"]
          ],
          []
        ]
      ]
    ]
```

Authors' Addresses

Philipp Kewisch
Mozilla Corporation
650 Castro Street, Suite 300
Mountain View, CA 94041
USA

EMail: mozilla@kewis.ch
URI: http://www.mozilla.org/

Cyrus Daboo
Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
USA

EMail: cyrus@daboo.name
URI: http://www.apple.com/

Mike Douglass
Rensselaer Polytechnic Institute
110 8th Street
Troy, NY 12180
USA

EMail: douglm@rpi.edu
URI: http://www.rpi.edu/