

JSON data formats for vCard and iCalendar
Internet-Draft
Intended status: Standards Track
Expires: January 04, 2014

P. Kewisch
Mozilla
July 03, 2013

jCard: The JSON format for vCard
draft-ietf-jcardcal-jcard-04

Abstract

This specification defines "jCard", a JSON format for vCard data.

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 <http://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 January 04, 2014.

Copyright Notice

Copyright (c) 2013 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 (<http://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 Simplified BSD License text as described in [Section 4.e](#) of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
---------------------------	---

2.	Conventions Used in This Document	3
3.	Converting from vCard to jCard	4
3.1.	Pre-processing	4
3.2.	Properties (RFC6350 section 6)	5
3.2.1.	Special Cases for Properties	6
3.2.1.1.	The VERSION Property	6
3.2.1.2.	Multi-valued Properties	6
3.2.1.3.	Grouping of Properties	6
3.2.1.4.	Structured Property Values	7
3.3.	Parameters (RFC6350 Section 5)	9
3.3.1.	VALUE parameter	10
3.3.2.	Multi-value Parameters	10
3.4.	Values (RFC6350 Section 4)	10
3.4.1.	Text (RFC6350 Section 4.1)	11
3.4.2.	URI (RFC6350 Section 4.2)	11
3.4.3.	Date (RFC6350 Section 4.3.1)	11
3.4.4.	Time (RFC6350 Section 4.3.2)	12
3.4.5.	Date-Time (RFC6350 Section 4.3.3)	13
3.4.6.	Date and/or Time (RFC6350 Section 4.3.4)	15
3.4.7.	Timestamp (RFC6350 Section 4.3.5)	15
3.4.8.	Boolean (RFC6350 Section 4.4)	16
3.4.9.	Integer (RFC6350 Section 4.5)	16
3.4.10.	Float (RFC6350 Section 4.6)	16
3.4.11.	UTC Offset (RFC6350 Section 4.7)	16
3.4.12.	Language Tag (RFC6350 Section 4.8)	17
3.5.	Extensions (RFC6350 Section 6.10)	17
4.	Converting from jCard into vCard	17
5.	Handling Unrecognized Properties or Parameters	18
5.1.	Converting vCard into jCard	19
5.2.	Converting jCard into vCard	19
5.3.	Examples	19
6.	Implementation Status (to be removed prior to publication as an RFC)	20
7.	Security Considerations	21
8.	IANA Considerations	21
8.1.	GROUP vCard Parameter	22
8.2.	UNKNOWN vCard Value Data Type	23
9.	Acknowledgments	23
10.	References	23
10.1.	Normative References	23
10.2.	Informative References	24
Appendix A.	ABNF Schema	24
Appendix B.	Examples	26
B.1.	Example: vCard of the author of RFC6350	26
B.1.1.	vCard Data	26
B.1.2.	jCard Data	27
Appendix C.	Change History (to be removed prior to publication as an RFC)	28

Author's Address	29
----------------------------	----

1. Introduction

The vCard data format [RFC6350] provides for the capture and exchange of information normally stored within an address book or directory application. The vCard format has gone through multiple revisions, most recently vCard 4. The purpose of this specification is to define "jCard", a JSON format for vCard data. One main advantage to using a JSON-based format as defined in [RFC4627] over the classic vCard format is easier processing for JavaScript based widgets and libraries, especially in the scope of web-based applications.

As certain similarities exist between vCard and the iCalendar data format [RFC5545], there is also an effort to define a JSON-based data format for calendar information called jCal [I-D.ietf-jcardcal-jcal] that parallels the format defined in this specification.

The purpose of this specification is to define "jCard", a JSON format for vCard data. One main advantage to using a JSON-based format as defined in [RFC4627] over the classic vCard format is easier processing for JavaScript based widgets and libraries, especially in the scope of web-based applications.

The key design considerations are essentially the same as those for [I-D.ietf-jcardcal-jcal] and [RFC6321], that is:

Round-tripping (converting a vCard instance to jCard 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 will not necessarily be preserved.

Preserve the semantics of the vCard data. While a simple consumer can easily browse the data in jCard, a full understanding of vCard is still required in order to modify and/or fully comprehend the directory data.

Ability to handle many extensions to the underlying vCard specification without requiring an update to this document.

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 jCard is JSON. Consequently, the terms "object" and "array" as well as the four primitive types are to be interpreted as described in [Section 1 of \[RFC4627\]](#).

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 vCard to jCard

This section describes how vCard objects are converted to jCard using a simple mapping between the vCard data model and JSON elements.

In [\[RFC6350\]](#), vCard objects are comprised of a set of "properties", "parameters" and "values". The top level of a vCard object contains "properties". A "property" has a "value" and a set of zero or more "parameters". Each of these entities have a representation in jCard, defined in the following sections. The representation of a vCard object in JSON will be named "jCard component" throughout this document.

3.1. Pre-processing

vCard uses a line folding mechanism to limit lines of data to a maximum line length (typically 72 characters) to ensure maximum likelihood of preserving data integrity as it is transported via various means (e.g., email) - see [Section 3.2 of \[RFC6350\]](#). Prior to converting vCard data into jCard all folded lines MUST be unfolded.

vCard data uses an "escape" character sequence for text values and property parameter values. See [Section 3.4 of \[RFC6350\]](#) as well as [\[RFC6868\]](#). When such text elements are converted into jCard, the vCard escaping MUST be removed first. The only escaping that may be applied is any escaping mandated by JSON.

One key difference in the formatting of values used in vCard and jCard is that in jCard the specification uses date/time values aligned with the extended format of [\[ISO.8601.2004\]](#). 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. Properties (RFC6350 section 6)

Each individual vCard property is represented in jCard by an array with three fixed elements, followed by one or more additional elements, depending on if the property is a multi-value property as described in [Section 3.3 of \[RFC6350\]](#).

The array consists of the following fixed elements:

1. The name of the property as a string, but in lowercase.
2. An object containing the parameters as described in [Section 3.3](#).
3. The type identifier string of the value, in lowercase.

The remaining elements of the array are used for the value of the property. For single-value properties, the array MUST have exactly four elements, for multi-valued properties as described in [Section 3.2.1.2](#) there can be any number of additional elements.

The array describing the property can then be inserted into the array designated for properties in the jCard component.

Example:

```
[ "vcard",  
  [  
    [ "version", {}, "text", "4.0"],  
    [ "fn", {}, "text", "John Doe"],  
    [ "gender", {}, "text", "M"],  
    ...  
  ]  
]
```

The property parameters in the second element of the property array associate a set of parameter names with their respective value. Parameters are further described in [Section 3.3](#).

To allow for a cleaner implementation, the parameter object MUST be present even if there are no parameters. In this case, an empty object MUST be used.

As described in [Section 3.2.1.4](#), it is important for a parser check the data type of the value even if it is assumed to be a string in most cases. The value could turn out to be a structured value, in which case the data type is an array.

3.2.1. Special Cases for Properties

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

3.2.1.1. The VERSION Property

The vCard format is versioned. [RFC6350] defines the VERSION property to be mandatory, therefore a "version" property MUST also be specified in a jCard component. Similarly, the value of the version property MUST be "4.0".

3.2.1.2. Multi-valued Properties

Various vCard properties defined in [RFC6350], for example the "CATEGORIES" property, are defined as multi-valued properties. In jCal these properties are added as further members of the array describing the property.

Note that additional multi-valued properties may be added in extensions to the iCalendar format.

Example:

```
[ "vcard",  
  [  
    [ "categories", {}, "text", "computers", "cameras"],  
    ...  
  ]  
]
```

3.2.1.3. Grouping of Properties

[RFC6350] [Section 3.3](#) defines a grouping construct that is used to group related properties together. To eliminate the need for an extra group syntax in jCard, thus unifying the general syntax with that of jCal, this specification introduces a "group" parameter, that holds the group name. The parameter MUST NOT be used in vCard as per [RFC6350], it is merely registered to reserve the parameter, avoiding collisions.

Namespace: <empty>

Parameter name: GROUP

Purpose: To simplify the jCard format.

Description: The GROUP parameter is reserved for the exclusive use of the jCard format RFCTODO. It MUST NOT be used in plain vCard [RFC6350], nor in xCard [RFC6351].

Format definition: (Not applicable)

Example: As this registration serves as a reservation of the GROUP parameter so that it is not used in vCard, there is no applicable vCard example. For examples of its usage in jCard, see RFCTODO

3.2.1.3.1. Group Conversion Rules

In jCard, the parameter's value is a single opaque string. Conversion rules are as follows:

- o From vCard to jCard, the group construct (see [RFC6350], Section 3.3, Page 7) is removed. In its place, the GROUP parameter is added (lowercased, as any other parameter). Its value is a string corresponding to the group name. The name's case MUST be preserved intact.
- o From jCard to vCard, the reverse procedure is performed. The GROUP parameter MUST NOT appear in the resulting vCard.

Example:

CONTACT.FN:Mr. John Q. Public\, Esq.

is equivalent to:

```
[ "fn", { "group": "CONTACT" }, "text", "Mr. John Q. Public, Esq." ]
```

3.2.1.4. Structured Property Values

The vCard specification defines properties with structured values, for example GENDER or ADR. A structured value is defined as a value that contains multiple text components, delimited by the SEMICOLON character. In jCard, the property value is an array containing one element for each text component.

vCard Example:

ADR;;;123 Main Street;Any Town;CA;91921-1234;U.S.A.

jCard Example:

```
[ "adr", {}, "text",  
  [  
    "", "", "123 Main Street",  
    "Any Town", "CA", "91921-1234", "U.S.A."  
  ]  
]
```

Some vCard properties, for example ADR, also allow a structured value element that itself has multiple values. In this case, the element of the array describing the structured value is itself an array with one element for each of the component's multiple values.

vCard Example:

ADR;;;My Street,Left Side,Second Shack;Hometown;PA;18252;U.S.A.

jCard Example:

```
[ "adr", {}, "text",  
  [  
    "", "",  
    [ "My Street", "Left Side", "Second Shack"],  
    "Hometown", "PA", "18252", "U.S.A."  
  ]  
]
```

In both cases, the array element values MUST have the primitive type that matches the jCard type identifier. In [\[RFC6350\]](#), there are only structured text values and thus only JSON strings are used. Extensions may for example define structured number or boolean values, where JSON number or boolean types MUST be used.

If a multi-valued text component is changed to hold only one value, the text component SHOULD be represented as a single primitive value, dropping the array completely. Nevertheless, a more simple implementation might choose to retain the array, with a single primitive value as its element.

Similarly, structured values that consist of two text components with one being optional (for example, GENDER) can be represented as a single text value. Therefore, parsers of jCard data SHOULD check even known property values for structured information by considering the JSON data type of the value, which can be an array or a primitive

value. This is especially important for languages where accessing array members is done by the same construct as accessing characters of a string.

Examples:

```
[ "gender", {}, "text", ["F", "grrrl"] ],  
[ "gender", {}, "text", "M" ]
```

Per [\[RFC6350\] Section 6.3.1](#), the component separator MUST be specified even if the component value is missing. Similarly, the jCard array containing the structured data MUST contain all required elements, even if they are empty.

vCard Example:

```
ADR;LABEL="123 Maple Ave\nSuite 901\nVancouver BC\nA1B 2C9\nCanada":;;;;;
```

jCard Example:

```
[ "adr",  
  { "label": "123 Maple Ave\nSuite 901\nVancouver BC\nA1B 2C9\nCanada"},  
  "text",  
  [ "", "", "", "", "", "", "" ]  
]
```

3.3. Parameters ([RFC6350 Section 5](#))

Property parameters are represented as a JSON object where each key-value pair represents the vCard 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 "LANG" property parameter is represented in jCard by the "lang" key. Any new vCard parameters added in the future will be converted in the same way.

Example:

```
[ "vcard",  
  [  
    [ "role", { "lang": "tr" }, "text", "roca"],  
    ...  
  ]  
]
```

3.3.1. VALUE parameter

vCard defines a "VALUE" property parameter ([Section 5.2 of \[RFC6350\]](#)). This property parameter MUST NOT be added to the parameters object. Instead, the value type is always explicitly mentioned in the third element of the array describing the property. Thus, when converting from vCard to jCard, any "VALUE" property parameters are skipped. When converting from jCard into vCard, the appropriate "VALUE" property parameter MUST be included in the vCard property if the value type is not "unknown" or the default value type for that property. See [Section 5](#) for information on handling unknown value types.

3.3.2. Multi-value Parameters

In [\[RFC6350\]](#), some parameters allow using a COMMA-separated list of values. To ease processing in jCard, the value to such parameters MUST be represented in an array containing the separated values. The array elements MUST be string values. Single-value parameters SHOULD be represented using a single string value, although a more simple implementation might prefer an array with one string element. An example for a such parameter is the vCard "SORT-AS" parameter, more such parameters may be added in extensions.

DQUOTE characters used to encapsulate the separated values MUST NOT be added to the jCard parameter value.

Example 1:

```
[ "vcard",
  [
    [ "n",
      { "sort-as": [ "Harten", "Rene" ] },
      "text",
      "van der Harten;Rene,J.;Sir;R.D.O.N."
    ],
    [ "fn", { }, "text", "Rene van der Harten" ]
    ...
  ]
]
```

3.4. Values ([RFC6350 Section 4](#))

The type of a vCard value is explicitly mentioned in the third element of the array describing a jCard property. The actual values of the property can be found in the fourth and following elements of the array.

3.4.1. Text ([RFC6350 Section 4.1](#))

Description: vCard "TEXT" property values are represented by a property with the type identifier "text". The value elements are JSON strings. For details on structured text values, see [Section 3.2.1.4](#).

Example:

```
["kind", {}, "text", "group"]
```

3.4.2. URI ([RFC6350 Section 4.2](#))

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

Example:

```
["source", {}, "uri", "ldap://ldap.example.com/cn=babs%20jensen"]
```

3.4.3. Date ([RFC6350 Section 4.3.1](#))

Description: vCard "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 [\[RFC6350\]](#), but represented using the extended format specified in [\[ISO.8601.2004\]](#), Section 4.1.2. If the complete representation is not used, the same date format restrictions regarding reduced accuracy, truncated representation and expanded representation noted in [\[RFC6350\] Section 4.1.2.3](#) apply. Whenever the extended format is not applicable, the basic format MUST be used.

ABNF Schema:

```
date-complete = year "-" month "-" day ;YYYY-MM-DD
```

```
date-noreduc = date-complete  
              / "--" month "-" day; --MM-DD  
              / "----" day; ---DDD
```

```
date = date-noreduc  
      / year; YYYY  
      / year "-" month ; YYYY-MM  
      / "--" month; --MM
```

Examples:

```
[ "bday", {}, "date", "1985-04-12"],
[ "bday", {}, "date", "1985-04"],
[ "bday", {}, "date", "1985"],
[ "bday", {}, "date", "--04-12"],
[ "bday", {}, "date", "---12"]
```

This table contains possible conversions between the vCard DATE format its jCard date. This information is just an example and not a formal specification of the syntax. The specification can be found in [ISO.8601.2000] and [ISO.8601.2004]:

	vCard	jCard
Complete	19850412	1985-04-12
Reduced	1985-04	1985-04
Reduced	1985	1985
Truncated	--0412	--04-12
Truncated	--04	--04
Truncated	---12	---12

3.4.4. Time (RFC6350 Section 4.3.2)

Description: vCard "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 [RFC6350], but represented using the extended format specified in [ISO.8601.2004], Section 4.2. If the complete representation is not used, the same time format restrictions regarding reduced accuracy, decimal fraction and truncated representation noted in [RFC6350] Section 4.3.2 apply. Whenever the extended format is not applicable, the basic format MUST be used. The seconds value of 60 MUST only be used to account for positive "leap" seconds and the midnight hour is always represented by 00, never 24. Fractions of a second are not supported by this format. Contrary to [I-D.ietf-jcardcal-jcal], UTC offsets are permitted within a time value.

ABNF Schema:

```
time-notrunc = hour [":" minute [":" second]] [zone]
```

```
time = time-notrunc
      / "-" minute ":" second [zone]; -mm:ss
      / "-" minute [zone]; -mm
      / "--" second [zone]; --ss
```

Examples:

```
["x-time-local", {}, "time", "12:30:00"],
["x-time-utc", {}, "time", "12:30:00Z"],
["x-time-offset", {}, "time", "12:30:00-08:00"],
["x-time-reduced", {}, "time", "23"],
["x-time-truncated", {}, "time", "-30"]
```

This table contains possible conversions between the vCard TIME format its jCard time. This information is just an example and not a formal specification of the syntax. The specification can be found in [\[ISO.8601.2000\]](#) and [\[ISO.8601.2004\]](#):

	vCard	jCard
Complete	232050	23:20:50
Reduced	2320	23:20
Reduced	23	23
Truncated	-2050	-20:50
Truncated	-20	-20
Truncated	--50	--50

Also, all combinations may have any zone designator appended, as in the complete representation.

3.4.5. Date-Time ([RFC6350 Section 4.3.3](#))

Description: vCard "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 [RFC6350], but represented using the extended format specified in [ISO.8601.2004], Section 4.3. If the complete representation is not used, the same date and time format restrictions as in Section 3.4.4 and Section 3.4.3 apply. Just as in [RFC6350], truncation of the date part is permitted.

Example:

```
[ "anniversary", {}, "date-time", "2013-02-14T12:30:00"],
[ "anniversary", {}, "date-time", "2013-01-10T19:00:00Z"],
[ "anniversary", {}, "date-time", "2013-08-15T09:45:00+01:00"],
[ "anniversary", {}, "date-time", "---15T09:45:00+01:00"]
```

This table contains possible conversions between the vCard DATE-TIME format its jCard date-time. This information is just an example and not a formal specification of the syntax. The specification can be found in [ISO.8601.2000] and [ISO.8601.2004]:

Representation	vCard	jCard
Complete	19850412T232050	1985-04-12T23:20:50
Complete	19850412T232050Z	1985-04-12T23:20:50Z
Complete	19850412T232050+0400	1985-04-12T23:20:50+04:00
Complete	19850412T232050+04	1985-04-12T23:20:50+04
Reduced	19850412T2320	1985-04-12T23:20
Reduced	19850412T23	1985-04-12T23
Truncated and Reduced	--0412T2320	--04-12T23:20
Truncated and Reduced	--04T2320	--04T23:20
Truncated and Reduced	---12T2320	---12T23:20
Truncated and Reduced	--0412T2320	--04-12T23:20
Truncated and Reduced	--04T23	--04T23

Reduced		
---------	--	--

As specified in [ISO.8601.2000], the date component shall not be represented with reduced accuracy and the time component shall not be truncated. Also, all combinations may have any zone designator appended, as in the complete representation.

3.4.6. Date and/or Time (RFC6350 Section 4.3.4)

Description: vCard "DATE-AND-OR-TIME" property values are represented by a property with the type identifier "date-and-or-time". The value elements are either a date-time (Section 3.4.5), a date (Section 3.4.3) or a time (Section 3.4.4) value. Just as in [RFC6350] Section 4.3.4, a stand-alone time value MUST always be preceded by a "T".

Example:

```
[ "bday", {}, "date-and-or-time", "2013-02-14T12:30:00" ],
[ "bday", {}, "date-and-or-time", "---22T14:00" ]
[ "bday", {}, "date-and-or-time", "1985" ],
[ "bday", {}, "date-and-or-time", "T12:30" ]
```

3.4.7. Timestamp (RFC6350 Section 4.3.5)

Description: vCard "TIMESTAMP" property values are represented by a property with the type identifier "timestamp". The value elements are JSON strings with the same timestamp value specified by [RFC6350], but represented using the extended format and complete representation specified in [ISO.8601.2004], Section 4.3.2.

Example:

```
[ "rev", {}, "timestamp", "2013-02-14T12:30:00" ],
[ "rev", {}, "timestamp", "2013-02-14T12:30:00Z" ],
[ "rev", {}, "timestamp", "2013-02-14T12:30:00-05" ],
[ "rev", {}, "timestamp", "2013-02-14T12:30:00-05:00" ]
```

This table contains possible conversions between the vCard TIMESTAMP format its jCard timestamp. This information is just an example and not a formal specification of the syntax. The specification can be found in [ISO.8601.2000] and [ISO.8601.2004]:

Representation	vCard	jCard
Complete	19850412T232050	1985-04-12T23:20:50
Complete	19850412T232050Z	1985-04-12T23:20:50Z
Complete	19850412T232050+0400	1985-04-12T23:20:50+04:00
Complete	19850412T232050+04	1985-04-12T23:20:50+04

3.4.8. Boolean ([RFC6350 Section 4.4](#))

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

Example:

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

3.4.9. Integer ([RFC6350 Section 4.5](#))

Description: vCard "INTEGER" property values are represented by a property with the type identifier "integer". The value elements are JSON primitive number values.

Examples:

```
["x-karma-points", {}, "integer", 42]
```

3.4.10. Float ([RFC6350 Section 4.6](#))

Description: vCard "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]
```

3.4.11. UTC Offset ([RFC6350 Section 4.7](#))

Description: vCard "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 [RFC6350], with the exception that the hour and minute components are separated by a ":" character, for consistency with the [ISO.8601.2004] timezone offset, extended format.

Example:

```
// Note: [RFC6350] mentions use of utc-offset
// for the TZ property as NOT RECOMMENDED
["tz", {}, "utc-offset", "-05:00"]
```

3.4.12. Language Tag (RFC6350 Section 4.8)

Description: vCard "LANGUAGE-TAG" property values are represented by a property with the type identifier "language-tag". The value elements are JSON strings containing a single language-tag, as defined in [RFC5646].

Example:

```
["lang", {}, "language-tag", "de"]
```

3.5. Extensions (RFC6350 Section 6.10)

vCard 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, the property parameter represented by an object. The property or parameter name uses the same name as for the vCard extension, but in lowercase. For example, the "X-FOO" property in vCard turns into the "x-foo" jCard property. See Section 5 for how to deal with default values for unrecognized extension properties or property parameters.

4. Converting from jCard into vCard

When converting property and property parameter values, the names SHOULD be converted to uppercase. Although vCard names are case insensitive, common practice is to keep them all uppercase following the actual definitions in [RFC6350].

Character escaping and line folding MUST be applied to the resulting vCard data as required by [RFC6350] and [RFC6868].

When converting to vCard, the VALUE parameter MUST be added to properties whose default value type is unknown. The VALUE parameter MAY be omitted for properties using the default value type.

5. Handling Unrecognized Properties or Parameters

In vCard, 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, "BDAY"'s default value type is "date-and-or-time", so "VALUE=date-and-or-time" need not be set as a property parameter. However, "BDAY" also allows a "text" value to be specified, and if that is used, "VALUE=text" has to be set as a property parameter.

When new properties are defined or "X-" properties used, a vCard to jCard 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 jCard, a new "unknown" property value type is introduced. Its purpose is to allow preserving unknown property values when round-tripping between jCard and vCard. To avoid collisions, this specification reserves the UNKNOWN property value type in vCard. It MUST NOT be used in any vCard as specified by [\[RFC6350\]](#), nor any extensions to it.

Value name: UNKNOWN

Purpose: To allow preserving unknown property values during round-tripping between jCard and vCard

Format definition: (Not applicable)

Description: The UNKNOWN value data type is reserved for the exclusive use of the jCard format RFCTODO. It MUST NOT be used in plain vCard [\[RFC6350\]](#).

Example: As this registration serves as a reservation of the UNKNOWN type so that it is not used in vCard, there is no applicable vCard example. For examples of its usage in jCard, see RFCTODO

5.1. Converting vCard into jCard

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 if the default type is not known that the type "unknown" is used. 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, treated as if it were a "TEXT" value.

5.2. Converting jCard into vCard

Since jCard always explicitly specifies the value type, it can always be converted to vCard using the VALUE parameter.

If the value type specified in jCard matches the default value type in vCard, the VALUE parameter MAY be omitted.

If the value type specified in jCard is set to "unknown", the value MUST be taken over in vCard without processing. In this case, the VALUE parameter MUST NOT be specified.

5.3. Examples

The following is an example of an unrecognized vCard property (that uses an "URI" value as its default), and the equivalent jCard representation of that property.

vCard:

X-COMPLAINT-URI:mailto:abuse@example.org

jCard:

```
["x-complaint-uri", {}, "unknown", "mailto:abuse@example.org"]
```

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

jCard:

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

vCard:

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

The following is an example of a jCard property (where the corresponding vCard property uses a "INTEGER" value as its default), and the equivalent vCard representation of that property. It is assumed that the parser has knowledge of the default data type for the "x-karma-points" property.

jCard:

```
["x-karma-points", {}, "integer", 95]
```

vCard:

```
X-KARMA-POINTS:95
```

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

vCard:

```
GENDER;X-PROBABILITY=0.8:M
```

jCard:

```
["gender", { "x-probability": "0.8" }, "text", "M"]
```

6. Implementation Status (to be removed prior to publication as an RFC)

This section describes libraries known to implement this draft as per [\[I-D.sheffer-running-code\]](#).

1. ICAL.js - Philipp Kewisch, James Lal. A JavaScript parser for iCalendar ([rfc5545](#))

Source: <https://github.com/mozilla-comm/ical.js/>

Maturity: alpha (for jCard)

Coverage: Currently geared towards jCal, therefore not all formats are supported. Includes an online validator. (as of rev 847c67c501, 2013-02-14)

Licensing: MPL, Mozilla Public License 2.0

2. Py Calendar - Cyrus Daboo. iCalendar/vCard Library

Source: <https://svn.calendarserver.org/repository/calendarserver/PyCalendar/branches/json/>

Maturity: production

Coverage: All aspects of this draft, up to version 01.

Licensing: Apache License, Version 2.0

3. ez-vcard - Michael Angstadt. A vCard parser library written in Java

Source: <https://code.google.com/p/ez-vcard/>

Maturity: production

Coverage All aspects of this draft.

Licensing: New BSD License

Additionally, interoperability testing of this draft is an ongoing effort under members of calconnect, the Calendaring and Scheduling Consortium. CalDAV Vendors are looking into supporting this draft.

7. Security Considerations

For security considerations specific to calendar data, see [Section 9 of \[RFC6350\]](#). Since this specification is a mapping from vCard, no new security concerns are introduced related to calendar data.

The use of JSON as a format does have security risks. [Section 7 of \[RFC4627\]](#) discusses these risks.

8. IANA Considerations

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

Type name: application

Subtype name: vcard+json

Required parameters: none

Optional parameters: version as defined for the text/vcard media type in [RFC6350].

Encoding considerations: Same as encoding considerations of application/json as specified in [RFC4627].

Security considerations: See [Section 7](#).

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

Published specification: This specification.

Applications which use this media type: Applications that currently make use of the text/vcard media type can use this as an alternative. Similarly, Applications that use the application/json media type to transfer directory data can use this to further specify the content.

Person & email address to contact for further information:
vcarddav@ietf.org

Intended usage: COMMON

Restrictions on usage: There are no restrictions on where this media type can be used.

Author: See the "Author's Address" section of this document.

Change controller: IETF

8.1. GROUP vCard Parameter

IANA has added the following entry to the vCard Parameters registry, defined in [Section 10.3.2 of \[RFC6350\]](#).

Namespace	Parameter	Reference
-----------	-----------	-----------

	GROUP	RFCTODO, Section 3.2.1.3
--	-------	--

8.2. UNKNOWN vCard Value Data Type

IANA has added the following entry to the vCard Data Types registry, defined in [Section 10.3.3 of \[RFC6350\]](#).

Value Data Type	Reference
UNKNOWN	RFCTODO, Section 5

9. Acknowledgments

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

10. References

10.1. Normative References

[I-D.ietf-jcardcal-jcal]

Kewisch, P., Daboo, C., and M. Douglass, "jCal: The JSON format for iCalendar", [draft-ietf-jcardcal-jcal-00](#) (work in progress), March 2013.

[ISO.8601.2000]

International Organization for Standardization, "Data elements and interchange formats -- Information interchange -- Representation of dates and times" ", ISO 8601, 12 2000.

[ISO.8601.2004]

International Organization for Standardization, "Data elements and interchange formats -- Information interchange -- Representation of dates and times" ", ISO 8601, 12 2004.

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", [BCP 14](#), [RFC 2119](#), March 1997.
- [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.
- [RFC5646] Phillips, A. and M. Davis, "Tags for Identifying Languages", [BCP 47](#), [RFC 5646](#), September 2009.
- [RFC6321] Daboo, C., Douglass, M., and S. Lees, "xCal: The XML Format for iCalendar", [RFC 6321](#), August 2011.
- [RFC6350] Perreault, S., "vCard Format Specification", [RFC 6350](#), August 2011.
- [RFC6351] Perreault, S., "xCard: vCard XML Representation", [RFC 6351](#), August 2011.
- [RFC6868] Daboo, C., "Parameter Value Encoding in iCalendar and vCard", [RFC 6868](#), February 2013.

10.2. Informative References

- [I-D.sheffer-running-code]
Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: the Implementation Status Section", [draft-sheffer-running-code-02](#) (work in progress), January 2013.
- [RFC4627] Crockford, D., "The application/json Media Type for JavaScript Object Notation (JSON)", [RFC 4627](#), July 2006.
- [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 vCard in JSON. ABNF Symbols not described here are taken from [RFC4627]. The schema is non-normative and given for reference only.

The numeric section numbers given in the comments refer to section in [RFC6350]. 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 [\[RFC6350\]](#).

Additional schemas may be available on the internet at [\[calconnect-artifacts\]](#).

```
; A jCard component uses the name "vcard" and a properties array.
; Restrictions to which properties and may be specified are to
; be taken from RFC6350.
```

```
jcardobject = begin-array
               DQUOTE component-name DQUOTE value-separator
               properties-array
               end-array
```

```
; A jCard 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.
```

```
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
```

```
; Each structured element may have multiple values if
```

```
; semantically allowed
```

```
structured-element = simple-prop-value / structured-multi-prop
```

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

```
; The jCard params-object is a JSON object which 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-separtor 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 = "text" / "uri" / "date" / "time" / "date-time" /
            "boolean" / "integer" / "float" / "utc-offset" /
            "language-tag" / x-type

; Property, parameter and type names MUST be lowercase. Additional
; semantic restrictions apply as described by this document and
; RFC6350.
component-name = lowercase-name
property-name = lowercase-name
param-name = lowercase-name
x-type = lowercase-name
lowercase-name = 1*(%x61-7A / DIGIT / "-")

```

Appendix B. Examples

This section contains an example of a vCard object with its jCard representation.

B.1. Example: vCard of the author of RFC6350

B.1.1. vCard Data

```

BEGIN:VCARD
VERSION:4.0
FN:Simon Perreault
N:Perreault;Simon;;;ing. jr,M.Sc.
BDAY:--0203
ANNIVERSARY:20090808T1430-0500
GENDER:M
LANG;PREF=1:fr
LANG;PREF=2:en
ORG;TYPE=work:Viagenie
ADR;TYPE=work::Suite D2-630;2875 Laurier;
Quebec;QC;G1V 2M2;Canada
TEL;VALUE=uri;TYPE="work,voice";PREF=1:tel:+1-418-656-9254;ext=102
TEL;VALUE=uri;TYPE="work,cell,voice,video,text":tel:+1-418-262-6501
EMAIL;TYPE=work:simon.perreault@viagenie.ca
GEO;TYPE=work:geo:46.772673,-71.282945

```

```
KEY;TYPE=work;VALUE=uri:
http://www.viagenie.ca/simon.perreault/simon.asc
TZ:-0500
URL;TYPE=home:http://nomis80.org
END:VCARD
```

B.1.2. jCard Data

```
[ "vcard",
  [
    [ "version", {}, "text", "4.0"],
    [ "fn", {}, "text", "Simon Perreault"],
    [ "n",
      {},
      "text",
      [ "Perreault", "Simon", "", "", [ "ing. jr", "M.Sc." ] ]
    ],
    [ "bday", {}, "date-and-or-time", "--02-03"],
    [ "anniversary",
      {},
      "date-and-or-time",
      "2009-08-08T14:30:00-05:00"
    ],
    [ "gender", {}, "text", "M"],
    [ "lang", { "pref": "1" }, "language-tag", "fr"],
    [ "lang", { "pref": "2" }, "language-tag", "en"],
    [ "org", { "type": "work" }, "text", "Viagenie"],
    [ "adr",
      { "type": "work" },
      "text",
      [
        "",
        "Suite D2-630",
        "2875 Laurier",
        "Quebec",
        "QC",
        "G1V 2M2",
        "Canada"
      ]
    ],
    [ "tel",
      { "type": [ "work", "voice" ], "pref": "1" },
      "uri",
      "tel:+1-418-656-9254;ext=102"
    ],
    [ "tel",
      { "type": [ "work", "cell", "voice", "video", "text" ] },
```

```
    "uri",
    "tel:+1-418-262-6501"
  ],
  [ "email",
    { "type": "work" },
    "text",
    "simon.perreault@viagenie.ca"
  ],
  [ "geo", { "type": "work" }, "uri", "geo:46.772673,-71.282945"],
  [ "key",
    { "type": "work" },
    "uri",
    "http://www.viagenie.ca/simon.perreault/simon.asc"
  ],
  [ "tz", {}, "utc-offset", "-05:00"],
  [ "url", { "type": "home" }, "uri", "http://nomis80.org"]
]
```

Appendix C. Change History (to be removed prior to publication as an RFC)

[draft-kewisch-vcard-in-json-01](#)

- * Added ABNF and improved references in date/time related sections
- * Changes to wording in "vCard Stream" section
- * Changes to wording about VALUE parameter when converting to vCard
- * Corrected missing "type" parameter and separator in example
- * Minor wording corrections

[draft-ietf-jcardcal-jcard-00](#)

- * Publication as a WG draft

[draft-ietf-jcardcal-jcard-01](#)

- * Changed grouping syntax to use new GROUP parameter and added respective IANA section
- * Added timestamp and date-and-or-time types instead of converting them from date/time/date-time

- * Added a further sentence on preprocessing and escaping to clarify that JSON escaping must be used
- * Described how to handle structured text values and structured text components with multiple values.
- * Corrections and additions to the ABNF Section, adaptations to example

[draft-ietf-jcardcal-jcard-02](#)

- * Made more clear that complete representation is not mandatory
- * Added sheffer-running-code section
- * Changed handling of unknown property parameter types
- * Minor corrections to sections regarding dates, fixing typos

[draft-ietf-jcardcal-jcard-03](#)

- * Add LABEL property example and description
- * Added acknowledgements
- * More typos fixed

[draft-ietf-jcardcal-jcard-04](#)

- * Added reference to [rfc6868](#)
- * Various editorial changes per jcardcal issue tracker
- * Resolved a few MAY/SHOULD conflicts
- * Put the VERSION property into its own section
- * Improved GROUP/UNKNOWN registrations by only putting vcard related information into the registration template
- * Removed vcard stream construct.
- * Added reference to [RFC6868](#) for both directions
- * Corrected some examples and ABNF

Author's Address

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

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