

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 17, 2013

P. Kewisch
Mozilla
February 13, 2013

jCard: The JSON format for vCard
draft-kewisch-vcard-in-json-00

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 August 17, 2013.

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.	vCard Stream	4
3.3.	Properties (RFC6350 section 6)	5
3.3.1.	Special Cases for Properties	7
3.3.1.1.	Multi-valued Properties	8
3.3.1.2.	Grouping of Properties	8
3.4.	Parameters (RFC6350 section 5)	8
3.4.1.	VALUE parameter	9
3.4.2.	Multi-value Parameters	9
3.5.	Values (RFC6350 section 4)	10
3.5.1.	Text (RFC6350 section 4.1)	10
3.5.2.	URI (RFC6350 section 4.2)	11
3.5.3.	Date (RFC6350 section 4.3.1)	11
3.5.4.	Time (RFC6350 section 4.3.2)	11
3.5.5.	Date-Time (RFC6350 section 4.3.3)	12
3.5.6.	Date and/or Time (RFC6350 section 4.3.4)	12
3.5.7.	Timestamp (RFC6350 section 4.3.5)	12
3.5.8.	Boolean (RFC6350 section 4.4)	13
3.5.9.	Integer (RFC6350 section 4.5)	13
3.5.10.	Float (RFC6350 section 4.6)	13
3.5.11.	UTC Offset (RFC6350 section 4.7)	13
3.5.12.	Language Tag (RFC6350 section 4.8)	14
3.6.	Extensions (RFC6350 section 6.10)	14
4.	Converting from jCard into vCard	14
5.	Handling Unrecognized Properties or Parameters	15
6.	Conversion of Date and Time Related Data Types	16
6.1.	Conversion from jCard to vCard	17
6.2.	Conversion from vCard to jCard	17
7.	Security Considerations	17
8.	IANA Considerations	17
9.	Acknowledgments	18
10.	References	19
10.1.	Normative References	19
10.2.	Informative References	20
Appendix A.	ABNF Schema	20
Appendix B.	Examples	21
B.1.	Example: vCard of the author of RFC6350	21
B.1.1.	vCard Data	22
B.1.2.	jCard Data	23
Appendix C.	Open Issues (to be removed prior to publication as an RFC)	24
Appendix D.	Change History (to be removed prior to publication as an RFC)	25

Kewisch

Expires August 17, 2013

[Page 2]

1. Introduction

The vCard data format [[RFC6350](#)] has gone through multiple revisions, most recently vCard 4. The goal followed by this format is the capture and exchange of information normally stored within an address book or directory application. As certain similarities to the iCalendar data format [[RFC5545](#)] exist it makes sense to define a JSON-based data format for vCards that is similar to the jCal format defined in [[draft-kewisch-et-al-icalendar-in-json](#)].

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 [[draft-kewisch-et-al-icalendar-in-json](#)] 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, with the exception of those which have default values.

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 data is converted to jCard using a simple mapping between the vCard data model and JSON elements.

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. When such text elements are converted into jCard the escaping MUST be removed. See [Section 3.4 of \[RFC6350\]](#).

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 complete representation, extended format of [\[ISO.8601.2004\]](#).

3.2. vCard Stream

The vCard 4 standard doesn't define a stream of vcard elements. To easier identify multiple jCard objects and provide compatibility to jCal, multiple jCard objects MAY be streamed using an array, where the first element is the string "vcardstream". Subsequent elements are vCard objects represented as described in this document.

In the typical case where there is only one vCard object, encapsulation inside a "vcardstream" array MAY be omitted.

A vCard stream can contain one or more vCard objects. Each vCard object, delimited by "BEGIN:VCARD" and "END:VCARD", is represented in JSON as a fixed length array with two elements:

1. The string "vcard"
2. An array of jCard properties

The representation of a vCard object in JSON will be named "vcard component" throughout this document.

Example:

```
[ "vcardstream",  
  [ "vcard",  
    [ /* properties */ ]  
  ],  
  [ "vcard",  
    [ /* properties */ ]  
  ],  
  ...  
]
```

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". vCard objects are delimited by the general properties "BEGIN" and "END" with the fixed value "VCARD" as defined in [Section 6.1.1](#) and 6.1.2 of [\[RFC6350\]](#). In addition, the vCard format is versioned, therefore the "version" property is mandatory. To comply with [Section 6.7.9 of \[RFC6350\]](#), the value of the version property MUST be "4.0".

3.3. 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 3.4](#).
3. The type identifier string of the value, in lowercase.

The remaining elements of the array are used for the values of the property. Properties have at least one value, so the array will have a minimum of four elements, but there can be any number of additional elements for multi-valued properties as described in [Section 3.3.1.1](#).

The array describing the property can then be inserted into the array designated for properties in the "vcard" 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 3.4](#).

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 an example, the table below shows vCard to jCard mappings for current vCard properties. Any new vCard properties added in the future will be converted in the same way.

Property	jCard Name	RFC6350 Section
BEGIN	not mapped	RFC6350 # 6.1.1
END	not mapped	RFC6350 # 6.1.2
SOURCE	source	RFC6350 # 6.1.3
KIND	kind	RFC6350 # 6.1.4
XML	xml	RFC6350 # 6.1.5
FN	fn	RFC6350 # 6.2.1
N	n	RFC6350 # 6.2.2
NICKNAME	nickname	RFC6350 # 6.2.3
PHOTO	photo	RFC6350 # 6.2.4
BDAY	bday	RFC6350 # 6.2.5
ANNIVERSARY	anniversary	RFC6350 # 6.2.6
GENDER	gender	RFC6350 # 6.2.7
ADR	adr	RFC6350 # 6.3.1
TEL	tel	RFC6350 # 6.4.1
EMAIL	email	RFC6350 # 6.4.2
IMPP	impp	RFC6350 # 6.4.3
LANG	lang	RFC6350 # 6.4.4
TZ	tz	RFC6350 # 6.5.1
GEO	geo	RFC6350 # 6.5.2
TITLE	title	RFC6350 # 6.6.1
ROLE	role	RFC6350 # 6.6.2
LOGO	logo	RFC6350 # 6.6.3
ORG	org	RFC6350 # 6.6.4
MEMBER	member	RFC6350 # 6.6.5
RELATED	related	RFC6350 # 6.6.6
CATEGORIES	categories	RFC6350 # 6.7.1
NOTE	note	RFC6350 # 6.7.2
PROPID	propid	RFC6350 # 6.7.3
REV	rev	RFC6350 # 6.7.4
SOUND	sound	RFC6350 # 6.7.5
UID	uid	RFC6350 # 6.7.6
CLIENTPIDMAP	clientpidmap	RFC6350 # 6.7.7
URL	url	RFC6350 # 6.7.8
VERSION	version	RFC6350 # 6.7.9
KEY	key	RFC6350 # 6.8.1
FBURL	fburl	RFC6350 # 6.9.1
CALADRURI	caladruri	RFC6350 # 6.9.2
CALURI	caluri	RFC6350 # 6.9.3

3.3.1. Special Cases for Properties

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

3.3.1.1. Multi-valued Properties

The following vCard properties can have values that consist of a list of "standard" vCard values separated by a specific delimiter. In jCard these properties are added as further members of the array describing the property.

Property	jCard Name	RFC6350 Section
NICKNAME	nickname	[RFC6350] # 6.2.3
CATEGORIES	categories	[RFC6350] # 6.7.1

Example:

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

3.3.1.2. Grouping of Properties

[\[RFC6350\]](#) [Section 3.3](#) defines a grouping construct that is used to group related properties together. In jCard, each grouped property appears as a separate property containing the group name and property name, separated by a "." character. This was done to maintain compatibility of array elements with [\[draft-kewisch-et-al-icalendar-in-json\]](#).

Example:

```
[ "vcard",
  [
    ["groupname.email", {}, "text", "johndoe@example.org"],
    ...
  ],
  ...
]
```

3.4. 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.

Example:

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

As an example, the table below shows vCard to jCard mappings for current vCard parameters. Any new vCard parameters added in the future will be converted in the same way.

Parameter	jCard Name	RFC6350 Section
LANGUAGE	language	RFC6350 # 5.1
VALUE	not mapped	RFC6350 # 5.2
PREF	pref	RFC6350 # 5.3
ALTID	altid	RFC6350 # 5.4
PID	pid	RFC6350 # 5.5
TYPE	type	RFC6350 # 5.6
MEDIATYPE	mediatype	RFC6350 # 5.7
CALSCALE	calscale	RFC6350 # 5.8
SORT-AS	sort-as	RFC6350 # 5.9
GEO	geo	RFC6350 # 5.10
TZ	tz	RFC6350 # 5.11

3.4.1. 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 the default value type for that property.

3.4.2. 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, but an array with one element MAY also be used.

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" ]
    ...
  ],
  ...
]
```

As an example, the table below shows a list of vCard parameters where COMMA-separated values are used. Any new vCard parameters added in the future will be handled in the same way.

Parameter	RFC6350 Section
PID	[RFC6350] # 5.5
TYPE	[RFC6350] # 5.6
SORT-AS	[RFC6350] # 5.9

3.5. 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.

3.5.1. 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.

Example:

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

3.5.2. 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"],  
...
```

3.5.3. 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 formatted using the [\[ISO.8601.2004\]](#) complete representation, extended format. The textual format specifies a four-digit year, two-digit month, and two-digit day of the month, separated by the "-" character.

Example:

```
...  
["bday", {}, "date", "1970-01-01"],  
...
```

3.5.4. 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 formatted using the [\[ISO.8601.2004\]](#) complete representation, extended format. The textual format specifies a two-digit, 24-hour of the day (i.e., values 00-23), two-digit minute in the hour (i.e., values 00-59), and two-digit seconds in the minute (i.e., values 00-60). The seconds value of 60 MUST only be used to

account for positive "leap" seconds. Fractions of a second are not supported by this format. Contrary to [\[draft-kewisch-et-al-icalendar-in-json\]](#), UTC offsets are permitted within a time value.

Example:

```
...
["x-time-local", {}, "time", "12:30:00"],
["x-time-utc", {}, "time", "12:30:00Z"],
["x-time-offset", "time", "12:30:00-0800"],
...
```

3.5.5. 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 formatted using the [\[ISO.8601.2004\]](#) complete representation, extended format. The same restrictions with respect to leap seconds, fractions of a second and UTC offsets as in [Section 3.5.4](#) apply.

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+0100"],
...
```

3.5.6. Date and/or Time ([RFC6350 section 4.3.4](#))

Description: jCard has no direct equivalent to vCard's "DATE-AND-OR-TIME" property value data type. Instead, the specified date, date-time or time format MUST be detected and the property MUST be represented by a property with the detected type identifier. See [Section 6](#) for more information on how conversion between jCard and vCard should be handled.

3.5.7. Timestamp ([RFC6350 section 4.3.5](#))

Description: jCard has no direct equivalent to vCard's "TIMESTAMP" property value data type. Instead, the specified complete date and time of day combination MUST be represented by a property with the "date-time" identifier. See [Section 6](#) for more information on how conversion between jCard and vCard should be handled.

3.5.8. 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],  
...
```

3.5.9. 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],  
...
```

3.5.10. 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],  
...
```

3.5.11. 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"],
..
```

3.5.12. 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"],
..
```

3.6. 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](#)].

Backslash escaping and line folding MUST be applied to the resulting vCard data as required by [[RFC6350](#)].

jCard properties that do not specify the default type for the vCard property MUST add a VALUE parameter when converting to vCard. jCard properties that specify the default type SHOULD NOT add a VALUE

parameter.

5. Handling Unrecognized Properties or Parameters

In vCard, properties have a default value type specified by their definition, e.g. "BDAY"'s value type is "date-or-date-time", but it can also be reset to a single "text" value. When a property uses its default value type, the "VALUE" property parameter does not need to be specified on the property.

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. As a result, the following rules are applied when dealing with unrecognized properties and property parameters:

- o When 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 string object. The content of that string is the unprocessed value text.
 - * 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.
- o When 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 SHOULD be omitted.

Example: 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", {}, "text", "mailto:abuse@example.org"],  
...
```

Example: 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

Example: 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. Conversion of Date and Time Related Data Types

[RFC6350] defines the data types DATE, DATE-TIME, TIME, DATE-AND-OR-TIME and TIMESTAMP, covering various aspects of dates and times. As jCard is more "strongly typed", some of these types have been consolidated. This section aims to aid conversion between jCard and vCard and vice versa.

Regardless of the date/time related property converted, jCard and vCard use different representations of the [ISO.8601.2004] format. The date format MUST be adjusted during conversion.

6.1. Conversion from jCard to vCard

- o If the property type of the jCard property matches the property's default type, the VALUE parameter MUST NOT be added to the vCard property.
- o For property types "date-and-or-time" or "timestamp", the property value MUST be converted to a permitted date and/or time format as specified in [\[RFC6350\] Section 4.3.4](#) and 4.3.5 and the VALUE parameter MUST NOT be added to the vCard property. If a permitted conversion cannot be done, the VALUE parameter must be added with the closest available date/time format identifier.
- o If the default type is unknown or the jCal property type does not match the default type, the VALUE parameter MUST be specified.

6.2. Conversion from vCard to jCard

- o If the parser knows the default type of the property and it is one of "DATE", "DATE-TIME" or "TIME", the properties can be directly converted following the guidelines of the respective format type in this document.
- o For the property type "date-and-or-time", the parser SHOULD detect if it is handling either a date, a time, or a date-time and use the respective format identifier in jCard.
- o For the property type "timestamp", the parser MUST use the format type identifier "date-time" in the jCard property.
- o If the (default) property type is unknown, the property value MUST be treated as opaque text and the "text" format type identifier MUST be used.

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

[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. This specification originated from the work of the XML-JSON technical committee of the Calendaring and Scheduling Consortium.

[10.](#) References

10.1. Normative References

- [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.
- [[draft-kewisch-et-al-icalendar-in-json](#)] Kewisch, P., Daboo, C., and M. Douglass, "jCal: The JSON format for iCalendar", draft [draft-kewisch-et-al-icalendar-in-json-02](#).

[10.2.](#) Informative References

[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 vCard Stream is an array with the first element being the
; string "vcardstream". All remaining elements are jcardobjects.
```

```
jcardstream = begin-array
               DQUOTE "vcardstream" DQUOTE
               *(value-separator jcardobject)
               end-array
```

```
jcardobject = component
```

```
; A jCard object consists of the name string "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 = string / number / boolean

; 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

; 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.1.1.1. 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", "-02-03" ],
    [ "anniversary", {}, "date-time", "2009-08-08T14:30:00-0500" ],
    [ "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", {}, "uri", "http://nomis80.org" ]
  ]
]
```


Appendix C. Open Issues (to be removed prior to publication as an RFC)

- o [\[RFC6350\]](#) doesn't define any kind of stream for multiple vcard elements. For similarity with the jCal draft, I have added a "vcardstream". Is this wanted and does the name fit? (Maybe rather "jcardstream" ?)
- o During conversion to jCard, I have dropped "date-and-or-time" and "timestamp" formats, since they can equally be represented by the formats "date", "time" and "date-time". I have added guidelines to [Section 6](#), but there may be some situations I am unaware of that make dropping these formats impossible.
- o [\[draft-kewisch-et-al-icalendar-in-json\]](#) and [\[RFC5545\]](#) define the concept of sub-components. vCard on the other hand merely mentions the "BEGIN" and "END" properties, which have only one possible value "VCARD". Should any preparations be done, in case an extension defines other possible values for the "BEGIN" and "END" properties, or should an extension to this document be defined when a such vCard extension emerges?
- o I am not clear on support for using 60 as a value for seconds (i.e. leap seconds) in vCard. This is defined quite clearly in [\[RFC5545\]](#). Please double check this.
- o This document needs some mention of the truncated date format. As I don't have access to the original ISO 8601 documents, I would appreciate some advice on how to best handle this.
- o This first draft doesn't yet incorporate all proposed changes discussed on the vcarddav and calsify mailing lists after publishing of [draft-kewisch-et-al-icalendar-in-json-01](#). Most notably, conversion of structured values into arrays or objects. This will be done in an upcoming draft and when an agreement is reached.
- o This document defines in [Section 3.3.1.2](#), that grouped properties are not to be handled differently than normal properties. This was done for compatibility with the jCal draft to make it easier for a parser to support both jCal and jCard. Another option would be to add a pseudo-property with the special type "group", the property name being the group name and the property value being an array of properties. Downside is that there is room for an extra set of parameters in the group container that does not serialize back to vCard.
- o There is some open discussion on the vcarddav and calsify lists about structured property and parameter values. The origin of

this format comes from jCal, where structured values are very rare and processing the remaining ones requires a processing library anyway. Therefore neither structured property values nor parameter values were added. For jCard on the other hand, structured values appear quite often. I have come up with three possible options for structured property values:

1. Use an array as the value, possibly with a null value for any empty structured property: foo;;bar becomes ["foo", null, "bar"]
2. Use an object for key-value combinations. Downside is that the key may not always be known, and needs to be defined at least for standard properties. (i.e for N: "surnames", "given", "additional", "honor-prefix", "honor-suffix"). Object values themselves could also be array values, i.e multiple honorific prefixes. Upside is that it makes it very easy to access the values even without using a dedicated client library.
3. Keep the property value as an opaque text and leave any splitting to client libraries. This makes it very simple to parse both iCalendar and vCard data into a similar format, but puts the burden of parsing the structured value on the client or client library.

Also, there should be a mention of how multi-values inside a structured value should be handled. With the second approach, these could themselves be an array, similar to how this document currently defines multi-value parameters.

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

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/>

