JMAP
Internet-Draft
Intended status: Standards Track
Expires: 17 July 2022

R. Stepanek
FastMail
M. Loffredo
IIT-CNR

13 January 2022

JSContact: A JSON representation of contact data
draft-ietf-jmap-jscontact-10

Abstract

This specification defines a data model and JSON representation of contact card information that can be used for data storage and exchange in address book or directory applications. It aims to be an alternative to the vCard data format and to be unambiguous, extendable and simple to process. In contrast to the JSON-based jCard format, it is not a direct mapping from the vCard data model and expands semantics where appropriate.

Status of This Memo

Copyright Notice

Table of Contents

```
   1.  Introduction . . . . . . . . . . . . . . . . . . . . . . . .   3
     1.1.  Relation to the xCard and jCard formats . . . . . . . . .   4
     1.2.  Terminology . . . . . . . . . . . . . . . . . . . . . . .   4
     1.3.  Vendor-specific Property Extensions and Values  . . . . .   4
     1.4.  Type Signatures . . . . . . . . . . . . . . . . . . . . .   5
     1.5.  Data types  . . . . . . . . . . . . . . . . . . . . . . .   5
       1.5.1.  Context . . . . . . . . . . . . . . . . . . . . . . .   5
       1.5.2.  Id  . . . . . . . . . . . . . . . . . . . . . . . . .   6
       1.5.3.  PatchObject . . . . . . . . . . . . . . . . . . . . .   6
       1.5.4.  Preference  . . . . . . . . . . . . . . . . . . . . .   7
       1.5.5.  UnsignedInt . . . . . . . . . . . . . . . . . . . . .   7
       1.5.6.  UTCDateTime . . . . . . . . . . . . . . . . . . . . .   8
   2.  Card  . . . . . . . . . . . . . . . . . . . . . . . . . . . .   8
     2.1.  Metadata properties . . . . . . . . . . . . . . . . . . .   8
       2.1.1.  @type . . . . . . . . . . . . . . . . . . . . . . . .   8
       2.1.2.  uid . . . . . . . . . . . . . . . . . . . . . . . . .   8
       2.1.3.  prodId  . . . . . . . . . . . . . . . . . . . . . . .   8
       2.1.4.  created . . . . . . . . . . . . . . . . . . . . . . .   8
       2.1.5.  updated . . . . . . . . . . . . . . . . . . . . . . .   9
       2.1.6.  kind  . . . . . . . . . . . . . . . . . . . . . . . .   9
       2.1.7.  relatedTo . . . . . . . . . . . . . . . . . . . . . .   9
       2.1.8.  language  . . . . . . . . . . . . . . . . . . . . . .  10
     2.2.  Name and Organization properties  . . . . . . . . . . . .  10
       2.2.1.  name  . . . . . . . . . . . . . . . . . . . . . . . .  10
       2.2.2.  fullName  . . . . . . . . . . . . . . . . . . . . . .  11
       2.2.3.  nickNames . . . . . . . . . . . . . . . . . . . . . .  11
       2.2.4.  organizations . . . . . . . . . . . . . . . . . . . .  12
       2.2.5.  titles  . . . . . . . . . . . . . . . . . . . . . . .  12
       2.2.6.  speakToAs . . . . . . . . . . . . . . . . . . . . . .  12
     2.3.  Contact and Resource properties . . . . . . . . . . . . .  13
       2.3.1.  emails  . . . . . . . . . . . . . . . . . . . . . . .  13
       2.3.2.  phones  . . . . . . . . . . . . . . . . . . . . . . .  14
       2.3.3.  online  . . . . . . . . . . . . . . . . . . . . . . .  15
       2.3.4.  photos  . . . . . . . . . . . . . . . . . . . . . . .  15
       2.3.5.  preferredContactMethod  . . . . . . . . . . . . . . .  16
       2.3.6.  preferredContactLanguages . . . . . . . . . . . . . .  16
     2.4.  Address and Location properties . . . . . . . . . . . . .  17
```

2.4.1. addresses 17
2.5. Multilingual properties 19
2.5.1. localizations 19
2.6. Additional properties 19
2.6.1. anniversaries 19
2.6.2. personalInfo 20
2.6.3. notes 21
2.6.4. categories 21
2.6.5. timeZones 21
3. CardGroup 21
3.1. Group properties 21
3.1.1. @type 21
3.1.2. uid 21
3.1.3. members 22
3.1.4. name 22
3.1.5. card 22
4. Implementation Status 22
4.1. IIT-CNR/Registro.it 22
5. IANA Considerations 23
6. Security Considerations 23
7. References 23
7.1. Normative References 23
7.2. Informative References 24
Authors' Addresses 25

1. Introduction

This document defines a data model for contact card data normally used in address book or directory applications and services. It aims to be an alternative to the vCard data format [RFC6350] and to provide a JSON-based standard representation of contact card data.

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

* Most of the initial set of attributes should be taken from the vCard data format [RFC6350] and extensions ([RFC6473], [RFC6474], [RFC6715], [RFC6869], [RFC8605]). The specification should add new attributes or value types, or not support existing ones, where appropriate. Conversion between the data formats need not fully preserve semantic meaning.

* The attributes of the cards data represented must be described as a simple key-value pair, reducing complexity of its representation.

* The data model should avoid all ambiguities and make it difficult to make mistakes during implementation.

* Extensions, such as new properties and components, MUST NOT lead to requiring an update to this document.

The representation of this data model is defined in the I-JSON format [RFC7493], which is a strict subset of the JavaScript Object Notation (JSON) Data Interchange Format [RFC8259]. Using JSON is mostly a pragmatic choice: its widespread use makes Card easier to adopt, and the availability of production-ready JSON implementations eliminates a whole category of parser-related interoperability issues.

1.1. Relation to the xCard and jCard formats

The xCard [RFC6351] and jCard [RFC7095] specifications define alternative representations for vCard data, in XML and JSON format respectively. Both explicitly aim to not change the underlying data model. Accordingly, they are regarded as equal to vCard in the context of this document.

1.2. Terminology

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

1.3. Vendor-specific Property Extensions and Values

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

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

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

1.4. Type Signatures

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

* * - The type is undefined (the value could be any type, although permitted values may be constrained by the context of this value).

* String - The JSON string type.

* Number - The JSON number type.

* Boolean - The JSON boolean type.

* A[B] - A JSON object where the keys are all of type A, and the values are all of type B.

* A[] - An array of values of type A.

* A|B - The value is either of type A or of type B.

1.5. Data types

In addition to the standard JSON data types, a couple of additional data types are common to the definitions of JSContact objects and properties.

1.5.1. Context

Contact information typically is associated with a context in which it should be used. For example, someone might have distinct phone numbers for work and private contexts. The Context data type enumerates common contexts.

Common context values are:

* private: The contact information may be used to contact the card holder in a private context.

* work: The contact information may be used to contact the card holder in a professional context.

Additional allowed values may be defined in the properties or data types that make use of the Context data type, registered in a future RFC, or a vendor-specific value.

1.5.2. Id

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

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

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

1.5.3. PatchObject

A PatchObject is of type String[*], and represents an unordered set of patches on a JSON object. Each key is a path represented in a subset of JSON pointer format [RFC6901]. The paths have an implicit leading /, so each key is prefixed with / before applying the JSON pointer evaluation algorithm.

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

1. The pointer MUST NOT reference inside an array (i.e., you MUST NOT insert/delete from an array; the array MUST be replaced in its entirety instead).

2. All parts prior to the last (i.e., the value after the final slash) MUST already exist on the object being patched.

3. There MUST NOT be two patches in the PatchObject where the pointer of one is the prefix of the pointer of the other, e.g., addresses/1/city and addresses.

4. The value for the patch MUST be valid for the property being set (of the correct type and obeying any other applicable restrictions), or if null the property MUST be optional.

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

* If null, remove the property from the patched object. If the key is not present in the parent, this a no-op.

* If non-null, set the value given as the value for this property (this may be a replacement or addition to the object being patched).

A PatchObject does not define its own @type property. Instead, a @type property in a patch MUST be handled as any other patched property value.

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

1.5.4. Preference

This data type allows to define a preference order on same-typed contact information. For example, a card holder may have two email addresses and prefer to be contacted with one of them.

A preference value MUST be an integer number in the range 1 and 100. Lower values correspond to a higher level of preference, with 1 being most preferred. If no preference is set, then the contact information MUST be interpreted as being least preferred.

Note that the preference only is defined in relation to contact information of the same type. For example, the preference orders within emails and phone numbers are indendepent of each other. Also note that the _preferredContactMethod_ property allows to define a preferred contact method across method types.

1.5.5. UnsignedInt

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

1.5.6. UTCDateTime

This is a string in [RFC3339] date-time format, with the further restrictions that any letters MUST be in uppercase, and the time offset MUST be the character Z. Fractional second values MUST NOT be included unless non-zero and MUST NOT have trailing zeros, to ensure there is only a single representation for each date-time.

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

2. Card

MIME type: application/jscontact+json;type=card

A Card object stores information about a person, organization or company.

2.1. Metadata properties

2.1.1. @type

Type: String (mandatory).

Specifies the type of this object. This MUST be Card.

2.1.2. uid

Type: String (mandatory).

An identifier, used to associate the object as the same across different systems, addressbooks and views. [RFC4122] describes a range of established algorithms to generate universally unique identifiers (UUID), and the random or pseudo-random version is recommended. For compatibility with [RFC6350] UIDs, implementations MUST accept both URI and free-form text.

2.1.3. prodId

Type: String (optional).

The identifier for the product that created the Card object.

2.1.4. created

Type: UTCDateTime (optional).

The date and time when this Card object was created.

2.1.5. updated

Type: UTCDateTime (optional).

The date and time when the data in this Card object was last modified.

2.1.6. kind

Type: String (optional). The kind of the entity the Card represents.

The value MUST be either one of the following values, registered in a future RFC, or a vendor-specific value:

* individual: a single person

* org: an organization

* location: a named location

* device: a device, such as appliances, computers, or network elements

* application: a software application

2.1.7. relatedTo

Type: String[Relation] (optional).

Relates the object to other Card and CardGroup objects. This is represented as a map, where each key is the uid of the related Card or CardGroup and the value defines the relation. The Relation object has the following properties:

* @type: String (mandatory). Specifies the type of this object. This MUST be Relation.

* relation: String[Boolean] (optional, default: empty Object) Describes how the linked object is related to the linking object. The relation is defined as a set of relation types. If empty, the relationship between the two objects is unspecified. Keys in the set MUST be one of the RELATED property [RFC6350] type parameter values, or an IANA-registered value, or a vendor-specific value. The value for each key in the set MUST be true.

2.1.8. language

Type: String (optional).

This defines the locale in which free-text property values can be assumed to be written in. The value MUST be a language tag as defined in [RFC5646]. Note that such values MAY be localized in the localizations Section 2.5.1 property.

2.2. Name and Organization properties

2.2.1. name

Type: Name (optional).

The name of the entity represented by this Card.

A Name object has the following properties

* @type: Name (mandatory). Specifies the type of this object. This MUST be Name.

* components: NameComponent[] (mandatory). The components making up the name. The component list MUST have at least one entry. Name components SHOULD be ordered such that their values joined by whitespace produce a valid full name of this entity. Doing so, implementations MAY ignore any components of type separator.

* locale: String (optional). The locale of the name. The value MUST be a language tag as defined [RFC5646].

A NameComponent object has the following properties:

* @type: String (mandatory). Specifies the type of this object. This MUST be NameComponent.

* value: String (mandatory). The value of this name component.

* type: String (mandatory). The type of this name component. The value MUST be either one of the following values, registered in a future RFC, or a vendor-specific value:

  - prefix. The value is a honorific title(s), e.g. "Mr", "Ms", "Dr".

  - given. The value is a given name, also known as "first name", "personal name".

- surname. The value is a surname, also known as "last name", "family name".

- middle. The value is a middle name, also known as "additional name".

- suffix. The value is a honorific suffix, e.g. "B.A.", "Esq.".

- separator. A formatting separator for two name components. The value property of the component includes the verbatim separator, for example a newline character.

* nth: UnsignedInt (optional, default: 1). Defines the rank of this name component to other name components of the same type. If set, the property value MUST be higher than or equal to 1.

  For example, two name components of type surname may have their nth property value set to 1 and 2, respectively. In this case, the first name component defines the surname, and the second name component the secondary surname.

  Note that this property value does not indicate the order in which to print name components of the same type. Some cultures print the secondary surname before the first surname, others the first before the second. Implementations SHOULD inspect the locale property of the Name object to determine the appropriate formatting. They MAY print name components in order of appearance in the components property of the Name object.

2.2.2. fullName

Type: String (optional).

The full name (e.g. the personal name and surname of an individual, the name of an organization) of the entity represented by this card. The purpose of this property is to define a name, even if the individual name components are not known. In addition, it is meant to provide alternative versions of the name for internationalisation. Implementations SHOULD prefer using the _name_ property over this one and SHOULD NOT store the concatenated name component values in this property.

2.2.3. nickNames

Type: String[] (optional).

The nick names of the entity represented by this card.

2.2.4. organizations

Type: Id[Organization] (optional).

The companies or organization names and units associated with this card. An Organization object has the following properties:

* @type: String (mandatory). Specifies the type of this object. This MUST be Organization.

* name: String (mandatory). The name of this organization.

* units: String[] (optional). Additional levels of organizational unit names.

2.2.5. titles

Type : Id[Title] (optional).

The job titles or functional positions of the entity represented by this card. A Title has object the following properties:

* @type: String (mandatory). Specifies the type of this object. This MUST be Title.

* title: String (mandatory). The title of the entity represented by this card.

* organization: Id (optional). The id of the organization in which this title is held.

2.2.6. speakToAs

Type: SpeakToAs (optional).

Provides information how to address, speak to or refer to the entity that is represented by this card. A SpeakToAs object has the following properties, of which at least one property other than @type MUST be set:

* @type: String (mandatory). Specifies the type of this object. This MUST be SpeakToAs.

* grammaticalGender: String (optional). Defines which grammatical gender to use in salutations and other grammatical constructs. Allowed values are:

  - animate

  - female

  - inanimate

  - male

  - neuter

  Note that the grammatical gender does not allow to infer the gender identities or biological sex of the contact.

* pronouns: String (optional). Defines the gender pronouns that the contact chooses to use for themselves. Any value or form is allowed. Examples in English include she/her and they/them/theirs.

The property values SHOULD be localized in the language defined in the language property. They MAY be overridden in the localizations property (Section 2.5.1).

2.3. Contact and Resource properties

2.3.1. emails

Type: Id[EmailAddress] (optional).

The email addresses to contact the entity represented by this card. An EmailAddress object has the following properties:

* @type: String (mandatory). Specifies the type of this object. This MUST be EmailAddress.

* email: String (mandatory). The email address. This MUST be an _addr-spec_ value as defined in Section 3.4.1 of [RFC5322].

* contexts: Context[Boolean] (optional) The contexts in which to use this email address. The value for each key in the object MUST be true.

* pref: Preference (optional) The preference of this email address in relation to other email addresses.

* label: String (optional). A label describing the value in more detail.

2.3.2. phones

Type: Id[Phone] (optional).

The phone numbers to contact the entity represented by this card. A Phone object has the following properties:

* @type: String (mandatory). Specifies the type of this object. This MUST be Phone.

* phone: String (mandatory). The phone value, as either a URI or a free-text phone number. Typical URI schemes are the [RFC3966] tel or [RFC3261] sip schemes, but any URI scheme is allowed.

* features: String[Boolean] (optional). The set of contact features that this phone number may be used for. The set is represented as an object, with each key being a method type. The value for each key in the object MUST be true. The method type MUST be either one of the following values, registered in a future RFC, or a vendor-specific value:

    - voice The number is for calling by voice.

    - fax The number is for sending faxes.

    - pager The number is for a pager or beeper.

    - text The number supports text messages (SMS).

    - cell The number is for a cell phone.

    - textphone The number is for a device for people with hearing or speech difficulties.

    - video The number supports video conferencing.

* contexts: Context[Boolean] (optional) The contexts in which to use this number. The value for each key in the object MUST be true.

* pref: Preference (optional) The preference of this number in relation to other numbers.

* label: String (optional). A label describing the value in more detail.

2.3.3. online

Type: Id[Resource] (optional).

The online resources and services that are associated with the entity represented by this card. A Resource object has the following properties:

* @type: String (mandatory). Specifies the type of this object. This MUST be Resource.

* resource: String (mandatory). The resource value, where the allowed value form is defined by the the _type_ property. In any case the value MUST NOT be empty.

* type: String (optional). The type of the resource value. Allowed values are:

  - uri The resource value is a URI, e.g. a website link. This MUST be a valid _URI_ as defined in Section 3 of [RFC3986] and updates.

  - username The resource value is a username associated with the entity represented by this card (e.g. for social media, or an IM client). The _label_ property SHOULD be included to identify what service this is for. For compatibility between clients, this label SHOULD be the canonical service name, including capitalisation. e.g. Twitter, Facebook, Skype, GitHub, XMPP. The resource value may be any non-empty free text.

* mediaType: String (optional). Used for URI resource values. Provides the media type [RFC2046] of the resource identified by the URI.

* contexts: Context[Boolean] (optional) The contexts in which to use this resource. The value for each key in the object MUST be true.

* pref: Preference (optional) The preference of this resource in relation to other resources.

* label: String (optional). A label describing the value in more detail.

2.3.4. photos

Type: Id[File] (optional).

A map of photo ids to File objects that contain photographs or images associated with this card. A typical use case is to include an avatar for display along the contact name.

A File object has the following properties:

* @type: String (mandatory). Specifies the type of this object. This MUST be File.

* href: String (mandatory). A URI where to fetch the data of this file.

* mediaType: String (optional). The content-type of the file, if known.

* size: UnsignedInt (optional). The size, in octets, of the file when fully decoded (i.e., the number of octets in the file the user would download), if known.

* pref: Preference (optional) The preference of this photo in relation to other photos.

* label: String (optional). A label describing the value in more detail.

2.3.5. preferredContactMethod

Type : String (optional)

Defines the preferred method to contact the holder of this card. The value MUST be the property names: emails, phones, online.

2.3.6. preferredContactLanguages

Type : String[ContactLanguage[]] (optional)

Defines the preferred languages for contacting the entity associated with this card. The keys in the object MUST be [RFC5646] language tags. The values are a (possibly empty) list of contact language preferences for this language. A valid ContactLanguage object MUST have at least one of its properties set.

A ContactLanguage object has the following properties:

* @type: String (mandatory). Specifies the type of this object. This MUST be ContactLanguage.

* context: Context (optional). Defines the context in which to use this language.

* pref: Preference (optional). Defines the preference of this language in relation to other languages of the same context.

Also see the definition of the VCARD LANG property (Section 6.4.4., [RFC6350]).

2.4. Address and Location properties

2.4.1. addresses

Type: Id[Address] (optional).

A map of address ids to Address objects, containing physical locations. An Address object has the following properties:

* @type: String (mandatory). Specifies the type of this object. This MUST be Address.

* fullAddress: String (optional). The complete address, excluding type and label. This property is mainly useful to represent addresses of which the individual address components are unknown, or to provide localized representations.

* street: StreetComponent[] (optional). The street address. The concatenation of the component values, separated by whitespace, SHOULD result in a valid street address for the address locale. Doing so, implementations MAY ignore any separator components. The StreetComponent object type is defined in the paragraph below.

* locality: String (optional). The city, town, village, post town, or other locality within which the street address may be found.

* region: String (optional). The province, such as a state, county, or canton within which the locality may be found.

* country: String (optional). The country name.

* postcode: String (optional). The postal code, post code, ZIP code or other short code associated with the address by the relevant country's postal system.

* countryCode: String (optional). The ISO-3166-1 country code.

* coordinates: String (optional) A [RFC5870] "geo:" URI for the address.

* timeZone: String (optional) Identifies the time zone this address is located in. This either MUST be a time zone name registered in the IANA Time Zone Database (https://www.iana.org/time-zones), or it MUST be a valid TimeZoneId as defined in [RFC8984]. For the latter, a corresponding time zone MUST be defined in the timeZones property.

* contexts: Context[Boolean] (optional). The contexts of the address information. In addition to the common contexts, allowed values are:

  - billing An address to be used for billing.

  - postal An address to be used for delivering physical items. The value for each key in the object MUST be true.

* pref: Preference (optional) The preference of this address in relation to other addresses.

* label: String (optional). A label describing the value in more detail.

A StreetComponent object has the following properties:

* @type: String (mandatory). Specifies the type of this object. This MUST be StreetComponent.

* type: String (mandatory). The type of this street component. The value MUST be either one of the following values, registered in a future RFC, or a vendor-specific value:

  - name. The street name.

  - number. The street number.

  - apartment. The apartment number or identifier.

  - room. The room number or identifier.

  - extension. The extension designation or box number.

  - direction. The cardinal direction, e.g. "North".

  - building. The building or building part this address is located in.

  - floor. The floor this address is located on.

  - postOfficeBox. The post office box number or identifier.

  - separator. A separator for two street components. The value property of the component includes the verbatim separator, for example a newline character.

  - unknown. A name component value for which no type is known.

* value: String (mandatory). The value of this street component.

2.5. Multilingual properties

2.5.1. localizations

Type: String[PatchObject] (optional).

A map of language tags [RFC5646] to patches, which localize a property value into the locale of the respective language tag. The paths in the PatchObject keys are relative to the Card object that includes the localizations property. A patch MUST NOT target the localizations property.

The following example shows a Card object, where one of its addresses Tokyo is localized for the jp locale.

```
"@type": "Card",
...
"addresses": {
  "addr1": {
    "@type": "Address",
    "locality": "Tokyo",
   }
 },
 "localizations": {
   "jp": {
      "addresses/addr1/locality":""
   }
 }
```

Figure 1

2.6. Additional properties

2.6.1. anniversaries

Type : Id[Anniversary] (optional).

These are memorable dates and events for the entity represented by this card. An Anniversary object has the following properties:

* @type: String (mandatory). Specifies the type of this object. This MUST be Anniversary.

* type: String (optional). Specifies the type of the anniversary. This RFC predefines the following types, but implementations MAY use additional values:

  - birth: a birth day anniversary

  - death: a death day anniversary

* date: String (mandatory). The date of this anniversary, in the form "YYYY-MM-DD" (any part may be all 0s for unknown) or a [RFC3339] timestamp.

* place: Address (optional). An address associated with this anniversary, e.g. the place of birth or death.

* label: String (optional). A label describing the value in more detail.

2.6.2. personalInfo

Type: Id[PersonalInformation] (optional).

Defines personal information about the entity represented by this card. A PersonalInformation object has the following properties:

* @type: String (mandatory). Specifies the type of this object. This MUST be PersonalInformation.

* type: String (mandatory). Specifies the type for this personal information. Allowed values are:

  - expertise: a field of expertise or credential

  - hobby: a hobby

  - interest: an interest

* value: String (mandatory). The actual information. This generally is free-text, but future specifications MAY restrict allowed values depending on the type of this PersonalInformation.

* level: String (optional) Indicates the level of expertise, or engagement in hobby or interest. Allowed values are: high, medium and low.

* label: String (optional). A label describing the value in more detail.

2.6.3. notes

Type: String (optional).

Arbitrary notes about the entity represented by this card.

2.6.4. categories

Type: String[Boolean] (optional). The set of free-text or URI categories that relate to the card. The set is represented as an object, with each key being a category. The value for each key in the object MUST be true.

2.6.5. timeZones

Type: String[TimeZone] (optional). Maps identifiers of custom time zones to their time zone definitions. For a description of this property see the timeZones property definition in [RFC8984].

3. CardGroup

MIME type: application/jscontact+json;type=cardgroup

A CardGroup object represents a group of cards. Its members may be Cards or CardGroups.

3.1. Group properties

3.1.1. @type

Type: String (mandatory).

Specifies the type of this object. This MUST be CardGroup.

3.1.2. uid

Type: String (mandatory). The uid of this group. Both CardGroup and Card share the same namespace for the uid property.

3.1.3. members

Type: String[Boolean] (mandatory). The members of this group.

The set is represented as an object, with each key being the uid of another Card or CardGroup. The value for each key in the object MUST be true.

3.1.4. name

Type: String (optional). The user-visible name for the group, e.g. "Friends". This may be any UTF-8 string of at least 1 character in length and maximum 255 octets in size. The same name may be used by two different groups.

3.1.5. card

Type: Card (optional). The card that represents this group.

4. Implementation Status

NOTE: Please remove this section and the reference to [RFC7942] prior to publication as an RFC. This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [RFC7942]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist. According to [RFC7942], "this will allow reviewers and working groups to assign due consideration to documents that have the benefit of running code, which may serve as evidence of valuable experimentation and feedback that have made the implemented protocols more mature. It is up to the individual working groups to use this information as they see fit".

4.1. IIT-CNR/Registro.it

* Responsible Organization: Institute of Informatics and Telematics of National Research Council (IIT-CNR)/Registro.it

* Location: https://rdap.pubtest.nic.it/ (https://rdap.pubtest.nic.it/)

* Description: This implementation includes support for RDAP queries using data from the public test environment of .it ccTLD. The RDAP server returns responses including Card in place of jCard when queries contain the parameter jscard=1.

* Level of Maturity: This is an "alpha" test implementation.

* Coverage: This implementation includes all of the features described in this specification.

* Contact Information: Mario Loffredo, mario.loffredo@iit.cnr.it

5. IANA Considerations

TBD

6. Security Considerations

TBD

7. References

7.1. Normative References

[RFC2046] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, DOI 10.17487/RFC2046, November 1996, <https://www.rfc-editor.org/info/rfc2046>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, DOI 10.17487/RFC4122, July 2005, <https://www.rfc-editor.org/info/rfc4122>.

[RFC5646] Phillips, A., Ed. and M. Davis, Ed., "Tags for Identifying Languages", BCP 47, RFC 5646, DOI 10.17487/RFC5646, September 2009, <https://www.rfc-editor.org/info/rfc5646>.

[RFC5870] Mayrhofer, A. and C. Spanring, "A Uniform Resource Identifier for Geographic Locations ('geo' URI)", RFC 5870, DOI 10.17487/RFC5870, June 2010, <https://www.rfc-editor.org/info/rfc5870>.

[RFC6350] Perreault, S., "vCard Format Specification", RFC 6350, DOI 10.17487/RFC6350, August 2011, <https://www.rfc-editor.org/info/rfc6350>.

[RFC6351] Perreault, S., "xCard: vCard XML Representation", RFC 6351, DOI 10.17487/RFC6351, August 2011, <https://www.rfc-editor.org/info/rfc6351>.

[RFC6901] Bryan, P., Ed., Zyp, K., and M. Nottingham, Ed., "JavaScript Object Notation (JSON) Pointer", RFC 6901, DOI 10.17487/RFC6901, April 2013, <https://www.rfc-editor.org/info/rfc6901>.

[RFC7095] Kewisch, P., "jCard: The JSON Format for vCard", RFC 7095, DOI 10.17487/RFC7095, January 2014, <https://www.rfc-editor.org/info/rfc7095>.

[RFC7493] Bray, T., Ed., "The I-JSON Message Format", RFC 7493, DOI 10.17487/RFC7493, March 2015, <https://www.rfc-editor.org/info/rfc7493>.

[RFC7942] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", BCP 205, RFC 7942, DOI 10.17487/RFC7942, July 2016, <https://www.rfc-editor.org/info/rfc7942>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

[RFC8984] Jenkins, N. and R. Stepanek, "JSCalendar: A JSON Representation of Calendar Data", RFC 8984, DOI 10.17487/RFC8984, July 2021, <https://www.rfc-editor.org/info/rfc8984>.

7.2. Informative References

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, DOI 10.17487/RFC3261, June 2002, <https://www.rfc-editor.org/info/rfc3261>.

[RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002, <https://www.rfc-editor.org/info/rfc3339>.

[RFC3966] Schulzrinne, H., "The tel URI for Telephone Numbers", RFC 3966, DOI 10.17487/RFC3966, December 2004, <https://www.rfc-editor.org/info/rfc3966>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/info/rfc4648>.

[RFC5322] Resnick, P., Ed., "Internet Message Format", RFC 5322, DOI 10.17487/RFC5322, October 2008, <https://www.rfc-editor.org/info/rfc5322>.

[RFC6473] Saint-Andre, P., "vCard KIND:application", RFC 6473, DOI 10.17487/RFC6473, December 2011, <https://www.rfc-editor.org/info/rfc6473>.

[RFC6474] Li, K. and B. Leiba, "vCard Format Extensions: Place of Birth, Place and Date of Death", RFC 6474, DOI 10.17487/RFC6474, December 2011, <https://www.rfc-editor.org/info/rfc6474>.

[RFC6715] Cauchie, D., Leiba, B., and K. Li, "vCard Format Extensions: Representing vCard Extensions Defined by the Open Mobile Alliance (OMA) Converged Address Book (CAB) Group", RFC 6715, DOI 10.17487/RFC6715, August 2012, <https://www.rfc-editor.org/info/rfc6715>.

[RFC6869] Salgueiro, G., Clarke, J., and P. Saint-Andre, "vCard KIND:device", RFC 6869, DOI 10.17487/RFC6869, February 2013, <https://www.rfc-editor.org/info/rfc6869>.

[RFC8605] Hollenbeck, S. and R. Carney, "vCard Format Extensions: ICANN Extensions for the Registration Data Access Protocol (RDAP)", RFC 8605, DOI 10.17487/RFC8605, May 2019, <https://www.rfc-editor.org/info/rfc8605>.

Authors' Addresses

Robert Stepanek
FastMail
PO Box 234, Collins St West
Melbourne VIC 8007
Australia

Email: rsto@fastmailteam.com

Mario Loffredo
IIT-CNR
Via Moruzzi,1
56124 Pisa
Italy

Email: mario.loffredo@iit.cnr.it