Workgroup: Calendaring Extensions
Internet-Draft: draft-ietf-calext-jscontact-06
Published: 10 January 2023
Intended Status: Standards Track
Expires: 14 July 2023
Authors: R. Stepanek M. Loffredo
Fastmail IIT-CNR

JSContact: A JSON representation of contact data

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

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

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

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

This Internet-Draft will expire on 14 July 2023.

Table of Contents

1. Introduction
  1.1. Relation to the xCard and jCard formats
  1.2. Notational Conventions
  1.3. ABNF Notations
  1.4. Type Signatures
  1.5. Data types
    1.5.1. Id
    1.5.2. Int and UnsignedInt
    1.5.3. PatchObject
    1.5.4. Resource
    1.5.5. UTCDateTime
  1.6. Common properties
    1.6.1. The contexts property
    1.6.2. The label property
    1.6.3. The pref property
  1.7. Versioning
    1.7.1. Version Scheme
    1.7.2. Version Updates
  1.8. Validating JSContact Properties
    1.8.1. IANA-registered Properties
    1.8.2. Unknown Properties
  1.9. Vendor-Specific Extensions
    1.9.1. Vendor-specific Properties
    1.9.2. Vendor-specific Values
  1.10. Reserved Property Names
2. Card
  2.1. Metadata properties
    2.1.1. @type
    2.1.2. @version
    2.1.3. created
    2.1.4. kind
    2.1.5. locale
    2.1.6. members
    2.1.7. prodId
    2.1.8. relatedTo
    2.1.9. uid
    2.1.10. updated
  2.2. Name and Organization properties
    2.2.1. fullName
    2.2.2. name
    2.2.3. nickNames
    2.2.4. organizations
    2.2.5. speakToAs
    2.2.6. titles

2.3. Contact properties
2.3.1. emails
2.3.2. onlineServices
2.3.3. phones
2.3.4. preferredContactChannels
2.3.5. preferredLanguages
2.4. Calendaring and Scheduling properties
2.4.1. calendars
2.4.2. schedulingAddresses
2.5. Address and Location properties
2.5.1. addresses
2.6. Resource properties
2.6.1. cryptoKeys
2.6.2. directories
2.6.3. links
2.6.4. media
2.7. Multilingual properties
2.7.1. localizations
2.8. Additional properties
2.8.1. anniversaries
2.8.2. keywords
2.8.3. notes
2.8.4. personalInfo
3. Implementation Status
3.1. IIT-CNR/Registro.it
4. IANA Considerations
4.1. Media Type Registration
4.2. Creation of the "JSContact Properties" Registry
4.2.1. Preliminary Community Review
4.2.2. Submit Request to IANA
4.2.3. Designated Expert Review
4.2.4. Change Procedures
4.2.5. "JSContact Properties" Registry Template
4.2.6. Initial Contents for the "JSContact Properties" Registry
4.3. Creation of the "JSContact Types" Registry
4.3.1. "JSContact Types" Registry Template
4.3.2. Initial Contents for the "JSContact Types" Registry
4.4. Creation of the "JSContact Enum Values" Registry
4.4.1. "JSContact Enum Values" Registry Property Template
4.4.2. "JSContact Enum Values" Registry Value Template
4.4.3. Initial Contents for the "JSContact Enum Values" Registry
5. Security Considerations
5.1. JSON Parsing
5.2. URI Values
6. References
6.1. Normative References
6.2. Informative References
Authors' Addresses

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].

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

* The data model and set of attributes should mostly be compatible with the one defined for the vCard data format [RFC6350] and extensions ([RFC6473], [RFC6474], [RFC6715], [RFC6869], [RFC8605]). The specification should add new attributes or value types where appropriate. Not all existing vCard definitions need an equivalent in JSContact, especially if the vCard definition is considered to be obsolete or otherwise inappropriate. Conversion between the data formats need not fully preserve semantic meaning.

* The attributes of the card 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 JSContact 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. Notational Conventions

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

1.3. ABNF Notations

The ABNF definitions in this document use the notations of [RFC5234]. ABNF rules not defined in this document either are defined in [RFC5234] (such as the ABNF for CRLF, WSP, DQUOTE, VCHAR, ALPHA, and DIGIT) or [RFC6350].

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[] - A JSON 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. 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. The map keys **MUST** be preserved across multiple versions of 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 (Section 2) 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.

1.5.2. Int and UnsignedInt

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

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.

1.5.3. 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 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 are invalid. Implementations **MUST NOT** apply partial patches.

1.5.4. Resource

This data type defines a resource associated with the entity represented by this card, identified by a URI [RFC3986]. Several property definitions later in this document refer to the Resource data type as the basis for their property-specific value types. The Resource data type defines the properties that are common to all of them. Property definitions making use of Resource **MAY** define additional properties for their value types.

A Resource object has the following properties:

* @type: String (mandatory). The allowed values are defined in the property definition that makes use of the Resource type.

* type: String (optional). The type of the resource. The allowed values are defined in the property definition that makes use of the Resource type.

* uri: String (mandatory). The resource value. This **MUST** be a *URI* as defined in Section 3 of [RFC3986] and updates.

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

* contexts: String[Boolean] (optional). The contexts in which to use this resource. Also see Section 1.6.1.

* pref: UnsignedInt (optional). The preference of this resource in relation to other resources. Also see Section 1.6.3.

* label: String (optional). A custom label for the value, see Section 1.6.2.

1.5.5. 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.

1.6. Common properties

Most of the properties in this document are specific to a single JSContact object type. Such properties are defined along with the respective object type. The properties in this section are common to multiple data types and are defined here to avoid repetition. Note that these properties **MUST** only be set for a JSContact object if they are explicitly mentioned to be allowed for this object type.

1.6.1. The contexts property

Type: String[Boolean]

This property associates contact information with one or more contexts in which it should be used. For example, someone might have distinct phone numbers for work and private contexts, and may set the desired context on the respective phone number in the phones (Section 2.3.3) property.

This document defines the following common contexts. Additional contexts may be defined in the properties or data types that make use of this property, may be registered in a future RFC, or be vendor-specific (Section 1.9.1).

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

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

1.6.2. The label property

Type: String

This property allows to associate contact data with user-defined labels. Such labels may be set for phone numbers, email addresses and resources. Typically, these labels are displayed along with their associated contact data in graphical user interfaces. While this specification does not place further restrictions on the value, implementors **SHOULD** take in mind that labels best be succinct, so that they properly display on small graphical interfaces and screens.

1.6.3. The pref property

Type: UnsignedInt

This property allows to define a preference order for contact information. For example, a card holder may have two email addresses and prefer to be contacted with one of them.

Its value **MUST** be 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 independent of each other.

1.7. Versioning

Every instance of a JSContact Card (Section 2) indicates which JSContact version its IANA-registered properties and values are based on. The version is indicated both in the @version (Section 2.1.2) property within the Card and in the version (Section 4.1) parameter of the JSContact MIME content type. All IANA-registered elements indicate the version at which they got introduced or obsoleted.

Implementors are **RECOMMENDED** to always support the latest version.

1.7.1. Version Scheme

A JSContact version consists of a numeric major and minor version. Later versions are numerically higher than former versions, with the major version being more significant than the minor version. A version value is produced by the ABNF

```
jsversion = 1*DIGIT "." 1*DIGIT
```

Differing major version values indicate substantial differences in JSContact semantics and format. Implementations **MUST** be prepared that property definitions and other JSContact elements differ in a backwards-incompatible manner.

Differing minor version values indicate additions that enrich JSContact data, but do not introduce backwards-incompatible changes. Typically, these are new property enum values or properties with narrow semantic scope. A new minor version **MUST NOT** require implementations to change their processing of JSContact data.

1.7.2. Version Updates

If Expert Review or the IETF working group decides that a new major JSContact version is required, a new standard RFC document **MUST** be published. Such an RFC document **MUST** specify all changes to the former JSContact version. An RFC document is not required to change the minor JSContact version.

Every new JSContact version **MUST** be registered at IANA in the JSContact Enum Value registry Table 6.

1.8. Validating JSContact Properties

JSContact objects are represented as I-JSON objects [RFC7493] and the keys of such objects are called properties. This specification distinguishes between three kinds of properties with regards to validation: IANA-registered properties and unknown properties are defined in this section, while vendor-specific properties are defined in Section 1.9.1. A JSContact object is invalid if any its properties are invalid. If a JSContact object is valid, implementations **MUST** preserve all its properties.

1.8.1. IANA-registered Properties

An IANA-registered property is any property that has been registered according to the IANA property registry rules as outlined in Section 4. All properties defined in this specification are IANA-registered properties.

Implementations **MUST** validate IANA-registered properties in JSContact data, unless they are unknown to the implementation (see Section 1.8.2). They **MUST** reject invalid IANA-registered properties. A property is invalid if its name matches the name of an IANA-registered property but the value violates its definition according to the JSContact specification version defined in the Card object @version property (Section 2.1.2).

IANA-registered property names **MUST NOT** contain US-ASCII control characters (U+0000 to U+001F, U+007F), the COLON (U+003A) or QUOTATION MARK (U+0022) characters. They **SHOULD** only contain US-ASCII alphanumeric characters (the ALPHA and DIGIT rules defined in Section 6.1 of [RFC2234]), but notable exceptions of this rule are metadata properties such as @type and @version defined in later

sections of this document. IANA-registered property names **SHOULD** be notated in lower camel case.

1.8.2. Unknown Properties

Implementations may encounter JSContact data where a property name is unknown to that implementation, but the name adheres to the restrictions of an IANA-registered property.

Implementations **MUST NOT** treat such properties as invalid. Instead, they **MUST** preserve them in the JSContact object. Implementations that create or update JSContact data **MUST** only set IANA-registered properties or vendor-specific properties, but **MUST** preserve any already existing unknown properties. This is to allow applications and services to interoperate without data loss, even if they do not implement the same set of JSContact extensions.

1.9. Vendor-Specific Extensions

Vendors may extend properties and values for experimentation or to store contacts data that only is useful for a single service or application. Such extensions are not meant for interoperation and vendors **MUST NOT** expect other implementations to process their contents. If instead interoperation is desired, vendors are strongly encouraged to define and register new properties, types and values at IANA. Typically, sending a short description to the IETF working group mailing list is enough for Expert Review to make a decision. Notably, publishing a new RFC document is not required in the general case. Section 4 defines how to register new properties, types or values at IANA. Section 1.8.1 defines the naming conventions for IANA-registered elements.

1.9.1. Vendor-specific Properties

Vendor-specific properties **MAY** be set in any JSContact object. Implementations **MUST** preserve vendor-specific properties in JSContact data, irrespective if they know their use. They **MUST NOT** reject the property value as invalid, unless they are in control of the vendor-specific property.

Vendor-specific property names **MUST** start with a vendor-specific prefix, followed by the COLON character (U+003A), followed by a name consisting of any other non-control ASCII or non-ASCII characters. The vendor-specific prefix **SHOULD** be a domain name under control of the service or application that sets the property, but it need not resolve in the Domain Name System [RFC1034] and [RFC1035]. The prefix ietf.org and its sub-domain names are reserved for IETF specifications. The name following the prefix **MUST NOT** contain the QUOTATION MARK (U+0022) character. It **SHOULD NOT** contain the TILDE (U+007E) and SOLIDUS (U+002F) characters, as these require special-

escaping when encoding a JSON Pointer [RFC6901] including that property.

The ABNF rule v-extension formally defines valid vendor-specific property names. Note that the vendor prefix allows for more values than are allowed as Internationalized Domain Names (IDN) [RFC8499]. This is to allow JSContact implementations simply validate property names without implementing the full set of rules that apply to domain names.

```
v-extension = v-prefix ":" v-name

v-prefix = v-label *("." v-label)

v-label = alnum-int / alnum-int *(alnum-int / "-") alnum-int

alnum-int = ALPHA / DIGIT / NON-ASCII
  ; see RFC 6350 Section 3.3

v-name = 1*(WSP / "!" / %x23-7e / NON-ASCII)
  ; any characters except CTLs and DQUOTE
  ; use of "/" (%x2f) and "~" (%x7e) is discouraged
```

The value of vendor-specific properties can be any valid JSON value, and naming restrictions do not apply to such values. Specifically, if the property value is a JSON object then the keys of such objects need not be named as vendor-specific properties.

The following all are valid examples of vendor-specific properties.

```
"example.com:foo": "bar",
"example.com:foo2": {
  "bar": "baz"
}
```

Figure 1

1.9.2. Vendor-specific Values

Some JSContact IANA-registered properties allow their values to be vendor-specific. One such example is the kind property Section 2.1.4, which enumerates its standard values but also allows for arbitrary vendor-specific values. Such vendor-specific values **MUST** be valid v-extension values as defined in Section 1.9.1. This is an example for a vendor-specific value:

```
"kind": "example.com:kind:foo"
```

Figure 2

Vendors are strongly encouraged to specify new standard values once a vendor-specific turns out to be useful also for other systems.

1.10. Reserved Property Names

This specification reserves the property name extra at IANA. A JSContact object **MUST NOT** be defined to have a property with this name. This is to provide implementations with a name to map unknown or vendor-specific properties to, and which never may occur as a property in a JSContact object.

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. @version

Type: String (mandatory).

Specifies the JSContact version used to define this card. The value **MUST** be one of the IANA-registered JSContact Enum Values for the @version property. This specification registers the JSContact version value 1.0 (Table 6).

```
"@version": "1.0",
```

Figure 3: @version example

2.1.3. created

Type: UTCDateTime (optional).

The date and time when this Card object was created.

```
"created": "1994-09-30T14:35:10Z",
```

Figure 4: created example

2.1.4. 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 (Section 1.9.1):

- *individual: a single person
- *group: a group person of persons or entities
- *org: an organization
- *location: a named location
- *device: a device, such as appliances, computers, or network elements
- *application: a software application

```
"kind": "individual",
```

Figure 5: kind example

2.1.5. locale

Type: String (optional).

This is the language tag, as defined in [RFC5646], that best describes the locale used for text in the card. Note that such values **MAY** be localized in the localizations property Section 2.7.1.

```
"locale": "de-AT",
```

Figure 6: locale example

2.1.6. members

Type: String[Boolean] (optional).

This identifies the set of cards that are members of this group card. Each key in the set is the uid property value of the member, each boolean value **MUST** be true. If this property is set, then the value of the kind property **MUST** be group.

The opposite is not true. A group Card will usually contain the members property to specify the members of the group, but it is not required to. A group Card without the members property can be

considered an abstract grouping, or one whose members are known empirically (e.g. "IETF Participants").

```
"kind": "group",
"fullName": "The Doe family",
"uid": "urn:uuid:ab4310aa-fa43-11e9-8f0b-362b9e155667",
"members": {
  "urn:uuid:03a0e51f-d1aa-4385-8a53-e29025acd8af": true,
  "urn:uuid:b8767877-b4a1-4c70-9acc-505d3819e519": true
}
```

Figure 7: members example

2.1.7. prodId

Type: String (optional).

The identifier for the product that created the Card object.

```
"prodId": "-//ONLINE DIRECTORY//NONSGML Version 1//EN"
```

Figure 8: prodId example

2.1.8. relatedTo

Type: String[Relation] (optional).

Relates the object to other Card objects. This is represented as a map, where each key is the uid of the related Card 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 (Section 1.9.1). The value for each key in the set **MUST** be true.

```
"relatedTo": {
  "urn:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6": {
    "@type": "Relation",
    "relation": {
      "friend": true
    }
  },
  "8cacdfb7d1ffdb59@example.com": {
    "@type": "Relation",
    "relation": {}
  }
}
```

Figure 9: relatedTo example

2.1.9. uid

Type: String (mandatory).

An identifier, used to associate the object as the same across different systems, address books and views. The value **SHOULD** be a URN [RFC8141] but for compatibility with [RFC6350] it **MAY** also be a URI [RFC3986] or free-text value. The value of the URN **SHOULD** be in the uuid namespace [RFC4122]. As of this writing, a revision of [RFC4122] is being worked on and is likely to introduce new UUID versions and best practices to generate global unique identifiers. Implementors **SHOULD** follow any recommendations described there. Until then, implementations **SHOULD** generate identifiers using the random or pseudo-random UUID version described in Section 4.4 of [RFC4122].

```
"uid": "urn:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6"
```

Figure 10: uid example

2.1.10. updated

Type: UTCDateTime (optional).

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

```
"updated": "1995-10-31T22:27:10Z"
```

Figure 11: updated example

2.2. Name and Organization properties

2.2.1. fullName

Type: String (optional).

This is the full name 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. If the name property is set, the fullName property **SHOULD NOT** be set. If both properties are set, applications **SHOULD** display the contents of the name property as the name of the entity represented by this card. Applications **SHOULD NOT** store the concatenated name component values of the name property in the fullName property value.

```
"fullName": "Mr.  John Q.  Public, Esq."
```

Figure 12: fullName example

2.2.2. 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 as a String produce a valid full name of this entity. This specification does not mandate how to do this but recommends the following: If at least one of two adjacent name components is of type separator then implementations **SHOULD** join their values without any additional character. Otherwise, inserting a single Space character in between name component values is a good choice.

*sortAs: String[String] (optional).

This defines how this name lexicographically sorts in relation to other names when compared by a name component type. The key in the map defines the name component type. The value for that key defines the verbatim string to compare when sorting by this name component type. Absence of a key indicates that this name component type should not be considered during sort. Sorting by

that missing name component type or if the sortAs property is not set is implementation-specific.

Each key in the map **MUST** be a valid name component type value as defined for the type property of the NameComponent object (see below). For each key in the map there **MUST** exist at least one NameComponent object having that type in the components property of this name.

Figure 13 illustrates the use of sortAs. The property value indicates that the middle name followed by both surnames should be used when sorting this name by surname. The absence of the middle indicates that the middle name on its own should be disregarded during sort. Even though the name only contains one name component for the given name, the sortAs property still explicitly defines how to sort by given name as otherwise sorting by it would be undefined.

```
"name": {
  "@type": "Name",
  "components": [
    {
      "@type": "NameComponent",
      "type": "given",
      "value": "Robert"
    },
    {
      "@type": "NameComponent",
      "type": "middle",
      "value": "Pau"
    },
    {
      "@type": "NameComponent",
      "type": "surname",
      "value": "Shou"
    },
    {
      "@type": "NameComponent",
      "type": "surname",
      "value": "Chang"
    }
  ],
  "sortAs": {
    "surname": "Pau Shou Chang",
    "given": "Robert"
  }
}
```

Figure 13: name example

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 (Section 1.9.1):

-prefix. The value is an 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 an 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 hyphen character.

*rank: 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 rank 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 Card object to determine the appropriate formatting. They **MAY** print name components in order of appearance in the components property of the Name object.

2.2.3. nickNames

Type: Id[NickName] (optional).

The nick names of the entity represented by this card. A NickName object has the following properties:

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

*name: String (mandatory). The nick name.

*contexts: String[Boolean] (optional) The contexts in which to use this nick name. Also see Section 1.6.1.

*pref: UnsignedInt (optional). The preference of this nick name in relation to other nick names. Also see Section 1.6.3.

```
"nickNames": {
  "k391": {
    "@type": "NickName",
    "name": "Johnny"
  }
}
```

Figure 14: nickNames example

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, of which at least one of name and units **MUST** be set:

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

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

*units: OrgUnit[] (optional). A list of organizational units. If set, the list **MUST** contain at least one entry.

*sortAs: String (optional). This defines how this organization name lexicographically sorts in relation to other organizations when compared by name. The value defines the verbatim string value to compare. In absence of this property, the name property value **SHOULD** be used for comparison.

*contexts: String[Boolean] (optional). The contexts in which association with this organization apply. For example, membership in a choir may only apply in a private context. Also see Section 1.6.1.

A OrgUnit object has the following properties:

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

*name: String (optional). The name of this organizational unit. If set, the value **MUST** be a non-empty string.

*sortAs: String (optional). This defines how this organization unit name lexicographically sorts in relation to other organizational units of the same level when compared by name. The level is defined by the array index of this organizational unit in the units property of the Organization object. The property value defines the verbatim string value to compare. In absence of this property, the name property value **SHOULD** be used for comparison.

```
"organizations": {
  "o1": {
    "@type": "Organization",
    "name": "ABC, Inc.",
    "units": [
      {
        "@type": "OrgUnit",
        "name": "North American Division"
      },
      {
        "@type": "OrgUnit",
        "name": "Marketing"
      }
    ],
    "sortAs": "ABC"
  }
}
```

Figure 15: organizations example

2.2.5. 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 assigned sex of the contact.

*pronouns: Id[Pronouns] (optional). Defines the pronouns that the contact chooses to use for themselves.

A Pronouns object has the following properties:

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

*pronouns: String (mandatory). Defines the pronouns. Any value or form is allowed. Examples in English include she/her and they/them/theirs. The value **MAY** be overridden in the localizations property (Section 2.7.1).

*contexts: String[Boolean] (optional). The contexts in which to use these pronouns. Also see Section 1.6.1.

*pref: UnsignedInt (optional). The preference of these pronouns in relation to other pronouns in the same context. Also see Section 1.6.3.

```
"speakToAs": {
  "grammaticalGender": "neuter",
  "pronouns": {
    "k19": {
      "@type": "Pronouns",
      "pronouns": "they/them",
      "pref": 2
    },
    "k32": {
      "@type": "Pronouns",
      "pronouns": "xe/xir",
      "pref": 1
    }
  }
}
```

Figure 16: speakToAs example

2.2.6. 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.

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

- *type: String (optional, default title). Describes the organizational or situational type of this title. Some organizations and individuals distinguish between *titles* as organizational positions and *roles* as more temporary assignments, such as in project management. If set, the property value **MUST** either be one of title and role, or be registered in a future RFC, or a vendor-specific value (Section 1.9.1).

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

```
"titles": {
  "le9": {
    "@type": "Title",
    "type": "title",
    "name": "Research Scientist"
  },
  "k2": {
    "@type": "Title",
    "type": "role",
    "name": "Project Leader",
    "organization": "o2"
  }
},
"organizations": {
  "o2": {
    "@type": "Organization",
    "name": "ABC, Inc."
  }
}
```

Figure 17: titles example

2.3. Contact 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.

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

- *contexts: String[Boolean] (optional). The contexts in which to use this email address. Also see Section 1.6.1.

- *pref: UnsignedInt (optional). The preference of this email address in relation to other email addresses. Also see Section 1.6.3.

- *label: String (optional). A custom label for the value, see Section 1.6.2.

```
"emails": {
  "e1": {
    "@type": "EmailAddress",
    "contexts": {
      "work": true
    },
    "address": "jqpublic@xyz.example.com"
  },
  "e2": {
    "@type": "EmailAddress",
    "address": "jane_doe@example.com",
    "pref": 1
  }
}
```

Figure 18: emails example

2.3.2. onlineServices

Type: Id[OnlineService] (optional).

The online services that are associated with the entity represented by this card. This can be messaging services, social media profiles, and other. An OnlineService object has the following properties:

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

* service: String (optional). The name of the online service or protocol. This **SHOULD** be the canonical service name including capitalization. Examples are GitHub, kakao, Mastodon.

* user: String (mandatory). This identifies the entity represented by this card at this online service. The type property defines the how to interpret the value.

* type: String (mandatory). This defines the type of the identifier in the user property. The type **MUST** be either one of the following values, registered in a future RFC, or a vendor-specific value (Section 1.9.1):

  - impp: The value of the user property is a URI primarily used for instant messaging. The service property **SHOULD** be set.

  - uri: The value of the user property is a service-specific URI, such as for a social media service. The service property **SHOULD** be set.

-username: The value of the user property is a service-specific username, such as for a social media service. Any free-text value is allowed. The service property **MUST** be set.

*contexts: String[Boolean] (optional). The contexts in which to use this service. Also see Section 1.6.1.

*pref: UnsignedInt (optional). The preference of this service in relation to other services. Also see Section 1.6.3.

*label: String (optional). A custom label for the value, see Section 1.6.2.

```
"onlineServices": {
  "x1": {
    "@type": "OnlineService",
    "user": "xmpp:alice@example.com",
    "type": "impp",
    "pref": 1
  }
}
```

Figure 19: onlineServices example

2.3.3. 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.

*number: String (mandatory). The phone number, as either a URI or free-text. 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 boolean value **MUST** be true. The method type **MUST** be either one of the following values, registered in a future RFC, or a vendor-specific value (Section 1.9.1):

-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: String[Boolean] (optional). The contexts in which to use this number. Also see Section 1.6.1.

*pref: UnsignedInt (optional). The preference of this number in relation to other numbers. Also see Section 1.6.3.

*label: String (optional). A custom label for the value, see Section 1.6.2.

```
"phones": {
  "tel0": {
    "@type": "Phone",
    "contexts": {
      "private": true
    },
    "features": {
      "voice": true
    },
    "number": "tel:+1-555-555-5555;ext=5555",
    "pref": 1
  },
  "tel3": {
    "@type": "Phone",
    "contexts": {
      "work": true
    },
    "number": "tel:+33-01-23-45-67"
  }
}
```

Figure 20: phones example

2.3.4. preferredContactChannels

Type : String[ContactChannelPreference[]] (optional).

Defines which channel the entity represented by this card prefers to be contacted with. The keys in the object **MUST** be either one of the

following values, registered in a future RFC, or a vendor-specific value (Section 1.9.1):

- *addresses. The entity prefers to be contacted by postal delivery to one of the entries in addresses (Section 2.5.1).

- *emails. The entity prefers to be contacted by one of the entries in emails (Section 2.3.1).

- *onlineServices. The entity prefers to be contacted by one of the entries in onlineServices (Section 2.3.2).

- *phones. The entity prefers to be contacted by one of the entries in phones (Section 2.3.3).

The values in the object are a (possibly empty) list of preferences for this contact channel. A valid ContactChannelPreference object **MUST** have at least one of its properties set in addition to the @type property.

A ContactChannelPreference object has the following properties:

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

- *contexts: String[Boolean] (optional). Defines the contexts in which to use this contact channel. Also see Section 1.6.1.

- *pref: UnsignedInt (optional). Defines the preference of this contact channel in relation to other contact channels with the same contexts. Also see Section 1.6.3.

```
"preferredContactChannels": {
  "emails": [
    {
      "@type": "ContactChannelPreference",
      "pref": 1
    }
  ],
  "phones": [
    {
      "@type": "ContactChannelPreference",
      "pref": 2
    }
  ]
}
```

Figure 21: preferredContactChannels example

2.3.5. preferredLanguages

Type : String[LanguagePreference[]] (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 LanguagePreference object **MUST** have at least one of its properties set in addition to the @type property.

A LanguagePreference object has the following properties:

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

*contexts: String[Boolean] (optional). Defines the contexts in which to use this language. Also see Section 1.6.1.

*pref: UnsignedInt (optional). Defines the preference of this language in relation to other languages of the same contexts. Also see Section 1.6.3.

```
"preferredLanguages": {
  "en": [
    {
      "@type": "LanguagePreference",
      "contexts": {
        "work": true
      },
      "pref": 1
    }
  ],
  "fr": [
    {
      "@type": "LanguagePreference",
      "contexts": {
        "work": true
      },
      "pref": 2
    },
    {
      "@type": "LanguagePreference",
      "contexts": {
        "private": true
      }
    }
  ]
}
```

Figure 22: preferredLanguages example

2.4. Calendaring and Scheduling properties

2.4.1. calendars

Type: Id[CalendarResource] (optional).

These are resources for calendaring, such as calendars to lookup free-busy information for the entity represented by this card. A CalendarResource object has all properties of the Resource (Section 1.5.4) data type, with the following additional definitions:

*The @type property value **MUST** be CalendarResource.

The type property value **MUST** be one of the following, or be defined in a future RFC or vendor-specific:

*calendar The resource is a calendar that contains entries such as calendar events or tasks.

*freeBusy The resource allows for free-busy lookups, for example to schedule group events.

```
"calendars": {
  "calA": {
    "@type": "CalendarResource",
    "type": "calendar",
    "uri": "ftp://ftp.example.com/calA.ics",
    "mediaType": "text/calendar"
  },
  "project-a": {
    "@type": "CalendarResource",
    "type": "freeBusy",
    "uri": "ftp://example.com/busy/project-a.ifb",
    "mediaType": "text/calendar"
  }
}
```

Figure 23: calendars example

2.4.2. schedulingAddresses

Type: Id[SchedulingAddress] (optional).

The scheduling addresses by which the entity may receive calendar scheduling invitations. A SchedulingAddress object has the following properties:

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

*uri: String (mandatory). The address to use for calendar scheduling with this contact. This **MUST** be a URI as defined in Section 3 of [RFC3986] and updates.

*contexts: String[Boolean] (optional). The contexts in which to use this scheduling address. Also see Section 1.6.1.

*pref: UnsignedInt (optional). The preference of this scheduling address in relation to other scheduling address. Also see Section 1.6.3.

*label: String (optional). A custom label for the scheduling address, see Section 1.6.2.

```
"schedulingAddresses": {
  "sched1": {
    "@type": "SchedulingAddress",
    "uri": "mailto:janedoe@example.com"
  }
}
```

Figure 24: schedulingAddresses example

2.5. Address and Location properties

2.5.1. 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.

*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 **MUST** be a time zone name registered in the IANA Time Zone Database

*contexts: String[Boolean] (optional). The contexts of the address information. The boolean value **MUST** be true. In addition to the common contexts (Section 1.6.1), allowed key values are:

-billing An address to be used for billing.

-delivery An address to be used for delivering physical items.

*fullAddress: String (optional). This is the full address, including street, region or country. The purpose of this property is to define an address, even if the individual address components are not known. If the street property is set, the fullAddress property **SHOULD NOT** be set. If both properties are set, applications **SHOULD** display the contents of the street property as the street address of the entity represented by this card. Applications **SHOULD NOT** store the concatenated street component values of the street property in the fullAddress property value.

*pref: UnsignedInt (optional). The preference of this address in relation to other addresses. Also see Section 1.6.3.

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 (Section 1.9.1):

  -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 street component value for which no type is known.

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

```
"addresses": {
  "k23": {
    "@type": "Address",
    "contexts": {
      "work": true
    },
    "fullAddress": "54321 Oak St\nReston\nVA\n20190\nUSA",
    "street": [
      {
        "@type": "StreetComponent",
        "type": "name",
        "value": "Oak St"
      },
      {
        "@type": "StreetComponent",
        "type": "number",
        "value": "54321"
      }
    ],
    "locality": "Reston",
    "region": "VA",
    "country": "USA",
    "postcode": "20190",
    "countryCode": "US"
  },
  "k24": {
    "@type": "Address",
    "contexts": {
      "private": true
    },
    "fullAddress": "12345 Elm St\nReston\nVA\n20190\nUSA",
    "street": [
      {
        "@type": "StreetComponent",
        "type": "name",
        "value": "Elm St"
      },
      {
        "@type": "StreetComponent",
        "type": "number",
        "value": "12345"
      }
    ],
    "locality": "Reston",
    "region": "VA",
    "country": "USA",
    "postcode": "20190",
    "countryCode": "US"
```

```
  }
}
```

Figure 25: addresses example

2.6. Resource properties

2.6.1. cryptoKeys

Type: Id[CryptoResource] (optional).

These are cryptographic resources such as public keys and certificates associated with the entity represented by this card. A CryptoResource object has all properties of the Resource (Section 1.5.4) data type, with the following additional definitions:

- The @type property value **MUST** be CryptoResource.

- The type property value either is not set, is defined in a future RFC or vendor-specific.

```
"cryptoKeys": {
  "mykey": {
    "@type": "CryptoResource",
    "uri": "http://www.example.com/keys/jdoe.cer"
  }
}
```

Figure 26: cryptoKeys example

2.6.2. directories

Type: Id[DirectoryResource] (optional).

These are directory service resources, such as entries in a directory or organizational directories for lookup. A DirectoryResource object has all properties of the Resource (Section 1.5.4) data type, with the following additional definitions:

- The @type property value **MUST** be DirectoryResource.

The type property value **MUST** be one of the following, or be defined in a future RFC or vendor-specific:

- directory The resource is a directory service where the entity represented by this card is part of. This typically is an organizational directory that also contains associated entities, e.g. co-workers and management in a company directory.

*entry The resource is a directory entry of the entity represented by this card. In contrast to the directory type, this is the specific URI for the entity *within* a directory.

In addition, the DirectoryResource object has the following property:

*listAs: UnsignedInt (optional). This defines the position of this directory resource in the list of all DirectoryResource objects having the same type in this card. If set, the listAs value **MUST** be higher than zero. Applications that display the directories of a Card in a list **SHOULD** order them such that entries with a higher property value sort after lower ones. Multiple directory resources **MAY** have the same listAs property value, or none at all. Sorting such entries is implementation-specific.

```
"directories": {
  "dir1": {
    "@type": "DirectoryResource",
    "type": "entry",
    "uri": "http://dir.example.com/addrbook/jdoe/Jean%20Dupont.vcf"
  },
  "dir2": {
    "@type": "DirectoryResource",
    "type": "directory",
    "uri": "ldap://ldap.example/o=Example%20Tech,ou=Engineering",
    "pref": 1
  }
```

Figure 27: directories example

2.6.3. links

Type: Id[LinkResource] (optional).

These are links to resources that do not fit any of the other use-case specific resource properties. A LinkResource object has all properties of the Resource (Section 1.5.4) data type, with the following additional definitions:

*The @type property value **MUST** be LinkResource.

The type property value either is not set, or **MUST** be one of the following, or be defined in a future RFC or vendor-specific:

*contact The resource is an URI by which the entity represented by this card may be contacted, including web forms or other media that require user interaction.

```
"links": {
  "link3": {
    "@type": "LinkResource",
    "type": "contact",
    "uri": "mailto:contact@example.com",
    "pref": 1
  }
}
```

Figure 28: links example

2.6.4. media

Type: Id[MediaResource] (optional).

These are media resources such as photographs, avatars, or sounds associated with the entity represented by this card. A MediaResource object has all properties of the Resource (Section 1.5.4) data type, with the following additional definitions:

- The @type property value **MUST** be MediaResource.

The type property value must be one of the following, or be defined in a future RFC or vendor-specific:

- photo The resource is a photograph or avatar.

- sound The resource is audio media, e.g. to specify the proper pronunciation of the name property contents.

- logo The resource is a graphic image or logo associated with entity represented by this card.

```
"media": {
  "res45": {
    "@type": "MediaResource",
    "type": "sound",
    "uri": "CID:JOHNQ.part8.19960229T080000.xyzMail@example.com"
  },
  "res47": {
    "@type": "MediaResource",
    "type": "logo",
    "uri": "http://www.example.com/pub/logos/abccorp.jpg"
  },
  "res1": {
    "@type": "MediaResource",
    "type": "photo",
    "uri": "..."
  }
}
```

Figure 29: media example

2.7. Multilingual properties

2.7.1. localizations

Type: String[PatchObject] (optional).

This property localizes property values in this Card to languages other than the main locale. Its purpose is to provide language-specific alternatives to existing values, not to add new values. In other words, a localized Card **SHOULD NOT** contain more information than its non-localized variant.

The keys in the localizations property object are language tags [RFC5646]. The values are patch objects which localize the Card in the respective language tag. The paths in the PatchObject are relative to the Card object that includes the localizations property. A patch **MUST NOT** target the localizations property.

Conceptually, a Card is localized as follows:

* Determine the language tag in which this Card should be localized in.

* If the localizations property includes a key for that language, obtain the PatchObject value. If there is no such key, stop.

* Create a copy of the Card, but do not copy the localizations property.

* Apply all patches in the PatchObject to the copy of the Card.

*Optionally, set the locale property in the copy of the Card.

*Use the patched copy of the Card as the localized variant of the original Card.

A patch in the PatchObject may patch a simple-typed property value, or a complex type.

Figure 30 shows how a single String property value is localized in the jp locale.

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

Figure 30

Figure 31 shows how a complete object property value is localized in the en locale.

```
"locale": "ru",
"name": {
  "@type": "Name",
  "components": [
    {
      "@type": "NameComponent",
      "type": "given",
      "value": "Фёдор"
    },
    {
      "@type": "NameComponent",
      "type": "middle",
      "value": "Михайлович"
    },
    {
      "@type": "NameComponent",
      "type": "surname",
      "value": "Достоевский"
    }
  ]
},
"localizations": {
  "en": {
    "name": {
      "@type": "Name",
      "components": [
        {
          "@type": "NameComponent",
          "type": "given",
          "value": "Fyodor"
        },
        {
          "@type": "NameComponent",
          "type": "middle",
          "value": "Mikhailovich"
        },
        {
          "@type": "NameComponent",
          "type": "surname",
          "value": "Dostoevsky"
        }
      ]
    }
  }
}
```

Figure 31

2.8. Additional properties

2.8.1. 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 birthday anniversary

  - -death: a deathday anniversary

  - -wedding: a wedding day anniversary

- *date: Timestamp|PartialDate (mandatory).

  The date of this anniversary in the Gregorian calendar. This **MUST** either be a whole or partial calendar date or a complete UTC timestamp (see the definition of the Timestamp and PartialDate object types below).

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

A Timestamp object has the following properties:

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

- *utc: UTCDateTime (mandatory). Specifies the point in time in UTC time.

A PartialDate object represents a complete or partial calendar date in the Gregorian calendar. It represents either a complete date, or a year, or a month in a year, or a day in a month. It has the following properties, of which at least year or month and day **MUST** be set:

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

- *year: UnsignedInt (optional). This is the calendar year.

*month: UnsignedInt (optional). This is the calendar month, represented as the integers 1 <= month <= 12. If this property is set then either year or day **MUST** be set.

*day: UnsignedInt (optional). This is the calendar month day, represented as the integers 1 <= day <= 31, depending on the validity within the month and year. If this property is set then month **MUST** be set.

*calendarScale: String (optional). This is the calendar system in which this date occurs, in lowercase. This **MUST** be either a CLDR-registered calendar system name [RFC7529] or a vendor-specific value. The year, month and day still **MUST** be represented in the Gregorian calendar. Note that for calendar systems with leap months, the year property might be required to convert between the Gregorian calendar date and the respective calendar system.

```
"anniversaries": {
  "k8": {
    "@type": "Anniversary",
    "type": "birth",
    "date": {
      "@type": "PartialDate",
      "year": 1953,
      "month": 4,
      "day": 15
    }
  },
  "k9": {
    "@type": "Anniversary",
    "type": "death",
    "date": {
      "@type": "Timestamp",
      "utc": "1996-10-15T23:10:00Z"
    },
    "place": {
      "@type": "Address",
      "fullAddress": "4445 Tree Street\nNew England, ND 58647\nUSA"
    }
  }
}
```

Figure 32: anniversaries example

2.8.2. keywords

Type: String[Boolean] (optional). A set of free-text keywords, also known as *tags*. The set is represented as an object, with each key being a keyword. The boolean value **MUST** be true.

```
"keywords": {
  "internet": true,
  "IETF": true
}
```

Figure 33: keywords example

2.8.3. notes

Type: Id[Note] (optional).

Free-text notes associated with this card. A Note object has the following properties:

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

*note: String (mandatory). The free text value of this note.

*created: UTCDateTime (optional). The date and time when this note was created.

*author: Author (optional). The author of this note.

An Author object has the following properties, of which at least one other than @type **MUST** be defined:

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

*name: String (optional). The name of this author.

*uri: String (optional). A URI value that identifies the author.

```
"notes": {
  "n1": {
    "note": "Open office hours are 1600 to 1715 EST, Mon-Fri",
    "created": "2022-11-23T15:01:32Z",
    "author": {
      "@type": "Author",
      "name": "John"
    }
  }
}
```

Figure 34: notes example

2.8.4. personalInfo

Type: Id[PersonalInfo] (optional).

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

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

*type: String (mandatory). Specifies the type for this personal information. The value **MUST** be one of the following, or be registered in a future RFC or vendor-specific (Section 1.9.1):

-expertise: a field of expertise or credential

-hobby: a hobby

-interest: an interest

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

*level: String (optional). Indicates the level of expertise, or engagement in hobby or interest. The value **MUST** be one of the following, or be registered in a future RFC or vendor-specific (Section 1.9.1): high, medium and low.

*listAs: UnsignedInt (optional). This defines the position of this personal information in the list of all PersonalInfo objects having the same type in this card. If set, the listAs value **MUST** be higher than zero. Applications that display personal information entries in a list **SHOULD** order them such that entries with a higher property value sort after lower ones. Multiple personal information entries **MAY** have the same listAs property

value, or none at all. Sorting such entries is implementation-specific.

*label: String (optional). A custom label. See Section 1.6.2.

```
"personalInfo": {
  "pi2": {
    "@type": "PersonalInfo",
    "type": "expertise",
    "value": "chemistry",
    "level": "high"
  },
  "pi1": {
    "@type": "PersonalInfo",
    "type": "hobby",
    "value": "reading",
    "level": "high"
  },
  "pi6": {
    "@type": "PersonalInfo",
    "type": "interest",
    "value": "r&b music",
    "level": "medium"
  }
}
```

Figure 35: personalInfo example

3. 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".

3.1. 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/

*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

4. IANA Considerations

4.1. Media Type Registration

This document defines a media type for use with JSContact data formatted in JSON.

Type name: application

Subtype name: jscontact+json

Required parameters: type

The type parameter conveys the type of the JSContact data in the body part. The allowed parameter values correspond to the @type property of the JSON-formatted JSContact object in the body. This RFC specifies a single type, but future RFC documents **MAY** extend this list:

card: The @type property value of the JSContact object **MUST** be Card.

The parameter **MUST NOT** occur more than once.

Optional parameters: version

The version parameter conveys the version of the JSContact object in the body part. Its value **MUST** match the value of the version property of the JSContact object.

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

Security considerations: See Section 5 of this document.

Interoperability considerations: While JSContact is designed to avoid ambiguities as much as possible, when converting objects from other contact formats to/from JSContact, it is possible that differing representations for the same logical data or ambiguities in interpretation might arise. The semantic equivalence of two JSContact objects may be determined differently by different applications, for example, where URL values differ in case between the two objects.

Published specification: TBD

Applications that use this media type: Applications that currently make use of the text/vcard media type can use this as an alternative.

Fragment identifier considerations: A JSON Pointer fragment identifier may be used, as defined in [RFC6901], Section 6.

Additional information:

Magic number(s): N/A

File extensions(s): N/A

Macintosh file type code(s): N/A

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

Intended usage: COMMON

Restrictions on usage: N/A

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

Change controller: IETF

4.2. Creation of the "JSContact Properties" Registry

IANA has created the "JSContact Properties" registry to allow interoperability of extensions to JSContact objects.

This registry follows the Expert Review process ([RFC8126], Section 4.5). If the "Intended Usage" field is common, sufficient documentation is required to enable interoperability. Preliminary community review for this registry is optional but strongly encouraged.

A registration can have an intended usage of common, reserved, or obsolete. IANA will list registrations with a common usage designation prominently and separately from those with other intended usage values.

A reserved registration reserves a property name without assigning semantics to avoid name collisions with future extensions or protocol use.

An obsolete registration denotes a property that is no longer expected to be added by up-to-date systems. A new property has probably been defined covering the obsolete property's semantics.

Every registration **MUST** define the version of the JSContact specification on which the definition of the newly registered, updated or obsoleted item is based on. This typically is the latest specification version that is in effect when the property gets registered. The version **MUST** be one of the allowed values of the @version property in the JSContact Enum Value registry (see Table 6).

The JSContact property registration procedure is not a formal standards process but rather an administrative procedure intended to allow community comment and check it is coherent without excessive time delay. It is designed to encourage vendors to document and register new properties they add for use cases not covered by the original specification, leading to increased interoperability.

4.2.1. Preliminary Community Review

Notice of a potential new registration **SHOULD** be sent to the Calext mailing list <calsify@ietf.org> for review. This mailing list is appropriate to solicit community feedback on a proposed new property.

Property registrations must be marked with their intended use: "common", "reserved", or "obsolete".

The intent of the public posting to this list is to solicit comments and feedback on the choice of the property name, the unambiguity of the specification document, and a review of any interoperability or

security considerations. The submitter may submit a revised registration proposal or abandon the registration completely at any time.

4.2.2. Submit Request to IANA

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

4.2.3. Designated Expert Review

The primary concern of the designated expert (DE) is preventing name collisions and encouraging the submitter to document security and privacy considerations. For a common-use registration, the DE is expected to confirm that suitable documentation, as described in Section 4.6 of [RFC8126], is available to ensure interoperability. That documentation will usually be in an RFC, but simple definitions are likely to use a web/wiki page, and if a sentence or two is deemed sufficient, it could be described in the registry itself. The DE should also verify that the property name does not conflict with work that is active or already published within the IETF. A published specification is not required for reserved or obsolete registrations.

The DE will either approve or deny the registration request and publish a notice of the decision to the Calext WG mailing list or its successor, as well as inform IANA. A denial notice must be justified by an explanation, and, in the cases where it is possible, concrete suggestions on how the request can be modified so as to become acceptable should be provided.

4.2.4. Change Procedures

Once a JSContact property has been published by IANA, the change controller may request a change to its definition. The same procedure that would be appropriate for the original registration request is used to process a change request.

JSContact property registrations may not be deleted; properties that are no longer believed appropriate for use can be declared obsolete by a change to their "intended usage" field; such properties will be clearly marked in the IANA registry.

Significant changes to a JSContact property's definition should be requested only when there are serious omissions or errors in the published specification, as such changes may cause interoperability issues. When review is required, a change request may be denied if it renders entities that were valid under the previous definition invalid under the new definition.

The owner of a JSContact property may pass responsibility to another person or agency by informing IANA; this can be done without discussion or review.

4.2.5. "JSContact Properties" Registry Template

Property Name: This is the name of the property. The property name **MUST NOT** already be registered for any of the object types listed in the "Property Context" field of this registration. Other object types **MAY** already have registered a different property with the same name; however, the same name **SHOULD** only be used when the semantics are analogous.

Property Type: This is the type of this property, using type signatures, as specified in Section 1.4. The property type **MUST** be registered in the "JSContact Types" registry.

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

Reference or Description: This is a brief description or RFC number and section reference where the property is specified (omitted for "reserved" property names). This must include references to all RFC documents where this property is introduced or updated.

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

Since Version: This defines the JSContact version on which this property definition is based on. The version **MUST** be one of the allowed values of the @version property in the JSContact Enum Value registry (see Table 6).

Until Version: This defines the JSContact version after which this property got obsoleted and **MUST NOT** be used in later versions. The Until Version value either **MUST NOT** be set, or be one of the allowed values of the @version property in the JSContact Enum Value registry (see Table 6).

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

4.2.6. Initial Contents for the "JSContact Properties" Registry

The following table lists the initial common usage entries of the "JSContact Properties" registry. The Since Version for all properties is 1.0. The Until Version for all properties is not set. All RFC section references are for this document. The change controller for all these properties is IETF.

Property Name	Property Type	Property Context	Re
@type	String	Address, Anniversary, Author, Card, CalendarResource, ContactChannelPreference, CryptoResource, DirectoryResource, EmailAddress, LanguagePreference, LinkResource, MediaResource, Name, NameComponent, NickName, Note, OnlineService, Organization, OrgUnit, PartialDate,PersonalInfo, Phone, Pronouns, Relation, Resource, SchedulingAddress, SpeakToAs, StreetComponent, Timestamp, Title	Se Se
@version	String	Card	Se
address	String	EmailAddress	Se
addresses	Id[Address]	Card	Se
anniversaries	Id[Anniversary]	Card	Se
author	Author	Note	Se
calendars	Id[CalendarResource]	Card	Se
calendarScale	String	PartialDate	Se
components	NameComponent[]	Name	Se
contexts	String[Boolean]	Address, NickName, Pronouns, EmailAddress, OnlineService, Phone, ContactChannelPreference, LanguagePreference, CalendarResource, CryptoResource, DirectoryResource, LinkResource, MediaResource, Organization, SchedulingAddress	Se
coordinates	String	Address	Se
country	String	Address	Se
countryCode	String	Address	Se
created	UTCDateTime	Card, Note	Se
date	Timestamp|PartialDate	Anniversary	Se
day	UnsignedInt	PartialDate	Se
directories	Id[DirectoryResource]	Card	Se

Property Name	Property Type	Property Context	Re
emails	Id[EmailAddress]	Card	Se
features	String[Boolean]	Phone	Se
fullAddress	String	Address	Se
fullName	String	Card	Se
grammaticalGender	String	SpeakToAs	Se
keywords	String[Boolean]	Card	Se
kind	String	Card	Se
label	String	EmailAddress, OnlineService, Phone, CalendarResource, CryptoResource, DirectoryResource, LinkResource, MediaResource, PersonalInfo, SchedulingAddress	Se
level	String	PersonalInfo	Se
links	Id[LinkResource]	Card	Se
listAs	UnsignedInt	DirectoryResource, PersonalInfo	Se
locale	String	Card	Se
locality	String	Address	Se
localizations	String[PatchObject]	Card	Se
media	Id[MediaResource]	Card	Se
mediaType	String	CalendarResource, CryptoResource, DirectoryResource, LinkResource, MediaResource	Se
members	String[Boolean]	Card	Se
month	UnsignedInt	PartialDate	Se
name	Name	Card	Se
name	String	Author, NickName, Organization, OrgUnit, Title	Se
nickNames	Id[NickName]	Card	Se
note	String	Note	Se
notes	Id[Note]	Card	Se
number	String	Phone	Se
onlineServices	Id[OnlineService]	Card	Se
organization	String	Title	Se
organizations	Id[Organization]	Card	Se
personalInfo	Id[PersonalInfo]	Card	Se
phones	Id[Phone]	Card	Se
place	Address	Anniversary	Se
postcode	String	Address	Se

Property Name	Property Type	Property Context	Re
pref	UnsignedInt	Address, NickName, Pronouns, EmailAddress, OnlineService, Phone, ContactChannelPreference, LanguagePreference, CalendarResource, CryptoResource, DirectoryResource, LinkResource, MediaResource, SchedulingAddress	Se
preferredContactChannels	String[ContactChannelPreference[]]	Card	Se
preferredLanguages	String[LanguagePreference[]]	Card	Se
prodId	String	Card	Se
pronouns	Id[Pronouns]	SpeakToAs	Se
pronouns	String	Pronouns	Se
rank	UnsignedInt	NameComponent	Se
region	String	Address	Se
relatedTo	String[Relation]	Card	Se
relation	String[Boolean]	Relation	Se
schedulingAddresses	Id[SchedulingAddress]	Card	Se
service	String	OnlineService	Se
sortAs	String[String]	Name	Se
sortAs	String	Organization, OrgUnit	Se
speakToAs	SpeakToAs	Card	Se
street	StreetComponent[]	Address	Se
timeZone	String	Address	Se
titles	Id[Title]	Card	Se
type	String	Anniversary, NameComponent, Title, CalendarResource, CryptoResource, DirectoryResource, LinkResource, MediaResource, OnlineService, StreetComponent, PersonalInfo	Se
uid	String	Card	Se
units	OrgUnit[]	Organization	Se
updated	UTCDateTime	Card	Se
uri	String	Author, CalendarResource, CryptoResource, DirectoryResource, LinkResource, MediaResource, SchedulingAddress	Se

Property Name	Property Type	Property Context	Re
user	String	OnlineService	Se
utc	UTCDateTime	Timestamp	Se
value	String	NameComponent, StreetComponent, PersonalInfo	Se
year	UnsignedInt	PartialDate	Se

The following table lists the initial reserved usage entries of the "JSContact Properties" registry. All RFC section references are for this document. The change controller for all these properties is IETF.

Property Name	Property Type	Property Context	Reference or Description	Intended Usage
extra	not applicable	not applicable	Section 1.10	reserved

Table 2: Initial Contents of the "JSContact Properties" Registry

4.3. Creation of the "JSContact Types" Registry

IANA has created the "JSContact Types" registry to avoid name collisions and provide a complete reference for all data types used for JSContact property values. The registration process is the same as for the "JSContact Properties" registry, as defined in Section 4.2.

4.3.1. "JSContact Types" Registry Template

Type Name: the name of the type

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

Intended Usage: common, reserved, or obsolete

Since Version: This defines the JSContact version on which this type definition is based on. The version **MUST** be one of the allowed values of the @version property in the JSContact Enum Value registry (see Table 6).

Until Version: This defines the JSContact version after which this type definition got obsoleted and **MUST NOT** be used in later versions. The Until Version value either **MUST** be not set, or one of the allowed values of the @version property in the JSContact Enum Value registry (see Table 6).

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

4.3.2. Initial Contents for the "JSContact Types" Registry

The following table lists the initial common usage entries of the JSContact Types registry. The Since Version for all types is 1.0. The Until Version for all types is not set. All RFC section references are for this document. The change controller for all these properties is IETF.

Type Name	Reference or Description
Address	Section 2.5.1
Anniversary	Section 2.8.1
Author	Section 2.8.3
Boolean	Section 1.4
CalendarResource	Section 2.4.1
Card	Section 2
ContactChannelPreference	Section 2.3.4
CryptoResource	Section 2.6.1
DirectoryResource	Section 2.6.2
EmailAddress	Section 2.3.1
Id	Section 1.5.1
Int	Section 1.5.2
LanguagePreference	Section 2.3.5
LinkResource	Section 2.6.3
MediaResource	Section 2.6.4
Name	Section 2.2.2
NameComponent	Section 2.2.2
NickName	Section 2.2.3
Note	Section 2.8.3
OnlineService	Section 2.3.2
Organization	Section 2.2.4
OrgUnit	Section 2.2.4
PartialDate	Section 2.8.1
PersonalInfo	Section 2.8.4
Phone	Section 2.3.3
Pronouns	Section 2.2.5
Relation	Section 2.1.8
SchedulingAddress	Section 2.4.2
SpeakToAs	Section 2.2.5
StreetComponent	Section 2.5.1
Timestamp	Section 2.8.1
Title	Section 2.2.6
UnsignedInt	Section 1.5.2

Type Name	Reference or Description
UTCDateTime	Section 1.5.5

Table 3: Initial Contents of the "JSContact Types" Registry

The following table lists the initial reserved usage entries of the JSContact Types registry. All types are for version 1.0. All RFC section references are for this document. The change controller for all these properties is IETF.

Type Name	Reference or Description
Resource	Section 1.5.4

Table 4: Initial Contents of the "JSContact Types" Registry

4.4. Creation of the "JSContact Enum Values" Registry

IANA has created the "JSContact Enum Values" registry to allow interoperable extension of semantics for properties with enumerable values. Each such property will have a subregistry of allowed values. The registration process for a new enum value or adding a new enumerable property is the same as for the "JSContact Properties" registry, as defined in Section 4.2.

4.4.1. "JSContact Enum Values" Registry Property Template

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

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

Context: This is the list of allowed object types where the property or properties may appear, as registered in the "JSContact Properties" registry. This disambiguates where there may be two distinct properties with the same name in different contexts.

Since Version: This defines the JSContact version on which this enum value definition is based on. The version **MUST** be one of the allowed values of the @version property in the JSContact Enum Value registry (see Table 6).

Until Version: This defines the JSContact version after which this enum value definition got obsoleted and **MUST NOT** be used in later versions. The Until Version value either **MUST** be not set, or one of the allowed values of the @version property in the JSContact Enum Value registry (see Table 6).

Change Controller:
(IETF for properties defined in RFCs from the IETF stream).

Initial Contents: This is the initial list of defined values for this enum, using the template defined in Section 4.4.2. A subregistry will be created with these values for this property name/context tuple.

4.4.2. "JSContact Enum Values" Registry Value Template

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

Enum Value: The verbatim value of the enum

Reference or Description: A brief description or RFC number and section reference for the semantics of this value

Since Version: The JSContact version on which the enum value definition is based on. The version **MUST** be one of the allowed values of the @version property in the JSContact Enum Value registry (see Table 6).

Until Version: The JSContact version after which this enum value got obsoleted and **MUST NOT** be used in later versions. The Until Version value either **MUST** be not set, or one of the allowed values of the @version property in the JSContact Enum Value registry (see Table 6).

4.4.3. Initial Contents for the "JSContact Enum Values" Registry

For each subregistry created in this section, all RFC section references are for this document. For all entries, the Since Version is 1.0, the Until Version is not set, the Change Controller is IETF.

Property Name: kind
Context: Card
Initial Contents:

Enum Value	Reference or Description
individual	Section 2.1.4
group	Section 2.1.4
org	Section 2.1.4
location	Section 2.1.4
device	Section 2.1.4
application	Section 2.1.4

Table 5: JSContact Enum Values for kind (Context: Card)

Property Name: @version
Context: Card
Initial Contents:

Enum Value	Reference or Description
1	Section 2.1.2

Table 6: JSContact Enum Values for @version (Context: Card)

Property Name: contexts
Context: NickName, Pronouns, EmailAddress, OnlineService, Phone, ContactChannelPreference, LanguagePreference, CalendarResource, CryptoResource, DirectoryResource, LinkResource, MediaResource, SchedulingAddress
Initial Contents:

Enum Value	Reference or Description
private	Section 1.6.1
work	Section 1.6.1

Table 7: JSContact Enum Values for contexts (Context: NickName, Pronouns, EmailAddress, OnlineService, Phone, ContactChannelPreference, LanguagePreference, CalendarResource, CryptoResource, DirectoryResource, LinkResource, MediaResource, SchedulingAddress)

Property Name: contexts
Context: Address
Initial Contents:

Enum Value	Reference or Description
private	Section 1.6.1
work	Section 1.6.1
billing	Section 2.5.1
delivery	Section 2.5.1

Table 8: JSContact Enum Values for contexts (Context: Address)

Property Name: features
Context: Phone
Initial Contents:

Enum Value	Reference or Description
voice	Section 2.3.3
fax	Section 2.3.3
pager	Section 2.3.3
text	Section 2.3.3
cell	Section 2.3.3
textphone	Section 2.3.3
video	Section 2.3.3

Table 9: JSContact Enum Values for features (Context: Phone)

Property Name: type
Context: StreetComponent
Initial Contents:

Enum Value	Reference or Description
name	Section 2.5.1
number	Section 2.5.1
apartment	Section 2.5.1
room	Section 2.5.1
extension	Section 2.5.1
direction	Section 2.5.1
building	Section 2.5.1
floor	Section 2.5.1
postOfficeBox	Section 2.5.1
separator	Section 2.5.1
unknown	Section 2.5.1

Table 10: JSContact Enum Values for type (Context: StreetComponent)

Property Name: type
Context: NameComponent
Initial Contents:

Enum Value	Reference or Description
prefix	Section 2.2.2
given	Section 2.2.2
surname	Section 2.2.2
middle	Section 2.2.2
suffix	Section 2.2.2
separator	Section 2.2.2

Table 11: JSContact Enum Values for type (Context: NameComponent)

Property Name: type
Context: Title
Initial Contents:

Enum Value	Reference or Description
title	Section 2.2.6
role	Section 2.2.6

Table 12: JSContact Enum Values for type (Context: Title)

Property Name: grammaticalGender
Context: SpeakToAs
Initial Contents:

Enum Value	Reference or Description
animate	Section 2.2.5

Enum Value	Reference or Description
female	Section 2.2.5
inanimate	Section 2.2.5
male	Section 2.2.5
neuter	Section 2.2.5

Table 13: JSContact Enum Values for type (Context: SpeakToAs)

Property Name: preferredContactChannels
Context: Card
Initial Contents:

Enum Value	Reference or Description
addresses	Section 2.3.4
emails	Section 2.3.4
onlineServices	Section 2.3.4
phones	Section 2.3.4

Table 14: JSContact Enum Values for preferredContactChannels (Context: Card)

Property Name: type
Context: CalendarResource
Initial Contents:

Enum Value	Reference or Description
calendar	Section 2.4.1
freeBusy	Section 2.4.1

Table 15: JSContact Enum Values for type (Context: CalendarResource)

Property Name: type
Context: DirectoryResource
Initial Contents:

Enum Value	Reference or Description
directory	Section 2.6.2
entry	Section 2.6.2

Table 16: JSContact Enum Values for type (Context: DirectoryResource)

Property Name: type
Context: LinkResource
Initial Contents:

Enum Value	Reference or Description
contact	Section 2.6.3

Table 17: JSContact Enum Values for type (Context: LinkResource)

Property Name: type
Context: MediaResource
Initial Contents:

Enum Value	Reference or Description
photo	Section 2.6.4
sound	Section 2.6.4
logo	Section 2.6.4

Table 18: JSContact Enum Values for type (Context: MediaResource)

Property Name: type
Context: Anniversary
Initial Contents:

Enum Value	Reference or Description
birth	Section 2.8.1
death	Section 2.8.1
wedding	Section 2.8.1

Table 19: JSContact Enum Values for type (Context: Anniversary)

Property Name: type
Context: OnlineService
Initial Contents:

Enum Value	Reference or Description
impp	Section 2.3.2
uri	Section 2.3.2
username	Section 2.3.2

Table 20: JSContact Enum Values for type (Context: OnlineService)

Property Name: type
Context: PersonalInfo
Initial Contents:

Enum Value	Reference or Description
expertise	Section 2.8.4
hobby	Section 2.8.4
interest	Section 2.8.4

Table 21: JSContact Enum Values for type (Context: PersonalInfo)

5. Security Considerations

Contact information is very privacy sensitive. It can reveal the identity, location and credentials information, employment status, interests and hobbies, and social network of a user. Its transmission and storage must be done carefully to protect it from possible threats, such as eavesdropping, replay, message insertion, deletion, modification, and on-path attacks.

The data being stored and transmitted may be used in systems with real-world consequences. For example, a malicious actor might provide JSContact data that uses the name of another person but insert their contact details to impersonate the unknown victim. Such systems must be careful to authenticate all data they receive to prevent them from being subverted and ensure the change comes from an authorized entity.

This document only defines the data format; such considerations are primarily the concern of the API or method of storage and transmission of such files.

5.1. JSON Parsing

The security considerations of [RFC8259] apply to the use of JSON as the data interchange format.

As for any serialization format, parsers need to thoroughly check the syntax of the supplied data. JSON uses opening and closing tags for several types and structures, and it is possible that the end of the supplied data will be reached when scanning for a matching closing tag; this is an error condition, and implementations need to stop scanning at the end of the supplied data.

JSON also uses a string encoding with some escape sequences to encode special characters within a string. Care is needed when processing these escape sequences to ensure that they are fully formed before the special processing is triggered, with special care taken when the escape sequences appear adjacent to other (non-escaped) special characters or adjacent to the end of data (as in the previous paragraph).

If parsing JSON into a non-textual structured data format, implementations may need to allocate storage to hold JSON string elements. Since JSON does not use explicit string lengths, the risk of denial of service due to resource exhaustion is small, but implementations may still wish to place limits on the size of allocations they are willing to make in any given context, to avoid untrusted data causing excessive memory allocation.

5.2. URI Values

Several JSContact properties contain URIs as values, and processing these properties requires extra care. Section 7 of [RFC3986] discusses security risks related to URIs.

Fetching remote resources carries inherent risks. Connections must only be allowed on well-known ports, using allowed protocols (generally, just HTTP/HTTPS on their default ports). The URL must be resolved externally and not allowed to access internal resources.

Connecting to an external source reveals IP (and therefore often location) information.

A maliciously constructed JSContact object may contain a very large number of URIs. In the case of published address books with a large number of subscribers, such objects could be widely distributed. Implementations should be careful to limit the automatic fetching of linked resources to reduce the risk of this being an amplification vector for a denial-of-service attack.

6. References

6.1. Normative References

[RFC1034] Mockapetris, P. and RFC Publisher, "Domain names - concepts and facilities", STD 13, RFC 1034, DOI 10.17487/RFC1034, November 1987, <https://www.rfc-editor.org/info/rfc1034>.

[RFC1035] Mockapetris, P. and RFC Publisher, "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, <https://www.rfc-editor.org/info/rfc1035>.

[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. and RFC Publisher, "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>.

[RFC2234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, DOI 10.17487/RFC2234, November 1997, <https://www.rfc-editor.org/info/rfc2234>.

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

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/

RFC5234, January 2008, <https://www.rfc-editor.org/info/rfc5234>.

[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>.

[RFC7529] Daboo, C. and G. Yakushev, "Non-Gregorian Recurrence Rules in the Internet Calendaring and Scheduling Core Object Specification (iCalendar)", RFC 7529, DOI 10.17487/RFC7529, May 2015, <https://www.rfc-editor.org/info/rfc7529>.

[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>.

[RFC8126] Cotton, M., Leiba, B., Narten, T., and RFC Publisher, "Guidelines for Writing an IANA Considerations Section in

RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[RFC8141] Saint-Andre, P. and J. Klensin, "Uniform Resource Names (URNs)", RFC 8141, DOI 10.17487/RFC8141, April 2017, <https://www.rfc-editor.org/info/rfc8141>.

[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>.

6.2. 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>.

[RFC8174] Leiba, B. and RFC Publisher, "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>.

[RFC8499] Hoffman, P., Sullivan, A., and K. Fujiwara, "DNS Terminology", BCP 219, RFC 8499, DOI 10.17487/RFC8499, January 2019, <https://www.rfc-editor.org/info/rfc8499>.

[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