

Internet Engineering Task Force  
Internet-Draft  
Intended status: Standards Track  
Expires: February 8, 2020

B. Rosen  
J. Malloy  
B. Henderson  
The MITRE Corporation  
August 7, 2019

Interoperability Profile for Relay User Equipment  
draft-rosen-rue-01

Abstract

This document identifies a minimum set of standards and requirements that must be supported by a Video Relay Service (VRS) Video Access Technology Reference Platform (VATRP)-compliant client and United States Telecommunications Relay Service providers required to be VATRP compliant. This Relay User Equipment specification only specifies a minimum set of requirements. It does not prohibit VRS providers or endpoint developers from developing or deploying additional capabilities, provided that doing so will not prevent compliance with the requirements specified here.

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 February 8, 2020.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	3
2. Scope . . . . .	3
3. Terminology . . . . .	3
4. Requirements Language . . . . .	6
5. General Requirements . . . . .	6
6. SIP Signaling . . . . .	6
6.1. Registration . . . . .	7
6.2. Session Establishment . . . . .	8
6.2.1. Normal Call Origination . . . . .	8
6.2.2. One-Stage Dial-Around Origination . . . . .	9
6.2.3. RUE Contact Information . . . . .	10
6.2.4. Incoming Calls . . . . .	10
6.2.5. Emergency Calls . . . . .	11
6.3. Mid Call Signaling . . . . .	11
6.4. URI Representation of Phone Numbers . . . . .	12
6.5. Transport . . . . .	12
7. Media . . . . .	12
7.1. SRTP and SRTCP . . . . .	12
7.2. Text-Based Communication . . . . .	13
7.3. Video . . . . .	13
7.4. Audio . . . . .	13
7.5. DTMF Digits . . . . .	13
7.6. Session Description Protocol . . . . .	13
7.7. Privacy . . . . .	13
7.8. Negative Acknowledgment, Packet Loss Indicator, and Full Intraframe Request Features . . . . .	13
8. Contacts . . . . .	14
8.1. CardDAV Login and Synchronization . . . . .	14
8.2. Contacts Import/Export Service . . . . .	14
9. Mail Waiting Indicator (MWI) . . . . .	15
10. Provisioning and Provider Selection . . . . .	15
10.1. RUE Provider Selection . . . . .	15
10.2. RUE Configuration Service . . . . .	16
10.3. Schemas . . . . .	19
11. Acknowledgements . . . . .	22
12. IANA Considerations . . . . .	22
13. Security Considerations . . . . .	22
14. Normative References . . . . .	22
Authors' Addresses . . . . .	28

## 1. Introduction

Video Relay Service (VRS) is a form of Telecommunications Relay Service (TRS) that enables persons with hearing disabilities who use sign language, such as American Sign Language (ASL), to communicate with voice telephone users through video equipment. These services also enable communication between such individuals directly in suitable modalities, including any combination of sign language via video, real-time text (RTT), and speech.

This Interoperability Profile for Relay User Equipment (RUE) is a profile of the Session Initiation Protocol (SIP) and related media protocols that enables end-user equipment registration and calling for VRS calls. It specifies the minimal set of call flows, Internet Engineering Task Force (IETF) and ITU-T standards that must be supported, provides guidance where the standards leave multiple implementation options, and specifies minimal and extended capabilities for RUE calls.

This RUE profile supports the requirements of relay services in the United States, as described in 47 CFR 64.601 et seq., but may be applicable to similar uses elsewhere.

## 2. Scope

This RUE Specification documents the standards and controls associated with the Video Access Technology Reference Platform (VATRP). This RUE specification identifies the minimum set of standards for the interface between the VATRP and Providers' networks to which the VATRP adheres. This RUE Specification does not prohibit the implementation of additional features or functionality by any Provider. It also contains some Provider-optional features. If a Provider offers the feature described by the optional specification on at least one endpoint, the Provider MUST supply the standardized interface described in this document for that feature. This edition of the RUE specification does not address Provider-to-Provider communication (covered in the US VRS Provider Interface Profile) or the user interface to the RUE.

## 3. Terminology

**Communication Assistant (CA):** The ASL interpreter stationed in a TRS-registered call center working for a VRS Provider, acting as part of the wire of a call to provide functionally equivalent phone service.

**Communication modality (modality):** A specific form of communication that may be employed by two users, e.g., English voice, Spanish voice, American Sign Language, English lip-reading, or French real-

time-text. Here, one communication modality is assumed to encompass both the language and the way that language is exchanged. For example, English voice and French voice are two different communication modalities.

**Default video relay service:** The video relay service operated by a subscriber's default VRS provider.

**Default video relay service Provider (default Provider):** The VRS provider that registers, and assigns a telephone number to, a specific subscriber. A subscriber's default Provider provides the VRS that handles incoming relay calls to the user. The default Provider also handles outgoing relay calls by default.

**Dial-around call:** A relay call where the subscriber specifies the use of a VRS provider other than one of the Providers with whom the subscriber is registered. This can be accomplished by the user dialing a "front-door" number for a VRS provider and signing or texting a phone number to call ("two-stage"). Alternatively, this can be accomplished by the user's RUE software instructing the server of its default VRS provider to automatically route the call through the alternate Provider to the desired public switched telephone network (PSTN) directory number ("one-stage").

**Full Intra Request (FIR):** A request to a media sender, requiring that media sender to send a Decoder Refresh Point at the earliest opportunity. FIR is sometimes known as "instantaneous decoder refresh request", "video fast update request", or "fast update request".

**NANP:** North America Numbering Plan (please refer to: <http://nationalnanpa.org>).

**Point-to-Point Call (P2P Call):** A call between two RUEs, without including a CA.

**Relay call:** A call that allows persons with hearing or speech disabilities to use a RUE to talk to users of traditional voice services with the aid of a communication assistant (CA) to relay the communication. Please refer to FCC-VRS-GUIDE.

**Relay number database (RND):** The iTRS Relay Number Database (RND) functions as a 10-digit NANP phone number lookup for SIP and H.323 URLs for TRS subscribers.

**Relay-to-relay call:** A call between two subscribers each using different forms of relay (video relay, IP relay, TTY), each with a separate CA to assist in relaying the conversation.

Relay service (RS): A service that allow a registered subscriber to use a RUE to make and receive relay calls, point-to-point calls, and relay-to-relay calls. The functions provided by the relay service include the provision of media links supporting the communication modalities used by the caller and callee, and user registration and validation, authentication, authorization, automatic call distributor (ACD) platform functions, routing (including emergency call routing), call setup, mapping, call features (such as call forwarding and video mail), and assignment of CAs to relay calls.

Relay service Provider (Provider): An organization that operates a relay service. A subscriber selects a relay service Provider to assign and register a telephone number for their use, to register with for receipt of incoming calls, and to provide the default service for outgoing calls.

Relay user: Please refer to "subscriber".

Relay user E.164 Number (user E.164): The telephone number assigned to the RUE in ITU-T E.164 format.

Relay user equipment (RUE): A SIP user agent (UA) enhanced with extra features to support a subscriber in requesting and using relay calls. A RUE may take many forms, including a stand-alone device; an application running on a general-purpose computing device such as a laptop, tablet or smart phone; or proprietary equipment connected to a server that provides the RUE interface.

Sign language: A language that uses hand gestures and body language to convey meaning including, but not limited to, American Sign Language (ASL).

Subscriber: An individual who has registered with a Provider and who obtains service by using relay user equipment. This is the traditional telecom term for an end-user customer, which in our case is a relay user.

Telecommunications relay services (TRS): Telephone transmission services that provide the ability for an individual who has a hearing impairment or speech impairment to engage in communication by wire or radio with a hearing individual in a manner that is functionally equivalent to the ability of an individual who does not have a hearing impairment or speech impairment to communicate using voice communication services by wire or radio. TRS includes services that enable two-way communication between an individual who uses a Telecommunications Device for the Deaf (TDD) or other non-voice terminal device and an individual who does not use such a device.

Video relay service (VRS): A relay service for people with hearing or speech disabilities who use sign language to communicate using video equipment (video RUE) with other people in real time. The video link allows the CA to view and interpret the subscriber's signed conversation and relay the conversation back and forth with the other party.

#### 4. Requirements Language

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119]

#### 5. General Requirements

All HTTP/HTTPS connections specified throughout this document MUST use HTTPS. Both HTTPS and all SIP connections MUST use TLS conforming to [RFC7525]

During the establishment of secure connections with a provider, the RUE MAY be asked by the server for a client certificate. In that case it SHOULD provide the provisioned client certificate (See Section 10.2. Providers MAY reject requests that fail to provide a recognized certificate.

All text data payloads not otherwise constrained by a specification in another standards document MUST be encoded as Unicode UTF/8.

#### 6. SIP Signaling

The RUE and Providers MUST conform to the following core SIP standards [RFC3261] (Base SIP) [RFC3263] (Locating SIP Servers), [RFC3264] (Offer/Answer), [RFC3840] (User Agent Capabilities), [RFC5626] (Outbound), [RFC4566] (Session Description Protocol), [RFC3323] (Privacy), [RFC3605] (RTCP Attribute in SDP), [RFC6665] (SIP Events), [RFC3311] (UPDATE Method), [RFC5393] (Loop-Fix), [RFC5658] (Record Route fix), [RFC5954] (ABNF fix), [RFC3960] (Early Media), and [RFC6442] (Geolocation Header).

In addition, the RUE MUST, and Providers MAY, conform to [RFC3327] (Path), [RFC5245] (ICE), [RFC3326] (Reason header), [RFC3515] (REFER Method), [RFC3891] (Replaces Header), [RFC3892] (Referred-By).

RUEs MUST include a "User-Agent" header field uniquely identifying the RUE application, platform, and version in all SIP requests, and MUST include a "Server" header field with the same content in SIP responses.

### 6.1. Registration

The RUE MUST register with a SIP registrar, following [RFC3261] and [RFC5626]. If the configuration (please refer to Section 11) contains multiple "outbound-proxies", then the RUE MUST use them as specified in [RFC5626] to establish multiple flows.

The request-URI for the REGISTER request MUST contain the "provider-domain" from the configuration. The To-URI and From-URI MUST be identical URIs, formatted as specified in Section 13, using the "phone-number" and "provider-domain" from the configuration.

The RUE determines the URI to resolve by initially determining if an outbound proxy is configured. If it is, the URI will be that of the outbound proxy. If no outbound proxy is configured, the URI will be the Request-URI from the REGISTER request. The RUE extracts the domain from that URI and consults the DNS record for that domain. The DNS entry MUST contain NAPTR records conforming to RFC3263. One of those NAPTR records MUST specify TLS as the preferred transport for SIP. For example, a DNS NAPTR query for "sip:pl.red.example.netv" could return:

```
IN NAPTR 50 50 "s" "SIPS+D2T" "" _sips._tcp.pl.red.example.net
IN NAPTR 90 50 "s" "SIP+D2T" "" _sip._tcp.pl.red.example.net
```

If the RUE receives a 439 (First Hop Lacks Outbound Support) response to a REGISTER request, it MUST re-attempt registration without using the outbound mechanism.

The registrar MAY authenticate using SIP MD5 digest authentication. The credentials to be used (username and password) MUST be supplied within the credentials section of the configuration and identified by the realm the registrar uses in a digest challenge. This username/password combination SHOULD NOT be the same as that used for other purposes, such as retrieving the RUE configuration or logging into the Provider's customer service portal. Because MD5 is considered insecure, [I-D.yusef-sipcore-digest-scheme] SHOULD be implemented by both the RUE and Providers and SHA-based digest algorithms SHOULD be used for digest authentication.

If the registration request fails with an indication that credentials from the configuration are invalid, then the RUE SHOULD retrieve a fresh version of the configuration. If credentials from a freshly retrieved configuration are found to be invalid, then the RUE MUST cease attempts to register and SHOULD inform the RUE User of the problem.

Support for multiple simultaneous registrations by Providers is OPTIONAL, as described in Section 2.

Multiple simultaneous RUE SIP registrations from different RUE devices with the same SIP URI SHOULD be permitted by the Provider. The Provider MAY limit the total number of simultaneous registrations. When a new registration request is received that results in exceeding the limit on simultaneous registrations, the Provider MAY then prematurely terminate another registration; however, it SHOULD NOT do this if it would disconnect an active call.

If a Provider prematurely terminates a registration to reduce the total number of concurrent registrations with the same URI, it SHOULD take some action to prevent the affected RUE from automatically re-registering and re-triggering the condition.

## 6.2. Session Establishment

### 6.2.1. Normal Call Origination

After initial SIP registration, the RUE adheres to SIP [RFC3261] basic call flows, as documented in [RFC3665].

The RUE MUST route all calls through the outbound proxy of the default Provider.

INVITE requests used to initiate calls SHOULD NOT contain Route headers. Route headers MAY be included in one-stage dial-around calls and emergency calls. The SIP URIs in the To field and the Request-URI MUST be formatted as specified in subsection 6.4 using the destination phone number. The domain field of the URIs SHOULD be the "provider-domain" from the configuration (e.g., sip:+13115552368@red.example.com;user=phone). The same exceptions apply, including anonymous calls.

Anonymous calls MUST be supported by both the RUE and Providers. An anonymous call is signaled per [RFC3323].

The From-URI MUST be formatted as specified in Section 6.4, using the phone-number and "provider-domain" from the configuration. It SHOULD also contain the display-name from the configuration when present. (Please refer to Section 10.2.)

Negotiated media MUST follow the guidelines specified in Section 7 of this document.



To allow time to timeout an unanswered call and direct it to a videomail server, the User Agent Client MUST NOT impose a time limit less than the default SIP Invite transaction timeout of 3 minutes.

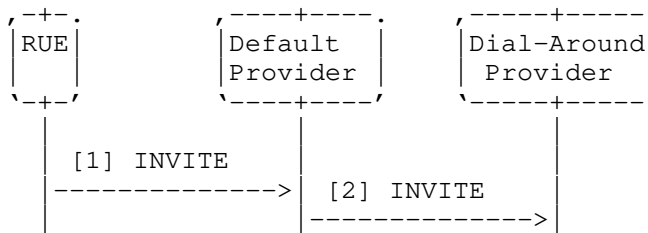
#### 6.2.2. One-Stage Dial-Around Origination

Outbound dial-around calls allow a RUE user to select any Provider to provide interpreting services for any call. "Two-stage" dial-around calls involve the RUE calling a telephone number that reaches the dial-around Provider and using signing or DTMF to provide the called party telephone number. In two-stage dial-around, the To URI is the URI of the dial-around Provider and the domain of the URI is the Provider domain from the configuration.

One-stage dial-around is a method where the called party telephone number is provided in the To URI and the Request-URI, using the domain of the dial-around Provider.

For one-stage dial-around, the RUE MUST follow the procedures in Section 6.2.1 with the following exception: the domain part of the SIP URIs in the To field and the Request-URI MUST be the domain of the dial-around Provider, discovered according to Section 10.1.

The following is a partial example of a one-stage dial-around call from VRS user +1-555-222-0001 hosted by red.example.com to a hearing user +1-555-123-4567 using dial-around to green.example.com for the relay service. Only important details of the messages are shown and many header fields have been omitted:



#### Message Details:

[1] INVITE Rue -> Default Provider

```

INVITE sip:+15551234567@green.example.net;user=phone SIP/2.0
To: <sip:+15551234567@green.example.net;user=phone>
From: "Bob Smith" <sip:+18135551212@red.example.net;user=phone>
Route: sip:green.example.net
  
```

[2] INVITE Default Provider -> Dial-Around Provider

```

INVITE sip:+15551234567@green.example.net;user=phone SIP/2.0
To: <sip:+15551234567@green.example.net;user=phone>
From: "Bob Smith" sip:+18135551212@red.example.net;user=phone
P-Asserted-Identity: sip:+18135551212@red.example.net
  
```

#### One Stage Dial-Around

##### 6.2.3. RUE Contact Information

To identify the owner of a RUE, the initial INVITE for a call from a RUE, or the 200 OK accepting a call by a RUE, identifies the owner by sending a Call-Info header with a purpose parameter of "rue-owner". The URI MAY be an HTTPS URI or Content-Indirect URL. The latter is defined by [RFC2392] to locate message body parts. This URI type is present in a SIP message to convey the RUE ownership information as a MIME body. The form of the RUE ownership information is an xCard [RFC6351]. Please refer to [RFC6442] for an example of using Content-Indirect URLs in SIP messages. Note that use of the Content-Indirect URL usually implies multiple message bodies ("mime/multipart").

##### 6.2.4. Incoming Calls

The RUE MUST accept inbound calls sent to it by the proxy mentioned in the configuration.

If Multiple simultaneous RUE SIP registrations from different RUE devices with the same SIP URI exist, the Provider MUST parallel fork the call to all registered RUEs so that they ring at the same time. The first RUE to reply with a 200 OK answers the call and the Provider MUST CANCEL other call branches.

#### 6.2.5. Emergency Calls

The RUE MUST comply with [RFC6881] for handling of emergency calls.

Providers MAY comply with RFC6881 for handling of emergency calls. In addition, they MUST:

- o Accept RUE emergency calls complying with the specifications in this document;
- o Recognize such calls as emergency calls and properly handle them as such;
- o Address other behavior not specified by RFC6881 as specified in Section 6.2.

Specifically, if the emergency call is to be handled using E9-1-1 (VPC) procedures, the Provider is responsible for modifying the INVITE to conform to the VPC requirements. In this case, location MAY be extracted from the RFC6881 conformant INVITE and used to propagate it to the VPC where possible with the emergency call. Because the RUE may have a more accurate and timely location of the device than the typical manual entry location for nomadic RUE devices, the RUE MUST send a Geolocation header containing its location in the REGISTER request if the configuration specifies it. The Provider MAY use that information to populate the location of the device in the VPC before any emergency call.

#### 6.3. Mid Call Signaling

The RUE and Providers MUST support re-INVITE to renegotiate media session parameters (among other uses). Per Section 7.1, the RUE MUST, and providers SHOULD, be able to support an INFO request for full frame refresh for devices in a call with the RUE that do not support RTCP mechanisms (please refer to Section 7.8). The RUE MUST support an in-dialog REFER ([RFC3515] updated by [RFC7647] and including support for norefersub per [RFC4488]) with the Replaces header [RFC3891] to enable call transfer.

#### 6.4. URI Representation of Phone Numbers

SIP URIs constructed from non-URI sources (dial strings) and sent to SIP proxies by the RUE MUST be represented as follows, depending on whether they can be represented as an E.164 number.

A dial string that can be written as an E.164 formatted phone number MUST be represented as a SIP URI with a URI ";user=phone" tag. The user part of the URI MUST be in conformance with 'global-number' defined in [RFC3966]. The user part MUST NOT contain any 'visual-separator' characters.

Dial strings that cannot be written as E.164 numbers MUST be represented as dialstring URIs, as specified by [RFC4967], e.g., sip:411@red.example.net;user=dialstring.

The domain part of Relay Service URIs and User Address of Records (AoR) MUST (using resolve (in accord with [RFC3263])) to globally routable IPv4 addresses. The AoRs MAY also resolve to IPv6 addresses.

#### 6.5. Transport

The RUE and providers MUST conform to [I-D.ietf-rtcweb-transports] with the understanding that this specification does not use the WebRTC data channel.

The RUE and providers MUST support SIP outbound [RFC5626] (please also refer to Section 6.1).

### 7. Media

This specification adopts the media specifications for WebRTC ([I-D.ietf-rtcweb-overview]). Where WebRTC defines how interactive media communications may be established using a browser as a client, this specification assumes a normal SIP call. The RTP, RTCP, SDP and specific media requirements specified for WebRTC are adopted for this document. The following sections specify the WebRTC documents to which conformance is required.

#### 7.1. SRTP and SRTCP

The RUE and Providers MUST support [I-D.ietf-rtcweb-rtp-usage] with the understanding that RUE does not specify an API and therefore MediaStreamTracks are not used. Implementations MUST conform to Section 6.4 of [I-D.ietf-rtcweb-security-arch].

## 7.2. Text-Based Communication

The RUE MUST and Providers MUST support real-time text ([RFC4102] and [RFC4103]) via T.140 media. One original and two redundant generations MUST be transmitted and supported, with a 300 ms transmission interval. Note that this is not how real time text is transmitted in WebRTC and some form of transcoder would be required to interwork real time text in the data channel of WebRTC to RFC4103 real time text.

## 7.3. Video

The RUE and Providers MUST conform to [RFC7742].

## 7.4. Audio

The RUE and Providers MUST conform to [RFC7874].

## 7.5. DTMF Digits

The RUE and Providers MUST support the "audio/telephone-event" [RFC4733] media type. They MUST support conveying event codes 0 through 11 (DTMF digits "0"-"9", "\*", "#") defined in Table 7 of [RFC4733]. Handling of other tones is OPTIONAL.

## 7.6. Session Description Protocol

The SDP offers and answers MUST conform [I-D.ietf-rtcweb-jsep] with the understanding that the RUE uses SIP transport for SDP.

## 7.7. Privacy

The RUE MUST be able to control privacy of the user by implementing a one-way mute of audio and or video, without signaling, locally, but MUST maintain any NAT bindings by periodically sending media packets on all active media sessions containing silence/comfort noise/black screen/etc. per [RFC6263].

## 7.8. Negative Acknowledgment, Packet Loss Indicator, and Full Intraframe Request Features

NACK SHOULD be used when negotiated and conditions warrant its use. Signaling picture losses as Packet Loss Indicator (PLI) SHOULD be preferred, as described in [RFC5104].

FIR SHOULD be used only in situations where not sending a decoder refresh point would render the video unusable for the users, as per RFC5104 subsection 4.3.1.2.

For backwards compatibility with calling devices that do not support the foregoing methods, the RUE MUST implement and use SIP INFO messages to send and receive XML encoded Picture Fast Update messages according to [RFC5168].

## 8. Contacts

### 8.1. CardDAV Login and Synchronization

Support of CardDAV by Providers is OPTIONAL, as described in Section 2.

The RUE MUST and Providers MAY be able to synchronize the user's contact directory between the RUE endpoint and one maintained by the user's VRS provider using CardDAV ([RFC6352] and [RFC6764]).

The configuration MAY supply a username and domain identifying a CardDAV server and address book for this account.

To access the CardDAV server and address book, the RUE MUST follow Section 6 of RFC6764, using the chosen username and domain in place of an email address. If the request triggers a challenge for digest authentication credentials, the RUE MUST attempt to continue using matching "credentials" from the configuration. If no matching credentials are configured, the RUE MUST use the SIP credentials from the configuration. If the SIP credentials fail, the RUE MUST query the user.

Synchronization using CardDAV MUST be a two-way synchronization service, with proper handling of asynchronous adds, changes, and deletes at either end of the transport channel.

### 8.2. Contacts Import/Export Service

Each Provider MUST supply a standard xCard import/export interface and the RUE MUST be able to export/import the list of contacts in xCard [RFC6351] XML format.

The RUE accesses this service via the "contacts" URI in the configuration. The URL MUST resolve to identify a web server resource that imports/exports contact lists for authorized users.

The RUE stores/retrieves the contact list (address book) by issuing an HTTPS POST or GET request. If the request triggers a challenge for digest authentication credentials, the RUE MUST attempt to continue using matching "credentials" from the configuration. If no credentials are configured, the RUE MUST query the user.

## 9. Mail Waiting Indicator (MWI)

Support of MWI by Providers is OPTIONAL, as described in Section 2

The RUE MUST and Providers SHOULD support subscriptions to "message-summary" events [RFC3842] to the URI specified in the configuration if the Provider supports message waiting indicator on any endpoint.

In notification bodies, videomail messages SHOULD be reported using "message-context-class multimedia-message" defined in [RFC3458].

## 10. Provisioning and Provider Selection

### 10.1. RUE Provider Selection

To allow the user to select a relay service, the RUE MAY obtain, on startup, a list of Providers from a configured accessible URL.

The provider list, formatted as JSON, contains:

- o Version: Specifies the version number of the Provider list format. A new version number SHOULD only be used if the new version is not backwards-compatible with the older version. A new version number is not needed if new elements are optional and can be ignored by older implementations.
- o Providers: An array where each entry describes one Provider. Each entry consists of the following items:
  - \* name: This parameter contains the text label identifying the Provider and is meant to be displayed to the human VRS user.
  - \* domain: The domain parameter is used for configuration purposes by the RUE (as discussed in Section 10.2) and as the domain to use when targeting one-stage dial-around calls to this Provider (as discussed in Section 6.2.2).
  - \* operator: (OPTIONAL) The operator parameter is a SIP URL that identifies the operator "front-door" that VRS users may contact for manual (two-stage) dial-around calls.

The VRS user interacts with the RUE to select from the Provider list one or more Providers with whom the user has already established an account.

```
{
  "version": 1,
  "providers": [
    {
      "name": "Red",
      "domain": "red.example.net",
      "operator": "sip:operator@red.example.net"
    },
    {
      "name": "Green",
      "domain": "green.example.net",
      "operator": "sip:+18885550123@green.example.net;user=phone"
    },
    {
      "name": "Blue",
      "domain": "blue.example.net"
    }
  ]
}
```

Example of a Provider list JSON object

## 10.2. RUE Configuration Service

The RUE is provisioned with one or more URIs that may be queried for configuration with HTTPS.

The data returned will include a set of key/value configuration parameters to be used by the RUE, formatted as a JSON object and identified by the associated [RFC7159] "application/json" MIME type, to allow for other formats in the future.

The configuration data payload includes the following data items. Items not noted as (OPTIONAL) are REQUIRED. If other unexpected items are found, they MUST be ignored.

- o **version:** Identifies the version of the configuration data format. A new version number SHOULD only be used if the new version is not backwards-compatible with the older version. A new version number is not needed if new elements are optional and can be ignored by older implementations.
- o **lifetime:** Specifies how long (in seconds) the RUE MAY cache the configuration values. Values may not be valid when lifetime expires. Emergency Calls MUST continue to work.
- o **display-name:** (OPTIONAL) A user-friendly name to identify the subscriber when originating calls.



- o **phone-number:** The telephone number (in E.164 format) assigned to this subscriber. This becomes the user portion of the SIP URI identifying the subscriber.
- o **provider-domain:** The DNS domain name of the default Provider servicing this subscriber.
- o **outbound-proxies:** (OPTIONAL) A URI of a SIP proxy to be used when sending requests to the Provider.
- o **mw:** (OPTIONAL) A URI identifying a SIP event server that generates "message-summary" events for this subscriber.
- o **videomail:** (OPTIONAL) A SIP URI that can be called to retrieve videomail messages.
- o **contacts:** An HTTPS URI that may be used to export (retrieve) the subscriber's complete contact list managed by the Provider.
- o **carddav:** (OPTIONAL) A username and domain name (separated by "@"") identifying a "CardDAV" server and user name that can be used to synchronize the RUE's contact list with the contact list managed by the Provider.
- o **sendLocationWithRegistration:** True if the RUE should send a Geolocation Header with REGISTER, false if it should not. Defaults to false if not present.
- o **turn-servers:** (OPTIONAL) An array of URLs identifying STUN and TURN servers available for use by the RUE for establishing media streams in calls via the Provider.
- o **credentials:** (OPTIONAL) TBD

```
{
  "version": 1,
  "lifetime": 86400,
  "display-name" : "Bob Smith",
  "phone-number": "+18135551212",
  "provider-domain": "red.example.net",
  "outbound-proxies": [
    "sip:p1.red.example.net",
    "sip:p2.red.example.net"
  ],
  "mw": "sip:+18135551212@red.example.net",
  "videomail": "sip:+18135551212@vm.red.example.net",
  "contacts": "https://red.example.net:443/contacts/1d5545awd"
  "carddav": "bob@red.example.com" ,
}
```

```
"sendLocationWithRegistration": false,
"ice-servers": [
  {"stun:stun.l.google.com:19302" },
  {"turn:turn.red.example.net:3478"}
],
"credentials": [
  {
    "realm": "red.example.net",
    "username": "bob",
    "password": "reg-pw"
  },
  {
    "realm": "proxies.red.example.net",
    "username": "bob",
    "password": "proxy-pw"
  },
  {
    "realm": "cd.red.example.net",
    "username": "bob",
    "password": "cd-pw"
  },
  {
    "realm": "vm.red.example.net",
    "username": "bob",
    "password": "vm-pw"
  },
  {
    "realm": "stun-turn.red.example.net",
    "username": "bob",
    "password": "stun-turn-pw"
  }
]
}
```

#### Example JSON configuration payload

The wire format of the data is in keeping with the standard JSON description in RFC7159.

The "lifetime" parameter in the configuration indicates how long the RUE MAY cache the configuration values. If the RUE caches configuration values, it MUST cryptographically protect them. The RUE SHOULD retrieve a fresh copy of the configuration before the lifetime expires or as soon as possible after it expires. The lifetime is not guaranteed: the configuration may change before the lifetime value expires. In that case, the Provider MAY indicate this by generating authorization challenges to requests and/or prematurely terminating a registration.

Note: In some cases, the RUE may successfully retrieve a fresh copy of the configuration using digest credentials cached from the prior retrieval. If this is not successful, then the RUE will need to ask the user for the username and password. Unfortunately, this authentication step might occur when the user is not present, preventing SIP registration and thus incoming calls. To avoid this situation, the RUE MAY retrieve a new copy of the configuration when it knows the user is present, even if there is time before the lifetime expires.

### 10.3. Schemas

The following JSON schemas are for the Provider List and the RUE Configuration. These are represented using the JSON Content Rules [JCR] schema notation.

```
{
  "version": 1,
  "providers": [
    1*
    {
      "name": string,
      "domain": fqdn,
      "?operator":
        uri,                                ; "front-door" access to provider
        * /^.*$/ : any                      ; (sip uri)
      ; (allow future extensions)
    }
  ] ,
  * /^.*$/ : any                          ; (allow future extensions)
}
```

Provider List JSON Schema

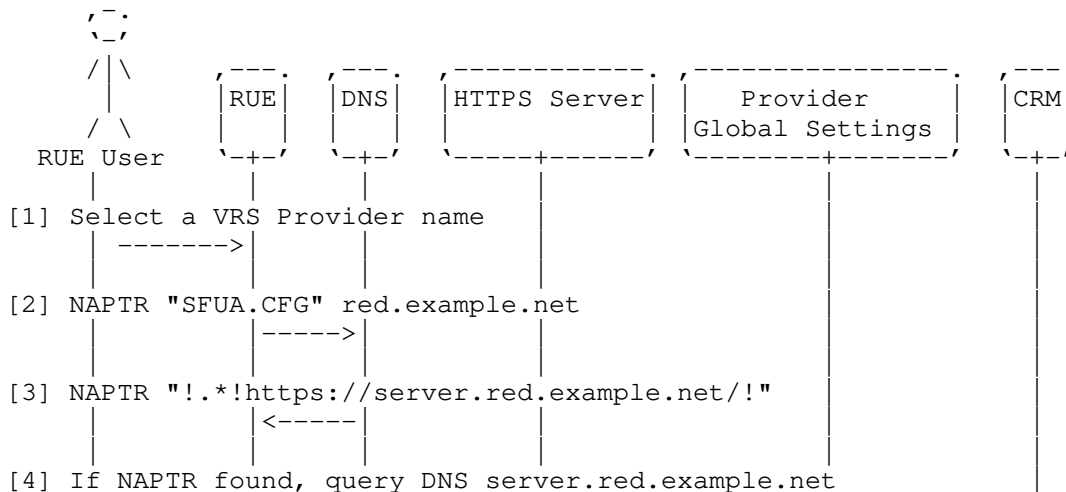
```

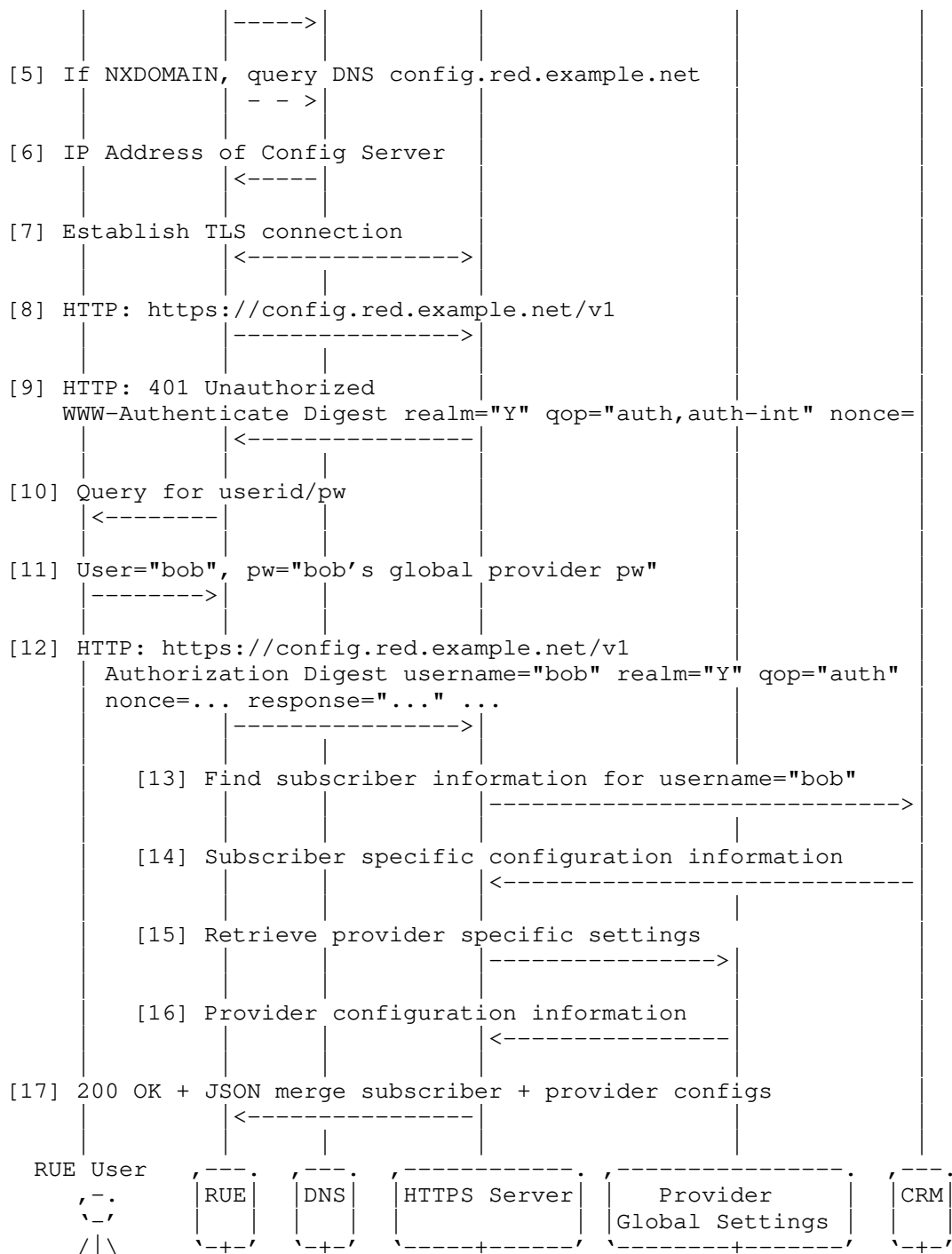
{
  "version": 1,                ; Interface version
  "lifetime": integer,         ; Deadline (in seconds) for
                                ; refreshing this config without
                                ; user input.
  "phone-number": /^\\+[0-9]+$ / , ; E.164 phone number
                                ; for this user
  ?"display-name" : string, ; display name for From: header
  "provider-domain": fqdn, ; SHOULD match that in Provider List
  ?"outbound-proxies": [ 1* : uri ], ; sip URIs
  ?"mwi": uri ,                ; sip URI for MWI subscriptions
  ?"videomail": uri ,          ; sip URI for videomail retrieval
  "contacts": uri ,            ; https URI for contact list retrieval
  ?"carddav": /^\\[^@]+@\\[^@]+$ / , ; for contact list synch
  ?"sendLocationWithRegistration": boolean , ; send location y/n
  ?"ice-servers":              ; (Required for ICE use)
    [ 1* : uri ],              ; (stun[s] & turn[s] URIs
  ?"credentials":              ; for digest authentication
    [ 1* {
      "realm": string,
      "username": string,
      "password": string
    } ],
  * /^\\.\\*$ / : any           ; (allow future extensions)
}

```

#### RUE Configuration JSON Schema

The following illustrates the message flow for retrieving a RUE automatic configuration using HTTPS Digest Authentication:







## RUE Configuration Retrieval

## 11. Acknowledgements

## 12. IANA Considerations

This memo includes no request to IANA.

## 13. Security Considerations

The RUE is required to communicate with servers on public IP addresses and specific ports to perform its required functions. If it is necessary for the RUE to function on a corporate or other network that operates a default-deny firewall between the RUE and these services, the user must arrange with their network manager for passage of traffic through such a firewall in accordance with the protocols and associated SRV records as exposed by the Provider. Because VRS providers may use different ports for different services, these port numbers may differ from Provider to Provider.

## 14. Normative References

## [I-D.ietf-rtcweb-jsep]

Uberti, J., Jennings, C., and E. Rescorla, "JavaScript Session Establishment Protocol", draft-ietf-rtcweb-jsep-26 (work in progress), February 2019.

## [I-D.ietf-rtcweb-overview]

Alvestrand, H., "Overview: Real Time Protocols for Browser-based Applications", draft-ietf-rtcweb-overview-19 (work in progress), November 2017.

## [I-D.ietf-rtcweb-rtp-usage]

Perkins, C., Westerlund, M., and J. Ott, "Web Real-Time Communication (WebRTC): Media Transport and Use of RTP", draft-ietf-rtcweb-rtp-usage-26 (work in progress), March 2016.

## [I-D.ietf-rtcweb-security-arch]

Rescorla, E., "WebRTC Security Architecture", draft-ietf-rtcweb-security-arch-20 (work in progress), July 2019.

## [I-D.ietf-rtcweb-transports]

Alvestrand, H., "Transports for WebRTC", draft-ietf-rtcweb-transports-17 (work in progress), October 2016.

- [I-D.yusef-sipcore-digest-scheme] Shekh-Yusef, R., "The Session Initiation Protocol (SIP) Digest Authentication Scheme", draft-yusef-sipcore-digest-scheme-07 (work in progress), April 2019.
- [pip] SIPForum, "VRS US Providers Profile TWG-6-1.0", 2015, <<https://www.sipforum.org/download/vrs-us-providers-profile-twg-6-1-0-pdf/#>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC2392] Levinson, E., "Content-ID and Message-ID Uniform Resource Locators", RFC 2392, DOI 10.17487/RFC2392, August 1998, <<https://www.rfc-editor.org/info/rfc2392>>.
- [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>>.
- [RFC3263] Rosenberg, J. and H. Schulzrinne, "Session Initiation Protocol (SIP): Locating SIP Servers", RFC 3263, DOI 10.17487/RFC3263, June 2002, <<https://www.rfc-editor.org/info/rfc3263>>.
- [RFC3264] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", RFC 3264, DOI 10.17487/RFC3264, June 2002, <<https://www.rfc-editor.org/info/rfc3264>>.
- [RFC3311] Rosenberg, J., "The Session Initiation Protocol (SIP) UPDATE Method", RFC 3311, DOI 10.17487/RFC3311, October 2002, <<https://www.rfc-editor.org/info/rfc3311>>.
- [RFC3323] Peterson, J., "A Privacy Mechanism for the Session Initiation Protocol (SIP)", RFC 3323, DOI 10.17487/RFC3323, November 2002, <<https://www.rfc-editor.org/info/rfc3323>>.
- [RFC3326] Schulzrinne, H., Oran, D., and G. Camarillo, "The Reason Header Field for the Session Initiation Protocol (SIP)", RFC 3326, DOI 10.17487/RFC3326, December 2002, <<https://www.rfc-editor.org/info/rfc3326>>.

- [RFC3327] Willis, D. and B. Hoeneisen, "Session Initiation Protocol (SIP) Extension Header Field for Registering Non-Adjacent Contacts", RFC 3327, DOI 10.17487/RFC3327, December 2002, <<https://www.rfc-editor.org/info/rfc3327>>.
- [RFC3458] Burger, E., Candell, E., Eliot, C., and G. Klyne, "Message Context for Internet Mail", RFC 3458, DOI 10.17487/RFC3458, January 2003, <<https://www.rfc-editor.org/info/rfc3458>>.
- [RFC3515] Sparks, R., "The Session Initiation Protocol (SIP) Refer Method", RFC 3515, DOI 10.17487/RFC3515, April 2003, <<https://www.rfc-editor.org/info/rfc3515>>.
- [RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, DOI 10.17487/RFC3550, July 2003, <<https://www.rfc-editor.org/info/rfc3550>>.
- [RFC3605] Huitema, C., "Real Time Control Protocol (RTCP) attribute in Session Description Protocol (SDP)", RFC 3605, DOI 10.17487/RFC3605, October 2003, <<https://www.rfc-editor.org/info/rfc3605>>.
- [RFC3665] Johnston, A., Donovan, S., Sparks, R., Cunningham, C., and K. Summers, "Session Initiation Protocol (SIP) Basic Call Flow Examples", BCP 75, RFC 3665, DOI 10.17487/RFC3665, December 2003, <<https://www.rfc-editor.org/info/rfc3665>>.
- [RFC3840] Rosenberg, J., Schulzrinne, H., and P. Kyzivat, "Indicating User Agent Capabilities in the Session Initiation Protocol (SIP)", RFC 3840, DOI 10.17487/RFC3840, August 2004, <<https://www.rfc-editor.org/info/rfc3840>>.
- [RFC3842] Mahy, R., "A Message Summary and Message Waiting Indication Event Package for the Session Initiation Protocol (SIP)", RFC 3842, DOI 10.17487/RFC3842, August 2004, <<https://www.rfc-editor.org/info/rfc3842>>.
- [RFC3891] Mahy, R., Biggs, B., and R. Dean, "The Session Initiation Protocol (SIP) "Replaces" Header", RFC 3891, DOI 10.17487/RFC3891, September 2004, <<https://www.rfc-editor.org/info/rfc3891>>.
- [RFC3892] Sparks, R., "The Session Initiation Protocol (SIP) Referred-By Mechanism", RFC 3892, DOI 10.17487/RFC3892, September 2004, <<https://www.rfc-editor.org/info/rfc3892>>.



- [RFC3960] Camarillo, G. and H. Schulzrinne, "Early Media and Ringing Tone Generation in the Session Initiation Protocol (SIP)", RFC 3960, DOI 10.17487/RFC3960, December 2004, <<https://www.rfc-editor.org/info/rfc3960>>.
- [RFC3966] Schulzrinne, H., "The tel URI for Telephone Numbers", RFC 3966, DOI 10.17487/RFC3966, December 2004, <<https://www.rfc-editor.org/info/rfc3966>>.
- [RFC4102] Jones, P., "Registration of the text/red MIME Sub-Type", RFC 4102, DOI 10.17487/RFC4102, June 2005, <<https://www.rfc-editor.org/info/rfc4102>>.
- [RFC4103] Hellstrom, G. and P. Jones, "RTP Payload for Text Conversation", RFC 4103, DOI 10.17487/RFC4103, June 2005, <<https://www.rfc-editor.org/info/rfc4103>>.
- [RFC4488] Levin, O., "Suppression of Session Initiation Protocol (SIP) REFER Method Implicit Subscription", RFC 4488, DOI 10.17487/RFC4488, May 2006, <<https://www.rfc-editor.org/info/rfc4488>>.
- [RFC4566] Handley, M., Jacobson, V., and C. Perkins, "SDP: Session Description Protocol", RFC 4566, DOI 10.17487/RFC4566, July 2006, <<https://www.rfc-editor.org/info/rfc4566>>.
- [RFC4585] Ott, J., Wenger, S., Sato, N., Burmeister, C., and J. Rey, "Extended RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/AVPF)", RFC 4585, DOI 10.17487/RFC4585, July 2006, <<https://www.rfc-editor.org/info/rfc4585>>.
- [RFC4733] Schulzrinne, H. and T. Taylor, "RTP Payload for DTMF Digits, Telephony Tones, and Telephony Signals", RFC 4733, DOI 10.17487/RFC4733, December 2006, <<https://www.rfc-editor.org/info/rfc4733>>.
- [RFC4961] Wing, D., "Symmetric RTP / RTP Control Protocol (RTCP)", BCP 131, RFC 4961, DOI 10.17487/RFC4961, July 2007, <<https://www.rfc-editor.org/info/rfc4961>>.
- [RFC4967] Rosen, B., "Dial String Parameter for the Session Initiation Protocol Uniform Resource Identifier", RFC 4967, DOI 10.17487/RFC4967, July 2007, <<https://www.rfc-editor.org/info/rfc4967>>.

- [RFC5104] Wenger, S., Chandra, U., Westerlund, M., and B. Burman, "Codec Control Messages in the RTP Audio-Visual Profile with Feedback (AVPF)", RFC 5104, DOI 10.17487/RFC5104, February 2008, <<https://www.rfc-editor.org/info/rfc5104>>.
- [RFC5168] Levin, O., Even, R., and P. Hagendorf, "XML Schema for Media Control", RFC 5168, DOI 10.17487/RFC5168, March 2008, <<https://www.rfc-editor.org/info/rfc5168>>.
- [RFC5245] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, DOI 10.17487/RFC5245, April 2010, <<https://www.rfc-editor.org/info/rfc5245>>.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <<https://www.rfc-editor.org/info/rfc5246>>.
- [RFC5389] Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, "Session Traversal Utilities for NAT (STUN)", RFC 5389, DOI 10.17487/RFC5389, October 2008, <<https://www.rfc-editor.org/info/rfc5389>>.
- [RFC5393] Sparks, R., Ed., Lawrence, S., Hawrylyshen, A., and B. Campen, "Addressing an Amplification Vulnerability in Session Initiation Protocol (SIP) Forking Proxies", RFC 5393, DOI 10.17487/RFC5393, December 2008, <<https://www.rfc-editor.org/info/rfc5393>>.
- [RFC5626] Jennings, C., Ed., Mahy, R., Ed., and F. Audet, Ed., "Managing Client-Initiated Connections in the Session Initiation Protocol (SIP)", RFC 5626, DOI 10.17487/RFC5626, October 2009, <<https://www.rfc-editor.org/info/rfc5626>>.
- [RFC5658] Froment, T., Lebel, C., and B. Bonnaerens, "Addressing Record-Route Issues in the Session Initiation Protocol (SIP)", RFC 5658, DOI 10.17487/RFC5658, October 2009, <<https://www.rfc-editor.org/info/rfc5658>>.
- [RFC5763] Fischl, J., Tschofenig, H., and E. Rescorla, "Framework for Establishing a Secure Real-time Transport Protocol (SRTP) Security Context Using Datagram Transport Layer Security (DTLS)", RFC 5763, DOI 10.17487/RFC5763, May 2010, <<https://www.rfc-editor.org/info/rfc5763>>.

- [RFC5764] McGrew, D. and E. Rescorla, "Datagram Transport Layer Security (DTLS) Extension to Establish Keys for the Secure Real-time Transport Protocol (SRTP)", RFC 5764, DOI 10.17487/RFC5764, May 2010, <<https://www.rfc-editor.org/info/rfc5764>>.
- [RFC5766] Mahy, R., Matthews, P., and J. Rosenberg, "Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)", RFC 5766, DOI 10.17487/RFC5766, April 2010, <<https://www.rfc-editor.org/info/rfc5766>>.
- [RFC5954] Gurbani, V., Ed., Carpenter, B., Ed., and B. Tate, Ed., "Essential Correction for IPv6 ABNF and URI Comparison in RFC 3261", RFC 5954, DOI 10.17487/RFC5954, August 2010, <<https://www.rfc-editor.org/info/rfc5954>>.
- [RFC6184] Wang, Y., Even, R., Kristensen, T., and R. Jesup, "RTP Payload Format for H.264 Video", RFC 6184, DOI 10.17487/RFC6184, May 2011, <<https://www.rfc-editor.org/info/rfc6184>>.
- [RFC6263] Marjou, X. and A. Sollaud, "Application Mechanism for Keeping Alive the NAT Mappings Associated with RTP / RTP Control Protocol (RTCP) Flows", RFC 6263, DOI 10.17487/RFC6263, June 2011, <<https://www.rfc-editor.org/info/rfc6263>>.
- [RFC6351] Perreault, S., "xCard: vCard XML Representation", RFC 6351, DOI 10.17487/RFC6351, August 2011, <<https://www.rfc-editor.org/info/rfc6351>>.
- [RFC6352] Daboo, C., "CardDAV: vCard Extensions to Web Distributed Authoring and Versioning (WebDAV)", RFC 6352, DOI 10.17487/RFC6352, August 2011, <<https://www.rfc-editor.org/info/rfc6352>>.
- [RFC6442] Polk, J., Rosen, B., and J. Peterson, "Location Conveyance for the Session Initiation Protocol", RFC 6442, DOI 10.17487/RFC6442, December 2011, <<https://www.rfc-editor.org/info/rfc6442>>.
- [RFC6665] Roach, A., "SIP-Specific Event Notification", RFC 6665, DOI 10.17487/RFC6665, July 2012, <<https://www.rfc-editor.org/info/rfc6665>>.

- [RFC6764] Daboo, C., "Locating Services for Calendaring Extensions to WebDAV (CalDAV) and vCard Extensions to WebDAV (CardDAV)", RFC 6764, DOI 10.17487/RFC6764, February 2013, <<https://www.rfc-editor.org/info/rfc6764>>.
- [RFC6881] Rosen, B. and J. Polk, "Best Current Practice for Communications Services in Support of Emergency Calling", BCP 181, RFC 6881, DOI 10.17487/RFC6881, March 2013, <<https://www.rfc-editor.org/info/rfc6881>>.
- [RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <<https://www.rfc-editor.org/info/rfc7159>>.
- [RFC7525] Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 7525, DOI 10.17487/RFC7525, May 2015, <<https://www.rfc-editor.org/info/rfc7525>>.
- [RFC7647] Sparks, R. and A. Roach, "Clarifications for the Use of REFER with RFC 6665", RFC 7647, DOI 10.17487/RFC7647, September 2015, <<https://www.rfc-editor.org/info/rfc7647>>.
- [RFC7742] Roach, A., "WebRTC Video Processing and Codec Requirements", RFC 7742, DOI 10.17487/RFC7742, March 2016, <<https://www.rfc-editor.org/info/rfc7742>>.
- [RFC7874] Valin, JM. and C. Bran, "WebRTC Audio Codec and Processing Requirements", RFC 7874, DOI 10.17487/RFC7874, May 2016, <<https://www.rfc-editor.org/info/rfc7874>>.

## Authors' Addresses

Brian Rosen  
Mars, PA  
US

Phone: +1 724 382 1051  
Email: [br@brianrosen.net](mailto:br@brianrosen.net)

Jim Malloy  
The MITRE Corporation  
McLean, VA  
US

Phone: +1 703 983 2835  
Email: jmalloy@mitre.org

Brett Henderson  
The MITRE Corporation  
McLean, VA  
US

Phone: +1 619 758 6071  
Email: brhenderson@mitre.org

Network Working Group  
Internet-Draft  
Intended status: Informational  
Expires: 23 July 2020

A. Rundgren  
Independent  
B. Jordan  
Broadcom  
S. Erdtman  
Spotify AB  
20 January 2020

JSON Canonicalization Scheme (JCS)  
draft-rundgren-json-canonicalization-scheme-17

Abstract

Cryptographic operations like hashing and signing need the data to be expressed in an invariant format so that the operations are reliably repeatable. One way to address this is to create a canonical representation of the data. Canonicalization also permits data to be exchanged in its original form on the "wire" while cryptographic operations performed on the canonicalized counterpart of the data in the producer and consumer end points, generate consistent results.

This document describes the JSON Canonicalization Scheme (JCS). The JCS specification defines how to create a canonical representation of JSON data by building on the strict serialization methods for JSON primitives defined by ECMAScript, constraining JSON data to the I-JSON subset, and by using deterministic property sorting.

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 23 July 2020.

## Copyright Notice

Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	3
2. Terminology . . . . .	4
3. Detailed Operation . . . . .	4
3.1. Creation of Input Data . . . . .	4
3.2. Generation of Canonical JSON Data . . . . .	5
3.2.1. Whitespace . . . . .	5
3.2.2. Serialization of Primitive Data Types . . . . .	5
3.2.2.1. Serialization of Literals . . . . .	6
3.2.2.2. Serialization of Strings . . . . .	6
3.2.2.3. Serialization of Numbers . . . . .	7
3.2.3. Sorting of Object Properties . . . . .	7
3.2.4. UTF-8 Generation . . . . .	9
4. IANA Considerations . . . . .	9
5. Security Considerations . . . . .	9
6. Acknowledgements . . . . .	10
7. References . . . . .	10
7.1. Normative References . . . . .	10
7.2. Informative References . . . . .	11
Appendix A. ES6 Sample Canonicalizer . . . . .	12
Appendix B. Number Serialization Samples . . . . .	13
Appendix C. Canonicalized JSON as "Wire Format" . . . . .	15
Appendix D. Dealing with Big Numbers . . . . .	15
Appendix E. String Subtype Handling . . . . .	16
E.1. Subtypes in Arrays . . . . .	18
Appendix F. Implementation Guidelines . . . . .	18
Appendix G. Open Source Implementations . . . . .	19
Appendix H. Other JSON Canonicalization Efforts . . . . .	20
Appendix I. Development Portal . . . . .	20
Appendix J. Document History . . . . .	20
Authors' Addresses . . . . .	22

## 1. Introduction

This document describes the JSON Canonicalization Scheme (JCS). The JCS specification defines how to create a canonical representation of JSON [RFC8259] data by building on the strict serialization methods for JSON primitives defined by ECMAScript [ES6], constraining JSON data to the I-JSON [RFC7493] subset, and by using deterministic property sorting. The output from JCS is a "Hashable" representation of JSON data that can be used by cryptographic methods. The subsequent paragraphs outline the primary design considerations.

Cryptographic operations like hashing and signing need the data to be expressed in an invariant format so that the operations are reliably repeatable. One way to accomplish this is to convert the data into a format that has a simple and fixed representation, like Base64Url [RFC4648]. This is how JWS [RFC7515] addressed this issue. Another solution is to create a canonical version of the data, similar to what was done for the XML Signature [XMLDSIG] standard.

The primary advantage with a canonicalizing scheme is that data can be kept in its original form. This is the core rationale behind JCS. Put another way, using canonicalization enables a JSON Object to remain a JSON Object even after being signed. This can simplify system design, documentation, and logging.

To avoid "reinventing the wheel", JCS relies on the serialization of JSON primitives (strings, numbers and literals), as defined by ECMAScript (aka JavaScript) beginning with version 6 [ES6], hereafter referred to as "ES6".

Seasoned XML developers may recall difficulties getting XML signatures to validate. This was usually due to different interpretations of the quite intricate XML canonicalization rules as well as of the equally complex Web Services security standards. The reasons why JCS should not suffer from similar issues are:

- o The absence of a namespace concept and default values.
- o Constraining data to the I-JSON [RFC7493] subset. This eliminates the need for specific parsers for dealing with canonicalization.
- o JCS compatible serialization of JSON primitives is currently supported by most Web browsers as well as by Node.js [NODEJS],
- o The full JCS specification is currently supported by multiple Open Source implementations (see Appendix G). See also Appendix F for implementation guidelines.



JCS is compatible with some existing systems relying on JSON canonicalization such as JWK Thumbprint [RFC7638] and Keybase [KEYBASE].

For potential uses outside of cryptography see [JSONCOMP].

The intended audiences of this document are JSON tool vendors, as well as designers of JSON based cryptographic solutions. The reader is assumed to be knowledgeable in ECMAScript including the "JSON" object.

## 2. Terminology

Note that this document is not on the IETF standards track. However, a conformant implementation is supposed to adhere to the specified behavior for security and interoperability reasons. This text uses BCP 14 to describe that necessary behavior.

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.

## 3. Detailed Operation

This section describes the details related to creating a canonical JSON representation, and how they are addressed by JCS.

Appendix F describes the RECOMMENDED way of adding JCS support to existing JSON tools.

### 3.1. Creation of Input Data

Data to be canonically serialized is usually created by:

- o Parsing previously generated JSON data.
- o Programmatically creating data.

Irrespective of the method used, the data to be serialized MUST be adapted for I-JSON [RFC7493] formatting, which implies the following:

- o JSON Objects MUST NOT exhibit duplicate property names.
- o JSON String data MUST be expressible as Unicode [UNICODE].

- o JSON Number data MUST be expressible as IEEE-754 [IEEE754] double precision values. For applications needing higher precision or longer integers than offered by IEEE-754 double precision, it is RECOMMENDED to represent such numbers as JSON Strings, see Appendix D for details on how this can be performed in an interoperable and extensible way.

An additional constraint is that parsed JSON String data MUST NOT be altered during subsequent serializations. For more information see Appendix E.

Note: although the Unicode standard offers the possibility of rearranging certain character sequences, referred to as "Unicode Normalization" (<https://www.unicode.org/reports/tr15/>), JCS compliant string processing does not take this in consideration. That is, all components involved in a scheme depending on JCS, MUST preserve Unicode string data "as is".

### 3.2. Generation of Canonical JSON Data

The following subsections describe the steps required to create a canonical JSON representation of the data elaborated on in the previous section.

Appendix A shows sample code for an ES6 based canonicalizer, matching the JCS specification.

#### 3.2.1. Whitespace

Whitespace between JSON tokens MUST NOT be emitted.

#### 3.2.2. Serialization of Primitive Data Types

Assume a JSON object as follows is parsed:

```
{
  "numbers": [333333333.33333329, 1E30, 4.50,
              2e-3, 0.00000000000000000000000000000001],
  "string": "\u20ac$\u000F\u000a'\u0042\u0022\u005c\\\"/\"",
  "literals": [null, true, false]
}
```

If the parsed data is subsequently serialized using a serializer compliant with ES6's "JSON.stringify()", the result would (with a line wrap added for display purposes only), be rather divergent with respect to the original data:

```
{ "numbers": [3333333333.3333333, 1e+30, 4.5, 0.002, 1e-27], "string":
"$\u000f\nA'B\"\\\"\\\"/"," literals": [null, true, false] }
```

The reason for the difference between the parsed data and its serialized counterpart, is due to a wide tolerance on input data (as defined by JSON [RFC8259]), while output data (as defined by ES6), has a fixed representation. As can be seen in the example, numbers are subject to rounding as well.

The following subsections describe the serialization of primitive JSON data types according to JCS. This part is identical to that of ES6. In the (unlikely) event that a future version of ECMAScript would invalidate any of the following serialization methods, it will be up to the developer community to either stick to this specification or create a new specification.

#### 3.2.2.1. Serialization of Literals

In accordance with JSON [RFC8259], the literals "null", "true", and "false" MUST be serialized as null, true, and false respectively.

#### 3.2.2.2. Serialization of Strings

For JSON String data (which includes JSON Object property names as well), each Unicode code point MUST be serialized as described below (see section 24.3.2.2 of [ES6]):

- o If the Unicode value falls within the traditional ASCII control character range (U+0000 through U+001F), it MUST be serialized using lowercase hexadecimal Unicode notation (\uhhhh) unless it is in the set of predefined JSON control characters U+0008, U+0009, U+000A, U+000C or U+000D which MUST be serialized as \b, \t, \n, \f and \r respectively.
- o If the Unicode value is outside of the ASCII control character range, it MUST be serialized "as is" unless it is equivalent to U+005C (\) or U+0022 (") which MUST be serialized as \\ and \" respectively.

Finally, the resulting sequence of Unicode code points MUST be enclosed in double quotes (").

Note: since invalid Unicode data like "lone surrogates" (e.g. U+DEAD) may lead to interoperability issues including broken signatures, occurrences of such data MUST cause a compliant JCS implementation to terminate with an appropriate error.

### 3.2.2.3. Serialization of Numbers

ES6 builds on the IEEE-754 [IEEE754] double precision standard for representing JSON Number data. Such data MUST be serialized according to section 7.1.12.1 of [ES6] including the "Note 2" enhancement.

Due to the relative complexity of this part, the algorithm itself is not included in this document. For implementers of JCS compliant number serialization, Google's implementation in V8 [V8] may serve as a reference. Another compatible number serialization reference implementation is Ryu [RYU], that is used by the JCS open source Java implementation mentioned in Appendix G. Appendix B holds a set of IEEE-754 sample values and their corresponding JSON serialization.

Note: since "NaN" (Not a Number) and "Infinity" are not permitted in JSON, occurrences of "NaN" or "Infinity" MUST cause a compliant JCS implementation to terminate with an appropriate error.

### 3.2.3. Sorting of Object Properties

Although the previous step normalized the representation of primitive JSON data types, the result would not yet qualify as "canonical" since JSON Object properties are not in lexicographic (alphabetical) order.

Applied to the sample in Section 3.2.2, a properly canonicalized version should (with a line wrap added for display purposes only), read as:

```
{"literals":[null,true,false],"numbers":[333333333.3333333,
1e+30,4.5,0.002,1e-27],"string":"$\u000f\nA'B\"\\\"\\\"/\"}
```

The rules for lexicographic sorting of JSON Object properties according to JCS are as follows:

- o JSON Object properties MUST be sorted recursively, which means that JSON child Objects MUST have their properties sorted as well.
- o JSON Array data MUST also be scanned for the presence of JSON Objects (if an object is found then its properties MUST be sorted), but array element order MUST NOT be changed.

When a JSON Object is about to have its properties sorted, the following measures MUST be adhered to:

- o The sorting process is applied to property name strings in their

"raw" (unescaped) form. That is, a newline character is treated as U+000A.

- o Property name strings to be sorted are formatted as arrays of UTF-16 [UNICODE] code units. The sorting is based on pure value comparisons, where code units are treated as unsigned integers, independent of locale settings.
- o Property name strings either have different values at some index that is a valid index for both strings, or their lengths are different, or both. If they have different values at one or more index positions, let k be the smallest such index; then the string whose value at position k has the smaller value, as determined by using the < operator, lexicographically precedes the other string. If there is no index position at which they differ, then the shorter string lexicographically precedes the longer string.

In plain English this means that property names are sorted in ascending order like the following:

```
" "  
"a"  
"aa"  
"ab"
```

The rationale for basing the sorting algorithm on UTF-16 code units is that it maps directly to the string type in ECMAScript (featured in Web browsers and Node.js), Java and .NET. In addition, JSON only supports escape sequences expressed as UTF-16 code units making knowledge and handling of such data a necessity anyway. Systems using another internal representation of string data will need to convert JSON property name strings into arrays of UTF-16 code units before sorting. The conversion from UTF-8 or UTF-32 to UTF-16 is defined by the Unicode [UNICODE] standard.

The following test data can be used for verifying the correctness of the sorting scheme in a JCS implementation. JSON test data:

```
{  
  "\u20ac": "Euro Sign",  
  "\r": "Carriage Return",  
  "\ufb33": "Hebrew Letter Dalet With Dagesh",  
  "1": "One",  
  "\ud83d\ude00": "Emoji: Grinning Face",  
  "\u0080": "Control",  
  "\u00f6": "Latin Small Letter O With Diaeresis"  
}
```

Expected argument order after sorting property strings:

```
"Carriage Return"
"One"
"Control"
"Latin Small Letter O With Diaeresis"
"Euro Sign"
"Emoji: Grinning Face"
"Hebrew Letter Dalet With Dagesh"
```

Note: for the purpose of obtaining a deterministic property order, sorting on UTF-8 or UTF-32 encoded data would also work, but the outcome for JSON data like above would differ and thus be incompatible with this specification. However, in practice, property names are rarely defined outside of 7-bit ASCII making it possible to sort on string data in UTF-8 or UTF-32 format without conversions to UTF-16 and still be compatible with JCS. If this is a viable option or not depends on the environment JCS is used in.

#### 3.2.4. UTF-8 Generation

Finally, in order to create a platform independent representation, the result of the preceding step **MUST** be encoded in UTF-8.

Applied to the sample in Section 3.2.3 this should yield the following bytes, here shown in hexadecimal notation:

```
7b 22 6c 69 74 65 72 61 6c 73 22 3a 5b 6e 75 6c 6c 2c 74 72
75 65 2c 66 61 6c 73 65 5d 2c 22 6e 75 6d 62 65 72 73 22 3a
5b 33 33 33 33 33 33 33 33 33 2e 33 33 33 33 33 33 33 2c 31
65 2b 33 30 2c 34 2e 35 2c 30 2e 30 30 32 2c 31 65 2d 32 37
5d 2c 22 73 74 72 69 6e 67 22 3a 22 e2 82 ac 24 5c 75 30 30
30 66 5c 6e 41 27 42 5c 22 5c 5c 5c 5c 5c 22 2f 22 7d
```

This data is intended to be usable as input to cryptographic methods.

#### 4. IANA Considerations

This document has no IANA actions.

#### 5. Security Considerations

It is crucial to perform sanity checks on input data to avoid overflowing buffers and similar things that could affect the integrity of the system.

When JCS is applied to signature schemes like the one described in

Appendix F, applications MUST perform the following operations before acting upon received data:

1. Parse the JSON data and verify that it adheres to I-JSON.
2. Verify the data for correctness according to the conventions defined by the ecosystem where it is to be used. This also includes locating the property holding the signature data.
3. Verify the signature.

If any of these steps fail, the operation in progress MUST be aborted.

## 6. Acknowledgements

Building on ES6 Number serialization was originally proposed by James Manger. This ultimately led to the adoption of the entire ES6 serialization scheme for JSON primitives.

Other people who have contributed with valuable input to this specification include Scott Ananian, Tim Bray, Ben Campbell, Adrian Farrell, Richard Gibson, Bron Gondwana, John-Mark Gurney, John Levine, Mark Miller, Matthew Miller, Mike Jones, Mark Nottingham, Mike Samuel, Jim Schaad, Robert Tupelo-Schneck and Michal Wadas.

For carrying out real world concept verification, the software and support for number serialization provided by Ulf Adams, Tanner Gooding and Remy Oudompheng was very helpful.

## 7. References

### 7.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [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>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.

- [RFC7493] Bray, T., Ed., "The I-JSON Message Format", RFC 7493, DOI 10.17487/RFC7493, March 2015, <<https://www.rfc-editor.org/info/rfc7493>>.
- [ES6] Ecma International, "ECMAScript 2015 Language Specification", June 2015, <<https://www.ecma-international.org/ecma-262/6.0/index.html>>.
- [IEEE754] IEEE, "IEEE Standard for Floating-Point Arithmetic", August 2008, <<http://grouper.ieee.org/groups/754/>>.
- [UNICODE] The Unicode Consortium, "The Unicode Standard, Version 12.1.0", May 2019, <<https://www.unicode.org/versions/Unicode12.1.0/>>.

## 7.2. Informative References

- [RFC7638] Jones, M. and N. Sakimura, "JSON Web Key (JWK) Thumbprint", RFC 7638, DOI 10.17487/RFC7638, September 2015, <<https://www.rfc-editor.org/info/rfc7638>>.
- [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>>.
- [RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <<https://www.rfc-editor.org/info/rfc7515>>.
- [JSONCOMP] A. Rundgren, "'Comparable' JSON - Work in progress", <<https://tools.ietf.org/html/draft-rundgren-comparable-json-04>>.
- [V8] Google LLC, "Chrome V8 Open Source JavaScript Engine", <<https://developers.google.com/v8/>>.
- [RYU] Ulf Adams, "Ryu floating point number serializing algorithm", <<https://github.com/ulfjack/ryu>>.
- [NODEJS] "Node.js", <<https://nodejs.org>>.
- [KEYBASE] "Keybase", <[https://keybase.io/docs/api/1.0/canonical\\_packings#json](https://keybase.io/docs/api/1.0/canonical_packings#json)>.
- [OPENAPI] "The OpenAPI Initiative", <<https://www.openapis.org/>>.
- [XMLDSIG] W3C, "XML Signature Syntax and Processing Version 1.1", <<https://www.w3.org/TR/xmlsig-core1/>>.



## Appendix A. ES6 Sample Canonicalizer

Below is an example of a JCS canonicalizer for usage with ES6 based systems:

```

////////////////////////////////////
// Since the primary purpose of this code is highlighting //
// the core of the JCS algorithm, error handling and      //
// UTF-8 generation were not implemented                  //
////////////////////////////////////
var canonicalize = function(object) {

    var buffer = '';
    serialize(object);
    return buffer;

    function serialize(object) {
        if (object === null || typeof object !== 'object' ||
            object.toJSON !== null) {
            ///////////////////////////////////
            // Primitive type or toJSON - Use ES6/JSON      //
            ///////////////////////////////////
            buffer += JSON.stringify(object);

        } else if (Array.isArray(object)) {
            ///////////////////////////////////
            // Array - Maintain element order                //
            ///////////////////////////////////
            buffer += '[';
            let next = false;
            object.forEach((element) => {
                if (next) {
                    buffer += ',';
                }
                next = true;
                ///////////////////////////////////
                // Array element - Recursive expansion //
                ///////////////////////////////////
                serialize(element);
            });
            buffer += ']';

        } else {
            ///////////////////////////////////
            // Object - Sort properties before serializing //
            ///////////////////////////////////
            buffer += '{';
            let next = false;

```

```

Object.keys(object).sort().forEach((property) => {
  if (next) {
    buffer += ',';
  }
  next = true;
  // Property names are strings - Use ES6/JSON //
  buffer += JSON.stringify(property);
  buffer += ':';
  // Property value - Recursive expansion //
  serialize(object[property]);
});
buffer += '}'
}
};

```

## Appendix B. Number Serialization Samples

The following table holds a set of ES6 compatible Number serialization samples, including some edge cases. The column "IEEE-754" refers to the internal ES6 representation of the Number data type which is based on the IEEE-754 [IEEE754] standard using 64-bit (double precision) values, here expressed in hexadecimal.

IEEE-754	JSON Representation	Comment
0000000000000000	0	Zero
8000000000000000	0	Minus zero
0000000000000001	5e-324	Min pos number
8000000000000001	-5e-324	Min neg number
7fefffffffffffffff	1.7976931348623157e+308	Max pos number
ffefffffffffffffffff	-1.7976931348623157e+308	Max neg number
4340000000000000	9007199254740992	Max pos int (1)
c340000000000000	-9007199254740992	Max neg int (1)
4430000000000000	295147905179352830000	~2**68 (2)

7fffffffffffffffffff				NaN	(3)
7ff0000000000000				Infinity	(3)
44b52d02c7e14af5		9.999999999999997e+22			
44b52d02c7e14af6		1e+23			
44b52d02c7e14af7		1.0000000000000001e+23			
444b1ae4d6e2ef4e		999999999999999700000			
444b1ae4d6e2ef4f		999999999999999900000			
444b1ae4d6e2ef50		1e+21			
3eb0c6f7a0b5ed8c		9.999999999999997e-7			
3eb0c6f7a0b5ed8d		0.000001			
41b3de4355555553		333333333.3333332			
41b3de4355555554		333333333.33333325			
41b3de4355555555		333333333.3333333			
41b3de4355555556		333333333.3333334			
41b3de4355555557		333333333.33333343			
becbf647612f3696		-0.0000033333333333333333			
43143ff3c1cb0959		1424953923781206.2		Round to even	(4)

## Notes:

- (1) For maximum compliance with the ES6 "JSON" object, values that are to be interpreted as true integers SHOULD be in the range -9007199254740991 to 9007199254740991. However, how numbers are used in applications do not affect the JCS algorithm.
- (2) Although a set of specific integers like  $2^{68}$  could be regarded as having extended precision, the JCS/ES6 number serialization algorithm does not take this in consideration.
- (3) Value out range, not permitted in JSON. See Section 3.2.2.3.

- (4) This number is exactly 1424953923781206.25 but will after the "Note 2" rule mentioned in Section 3.2.2.3 be truncated and rounded to the closest even value.

For a more exhaustive validation of a JCS number serializer, you may test against a file (currently) available in the development portal (see Appendix I), containing a large set of sample values. Another option is running V8 [V8] as a live reference together with a program generating a substantial amount of random IEEE-754 values.

#### Appendix C. Canonicalized JSON as "Wire Format"

Since the result from the canonicalization process (see Section 3.2.4), is fully valid JSON, it can also be used as "Wire Format". However, this is just an option since cryptographic schemes based on JCS, in most cases would not depend on that externally supplied JSON data already is canonicalized.

In fact, the ES6 standard way of serializing objects using "JSON.stringify()" produces a more "logical" format, where properties are kept in the order they were created or received. The example below shows an address record which could benefit from ES6 standard serialization:

```
{
  "name": "John Doe",
  "address": "2000 Sunset Boulevard",
  "city": "Los Angeles",
  "zip": "90001",
  "state": "CA"
}
```

Using canonicalization the properties above would be output in the order "address", "city", "name", "state" and "zip", which adds fuzziness to the data from a human (developer or technical support), perspective. Canonicalization also converts JSON data into a single line of text, which may be less than ideal for debugging and logging.

#### Appendix D. Dealing with Big Numbers

There are several issues associated with the JSON Number type, here illustrated by the following sample object:

```
{
  "giantNumber": 1.4e+9999,
  "payMeThis": 26000.33,
  "int64Max": 9223372036854775807
}
```

Although the sample above conforms to JSON [RFC8259], applications would normally use different native data types for storing "giantNumber" and "int64Max". In addition, monetary data like "payMeThis" would presumably not rely on floating point data types due to rounding issues with respect to decimal arithmetic.

The established way handling this kind of "overloading" of the JSON Number type (at least in an extensible manner), is through mapping mechanisms, instructing parsers what to do with different properties based on their name. However, this greatly limits the value of using the JSON Number type outside of its original somewhat constrained, JavaScript context. The ES6 "JSON" object does not support mappings to JSON Number either.

Due to the above, numbers that do not have a natural place in the current JSON ecosystem MUST be wrapped using the JSON String type. This is close to a de-facto standard for open systems. This is also applicable for other data types that do not have direct support in JSON, like "DateTime" objects as described in Appendix E.

Aided by a system using the JSON String type; be it programmatic like

```
var obj = JSON.parse('{ "giantNumber": "1.4e+9999" }');
var biggie = new BigNumber(obj.giantNumber);
```

or declarative schemes like OpenAPI [OPENAPI], JCS imposes no limits on applications, including when using ES6.

#### Appendix E. String Subtype Handling

Due to the limited set of data types featured in JSON, the JSON String type is commonly used for holding subtypes. This can depending on JSON parsing method lead to interoperability problems which MUST be dealt with by JCS compliant applications targeting a wider audience.

Assume you want to parse a JSON object where the schema designer assigned the property "big" for holding a "BigInt" subtype and "time" for holding a "DateTime" subtype, while "val" is supposed to be a JSON Number compliant with JCS. The following example shows such an object:

```
{
  "time": "2019-01-28T07:45:10Z",
  "big": "055",
  "val": 3.5
}
```

Parsing of this object can be accomplished by the following ES6 statement:

```
var object = JSON.parse(JSON_object_featured_as_a_string);
```

After parsing the actual data can be extracted which for subtypes also involve a conversion step using the result of the parsing process (an ECMAScript object) as input:

```
... = new Date(object.time); // Date object
... = BigInt(object.big);    // Big integer
... = object.val;            // JSON/JS number
```

Note that the "BigInt" data type is currently only natively supported by V8 [V8].

Canonicalization of "object" using the sample code in Appendix A would return the following string:

```
{"big":"055","time":"2019-01-28T07:45:10Z","val":3.5}
```

Although this is (with respect to JCS) technically correct, there is another way parsing JSON data which also can be used with ECMAScript as shown below:

```
// "BigInt" requires the following code to become JSON serializable
BigInt.prototype.toJSON = function() {
  return this.toString();
};

// JSON parsing using a "stream" based method
var object = JSON.parse(JSON_object_featured_as_a_string,
  (k,v) => k == 'time' ? new Date(v) : k == 'big' ? BigInt(v) : v
);
```

If you now apply the canonicalizer in Appendix A to "object", the following string would be generated:

```
{"big":"55","time":"2019-01-28T07:45:10.000Z","val":3.5}
```

In this case the string arguments for "big" and "time" have changed with respect to the original, presumably making an application depending on JCS fail.

The reason for the deviation is that in stream and schema based JSON parsers, the original "string" argument is typically replaced on-the-fly by the native subtype which when serialized, may exhibit a different and platform dependent pattern.

That is, stream and schema based parsing MUST treat subtypes as "pure" (immutable) JSON String types, and perform the actual conversion to the designated native type in a subsequent step. In modern programming platforms like Go, Java and C# this can be achieved with moderate efforts by combining annotations, getters and setters. Below is an example in C#/Json.NET showing a part of a class that is serializable as a JSON Object:

```
// The "pure" string solution uses a local
// string variable for JSON serialization while
// exposing another type to the application
[JsonProperty("amount")]
private string _amount;

[JsonIgnore]
public decimal Amount {
    get { return decimal.Parse(_amount); }
    set { _amount = value.ToString(); }
}
```

In an application "Amount" can be accessed as any other property while it is actually represented by a quoted string in JSON contexts.

Note: the example above also addresses the constraints on numeric data implied by I-JSON (the C# "decimal" data type has quite different characteristics compared to IEEE-754 double precision).

#### E.1. Subtypes in Arrays

Since the JSON Array construct permits mixing arbitrary JSON data types, custom parsing and serialization code may be required to cope with subtypes anyway.

#### Appendix F. Implementation Guidelines

The optimal solution is integrating support for JCS directly in JSON serializers (parsers need no changes). That is, canonicalization would just be an additional "mode" for a JSON serializer. However, this is currently not the case. Fortunately, JCS support can be introduced through externally supplied canonicalizer software acting as a post processor to existing JSON serializers. This arrangement also relieves the JCS implementer from having to deal with how underlying data is to be represented in JSON.

The post processor concept enables signature creation schemes like the following:

1. Create the data to be signed.

2. Serialize the data using existing JSON tools.
3. Let the external canonicalizer process the serialized data and return canonicalized result data.
4. Sign the canonicalized data.
5. Add the resulting signature value to the original JSON data through a designated signature property.
6. Serialize the completed (now signed) JSON object using existing JSON tools.

A compatible signature verification scheme would then be as follows:

1. Parse the signed JSON data using existing JSON tools.
2. Read and save the signature value from the designated signature property.
3. Remove the signature property from the parsed JSON object.
4. Serialize the remaining JSON data using existing JSON tools.
5. Let the external canonicalizer process the serialized data and return canonicalized result data.
6. Verify that the canonicalized data matches the saved signature value using the algorithm and key used for creating the signature.

A canonicalizer like above is effectively only a "filter", potentially usable with a multitude of quite different cryptographic schemes.

Using a JSON serializer with integrated JCS support, the serialization performed before the canonicalization step could be eliminated for both processes.

#### Appendix G. Open Source Implementations

The following Open Source implementations have been verified to be compatible with JCS:

- \* JavaScript: <https://www.npmjs.com/package/canonicalize>
- \* Java: <https://github.com/erdtman/java-json-canonicalization>



- \* Go: <https://github.com/cyberphone/json-canonicalization/tree/master/go>
- \* .NET/C#: <https://github.com/cyberphone/json-canonicalization/tree/master/dotnet>
- \* Python: <https://github.com/cyberphone/json-canonicalization/tree/master/python3>

#### Appendix H. Other JSON Canonicalization Efforts

There are (and have been) other efforts creating "Canonical JSON". Below is a list of URLs to some of them:

- \* <https://tools.ietf.org/html/draft-staykov-hu-json-canonical-form-00>
- \* <https://gibson042.github.io/canonicaljson-spec/>
- \* [http://wiki.laptop.org/go/Canonical\\_JSON](http://wiki.laptop.org/go/Canonical_JSON)

The listed efforts all build on text level JSON to JSON transformations. The primary feature of text level canonicalization is that it can be made neutral to the flavor of JSON used. However, such schemes also imply major changes to the JSON parsing process which is a likely hurdle for adoption. Albeit at the expense of certain JSON and application constraints, JCS was designed to be compatible with existing JSON tools.

#### Appendix I. Development Portal

The JCS specification is currently developed at:  
<https://github.com/cyberphone/ietf-json-canon>.

JCS source code and extensive test data is available at:  
<https://github.com/cyberphone/json-canonicalization>

#### Appendix J. Document History

[[ This section to be removed by the RFC Editor before publication as an RFC ]]

Version 00-06:

- \* See IETF diff listings.

Version 07:

- \* Initial conversion to XML RFC version 3.
- \* Changed intended status to "Informational".
- \* Added UTF-16 test data and explanations.

Version 08:

- \* Updated Abstract.
- \* Added a "Note 2" number serialization sample.
- \* Updated Security Considerations.
- \* Tried to clear up the JSON input data section.
- \* Added a line about Unicode normalization.
- \* Added a line about serialiation of structured data.
- \* Added a missing fact about "BigInt" (V8 not ES6).

Version 09:

- \* Updated initial line of Abstract and Introduction.
- \* Added note about breaking ECMAScript changes.
- \* Minor language nit fixes.

Version 10-12:

- \* Language tweaks.

Version 13:

- \* Reorganized Section 3.2.2.3.

Version 14:

- \* Improved introduction + some minor changes in security considerations, aknowlegdgements, and unicode normalization.
- \* Generalized data representation issues by updating Appendix F.

Version 15:

- \* Minor nits, reverted the IEEE-754 table to ASCII.

- \* Added a bit more meat to the IEEE-754 table.
- \* Changed all <artwork> to: type="ascii-art" and removed name="".

Version 16:

- \* Updated section 2 according to AD's wish.

Version 17:

- \* Updated section 2 after IESG input.
- \* Author affiliation update.

#### Authors' Addresses

Anders Rundgren  
Independent  
Montpellier  
France

Email: anders.rundgren.net@gmail.com  
URI: <https://www.linkedin.com/in/andersrundgren/>

Bret Jordan  
Broadcom  
1320 Ridder Park Drive  
San Jose, CA 95131  
United States of America

Email: [bret.jordan@broadcom.com](mailto:bret.jordan@broadcom.com)

Samuel Erdtman  
Spotify AB  
Birger Jarlsgatan 61, 4tr  
SE-113 56 Stockholm  
Sweden

Email: [erdtman@spotify.com](mailto:erdtman@spotify.com)

TBD  
Internet-Draft  
Intended status: Standards Track  
Expires: July 5, 2020

R. Stepanek  
FastMail  
M. Loffredo  
IIT-CNR  
January 2, 2020

JSContact: A JSON representation of contact data  
draft-stepanek-jscontact-06

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 July 5, 2020.

Copyright Notice

Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must

include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	3
1.1. Relation to the xCard and jCard formats . . . . .	3
1.2. Terminology . . . . .	4
1.3. Vendor-specific Property Extensions and Values . . . . .	4
2. JSCard . . . . .	4
2.1. Metadata properties . . . . .	4
2.1.1. uid . . . . .	4
2.1.2. prodId . . . . .	4
2.1.3. updated . . . . .	5
2.1.4. kind . . . . .	5
2.1.5. relatedTo . . . . .	5
2.2. Name and Organization properties . . . . .	6
2.2.1. fullName . . . . .	6
2.2.2. name . . . . .	6
2.2.3. organization . . . . .	6
2.2.4. jobTitle . . . . .	7
2.2.5. role . . . . .	7
2.3. Contact and Resource properties . . . . .	7
2.3.1. emails . . . . .	7
2.3.2. phones . . . . .	7
2.3.3. online . . . . .	8
2.3.4. preferredContactMethod . . . . .	8
2.3.5. preferredContactLanguages . . . . .	8
2.4. Address and Location properties . . . . .	9
2.4.1. addresses . . . . .	9
2.5. Additional properties . . . . .	10
2.5.1. anniversaries . . . . .	10
2.5.2. personalInfo . . . . .	11
2.5.3. notes . . . . .	11
2.5.4. categories . . . . .	12
2.6. Common JSCard types . . . . .	12
2.6.1. LocalizedString . . . . .	12
2.6.2. Resource . . . . .	12
3. JSCardGroup . . . . .	12
3.1. Properties . . . . .	13
3.1.1. uid . . . . .	13
3.1.2. name . . . . .	13
3.1.3. cards . . . . .	13
4. Implementation Status . . . . .	13
4.1. IIT-CNR/Registro.it . . . . .	14
5. IANA Considerations . . . . .	14
6. Security Considerations . . . . .	14

7. References	14
7.1. Normative References	14
7.2. Informative References	15
7.3. URIs	16
Authors' Addresses	16

## 1. Introduction

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

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

- o Most of the initial set of attributes should be taken from the vCard data format [RFC6350] and extensions ([RFC6473], [RFC6474], [RFC6715], [RFC6869], [RFC8605]). The specification should add new attributes or value types, or not support existing ones, where appropriate. Conversion between the data formats need not fully preserve semantic meaning.
- o The attributes of the cards data represented must be described as a simple key-value pair, reducing complexity of its representation.
- o The data model should avoid all ambiguities and make it difficult to make mistakes during implementation.
- o 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 JSCard easier to adopt, and the availability of production-ready JSON implementations eliminates a whole category of parser-related interoperability issues.

### 1.1. Relation to the xCard and jCard formats

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

## 1.2. Terminology

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

## 1.3. Vendor-specific Property Extensions and Values

Vendors MAY add additional properties to JSContact objects to support their custom features. The names of these properties MUST be prefixed with a domain name controlled by the vendor to avoid conflict, e.g. "example.com/customprop".

Some JSContact properties allow vendor-specific value extensions. If so, vendor-specific values MUST be prefixed with a domain name controlled by the vendor, e.g. "example.com/customrel".

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

## 2. JSCard

MIME type: "application/jscontact+json;type=jscard"

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

### 2.1. Metadata properties

#### 2.1.1. uid

Type: "String" (mandatory).

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

#### 2.1.2. prodId

Type: "String" (optional).

The identifier for the product that created the JSCard object.

### 2.1.3. updated

Type: "String" (mandatory).

The date and time when the data in this JSCard object was last modified. The timestamp MUST be formatted as specified in [RFC3339].

### 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:

- o "individual": a single person
- o "org": an organization
- o "location": a named location
- o "device": a device, such as appliances, computers, or network elements
- o "application": a software application

### 2.1.5. relatedTo

Type: "String[Relation]" (optional).

Relates the object to other JSCard objects. This is represented as a map of the URI (or single text value) of the related objects to a possibly empty set of relation types. The Relation object has the following properties:

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

Note, the Relation object only has one property; it is specified as an object with a single property to allow for extension in the future.



## 2.2. Name and Organization properties

### 2.2.1. fullName

Type: "LocalizedString" (optional).

The full name (e.g. the personal name and surname of an individual, the name of an organization) of the entity represented by this card.

### 2.2.2. name

Type: "NameComponent[]" (optional).

The name components of the name of the entity represented by this JSCard. Name components SHOULD be ordered such that their values joined by whitespace produce a valid full name of this entity.

A NameComponent has the following properties:

- o value: "String" (mandatory). The value of this name component.
- o type: "String" (mandatory). The type of this name component. Valid name component types are:
  - \* "prefix". The value is a honorific title(s), e.g. "Mr", "Ms", "Dr".
  - \* "personal". The value is a personal name(s), also known as "first name", "given name".
  - \* "surname". The value is a surname, also known as "last name", "family name".
  - \* "additional". The value is an additional name, also known as "middle name".
  - \* "suffix". The value is a honorific suffix, e.g. "B.A.", "Esq".
  - \* "nickname". The value is a nickname.

### 2.2.3. organization

Type: "LocalizedString[]" (optional).

The company or organization name and units associated with this card. The first entry in the list names the organization, and any following entries name organizational units.

#### 2.2.4. jobTitle

Type : "LocalizedString[]" (optional).

The job title(s) or functional position(s) of the entity represented by this card.

#### 2.2.5. role

Type : "LocalizedString[]" (optional).

The role(s), function(s) or part(s) played in a particular situation by the entity represented by this card. In contrast to a job title, the roles might differ for example in project contexts.

### 2.3. Contact and Resource properties

#### 2.3.1. emails

Type: "Resource[]" (optional).

An array of Resource objects where the values are URLs in the [RFC6068] "mailto" scheme or free-text email addresses. Types are:

- o "personal" The address is for emailing in a personal context.
- o "work" The address is for emailing in a professional context.
- o "other" The address is for some other purpose. A label property MAY be included to display next to the address to help the user identify its purpose.

#### 2.3.2. phones

Type: "Resource[]" (optional).

An array of Resource objects where the values are URIs scheme or free-text phone numbers. Typical URI schemes are the [RFC3966] "tel" or [RFC3261] "sip" schemes, but any URI scheme is allowed. Resource types are:

- o "voice" The number is for calling by voice.
- o "fax" The number is for sending faxes.
- o "pager" The number is for a pager or beeper.

- o "other" The number is for some other purpose. A label property MAY be included to display next to the number to help the user identify its purpose.

The following labels are pre-defined for phone resources:

- o "private" The phone number should be used in a private context.
- o "work" The phone number should be used in a professional context

#### 2.3.3. online

Type: "Resource[]" (optional).

An array of Resource objects where the values are URIs or usernames associated with the card for online services. Types are:

- o "uri" The value is a URI, e.g. a website link.
- o "username" The value is a username associated with the entity represented by this card (e.g. for social media, or an IM client). A label property SHOULD be included to identify what service this is for. For compatibility between clients, this label SHOULD be the canonical service name, including capitalisation. e.g. "Twitter", "Facebook", "Skype", "GitHub", "XMPP".
- o "other" The value is something else not covered by the above categories. A label property MAY be included to display next to the number to help the user identify its purpose.

#### 2.3.4. preferredContactMethod

Type : "String" (optional)

Defines the preferred contact method or resource with additional information about this card. The value MUST be the property name of one of the Resource lists: "emails", "phones", "online", "other".

#### 2.3.5. preferredContactLanguages

Type : "String[ContactLanguage[]]" (optional)

Defines the preferred languages for contacting the entity associated with this card. The keys in the object MUST be [RFC5646] language tags. The values are a (possibly empty) list of contact language preferences for this language. Also see the definition of the VCARD LANG property (Section 6.4.4., [RFC6350]).

A ContactLanguage object has the following properties:

- o type: "String" (optional). Defines the context of this preference. This could be "work", "home" or another value.
- o preference: "Number" (optional). Defines the preference order of this language for the context defined in the type property. If set, the property value MUST be between 1 and 100 (inclusive). Lower values correspond to a higher level of preference, with 1 being most preferred. If not set, the default MUST be to interpret the language as the least preferred in its context. Preference orders SHOULD be unique across language for a specific type.

A valid ContactLanguage object MUST have at least one of its properties set.

## 2.4. Address and Location properties

### 2.4.1. addresses

Type: Address[] (optional).

An array of Address objects, containing physical locations. An Address object has the following properties:

- o type: "String" (optional, default "other"). Specifies the context of the address information. The value MUST be either one of the following values, registered in a future RFC, or a vendor-specific value:
  - \* "home" An address of a residence.
  - \* "work" An address of a workplace.
  - \* "billing" An address to be used for billing.
  - \* "postal" An address to be used for delivering physical items.
  - \* "other" An address not covered by the above categories.
- o label: "String" (optional). A label describing the value in more detail.
- o fullAddress: "LocalizedString" (optional). The complete address, excluding type and label. This property is mainly useful to represent addresses of which the individual address components are unknown, or to provide localized representations.

- o street: "String" (optional). The street address. This MAY be multiple lines; newlines MUST be preserved.
- o extension: "String" (optional) The extended address, such as an apartment or suite number, or care-of address.
- o locality: "String" (optional). The city, town, village, post town, or other locality within which the street address may be found.
- o region: "String" (optional). The province, such as a state, county, or canton within which the locality may be found.
- o country: "String" (optional). The country name.
- o postOfficeBox: "String" (optional) The post office box.
- o 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.
- o countryCode: "String" (optional). The ISO-3166-1 country code.
- o coordinates: "String" (optional) A [RFC5870] "geo:" URI for the address.
- o timeZone: "String" (optional) Identifies the time zone this address is located in. This SHOULD be a time zone name registered in the IANA Time Zone Database [1]. Unknown time zone identifiers MAY be ignored by implementations.
- o isPreferred: Boolean (optional, default: false). Whether this Address is the preferred for its type. This SHOULD only be one per type.

## 2.5. Additional properties

### 2.5.1. anniversaries

Type : Anniversary[] (optional).

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

- o type: "String" (mandatory). Specifies the type of the anniversary. This RFC predefines the following types, but implementations MAY use additional values:

- \* "birth": a birth day anniversary
- \* "death": a death day anniversary
- \* "other": an anniversary not covered by any of the known types.
- o label: "String" (optional). A label describing the value in more detail, especially if the type property has value "other" (but MAY be included with any type).
- o date: "String" (mandatory). The date of this anniversary, in the form "YYYY-MM-DD" (any part may be all 0s for unknown) or a [RFC3339] timestamp.
- o place: Address (optional). An address associated with this anniversary, e.g. the place of birth or death.

#### 2.5.2. personalInfo

Type: PersonalInformation[] (optional).

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

- o type: "String" (mandatory). Specifies the type for this personal information. Allowed values are:
  - \* "expertise": a field of expertise or credential
  - \* "hobby": a hobby
  - \* "interest": an interest
  - \* "other": an information not covered by the above categories
- o value: "String" (mandatory). The actual information. This generally is free-text, but future specifications MAY restrict allowed values depending on the type of this PersonalInformation.
- o level: "String" (optional) Indicates the level of expertise, or engagement in hobby or interest. Allowed values are: "high", "medium" and "low".

#### 2.5.3. notes

Type: "LocalizedString[]" (optional).

Arbitrary notes about the entity represented by this card.

#### 2.5.4. categories

Type: "String[]" (optional). A list of free-text or URI categories that relate to the card.

### 2.6. Common JSCard types

#### 2.6.1. LocalizedString

A LocalizedString object has the following properties:

- o value: "String" (mandatory). The property value.
- o language: "String" (optional). The [RFC5646] language tag of this value, if any.
- o localizations: "String[String]" (optional). A map from [RFC5646] language tags to the value localized in that language.

#### 2.6.2. Resource

A Resource object has the following properties:

- o type: "String" (mandatory). Specifies the context of the resource. This MUST be taken from the set of values specified for the respective property.
- o label: "String" (optional). A label describing the value in more detail, especially if the type property has value "other" (but MAY be included with any type).
- o value: "String" (mandatory). The actual resource value, e.g. an email address or phone number.
- o mediaType: "String" (optional). Used for properties with URI values. Provides the media type [RFC2046] of the resource identified by the URI.
- o isPreferred: Boolean (optional, default: false). Whether this resource is the preferred for its type. This SHOULD only be one per type.

### 3. JSCardGroup

MIME type: "application/jscontact+json;type=jscardgroup"

A JSCardGroup object represents a named set of JSCards.

### 3.1. Properties

#### 3.1.1. uid

Type : "String" (mandatory).

A globally unique identifier. The same requirements as for the JSCard uid property apply.

#### 3.1.2. name

Type: "String" (optional).

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

#### 3.1.3. cards

Type : "JSCard[]" (mandatory). The cards in the group. Implementations MUST preserve the order of list entries.

### 4. Implementation Status

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



#### 4.1. IIT-CNR/Registro.it

- o Responsible Organization: Institute of Informatics and Telematics of National Research Council (IIT-CNR)/Registro.it
- o Location: <https://rdap.pubtest.nic.it/> [2]
- o Description: This implementation includes support for RDAP queries using data from the public test environment of .it ccTLD. The RDAP server does not implement any security policy because data returned by this server are only for experimental testing purposes. The RDAP server returns responses including JSCard in place of jCard when queries contain the parameter jscard=1.
- o Level of Maturity: This is a "proof of concept" research implementation.
- o Coverage: This implementation includes all of the features described in this specification.
- o Contact Information: Mario Loffredo, [mario.loffredo@iit.cnr.it](mailto:mario.loffredo@iit.cnr.it)

#### 5. IANA Considerations

TBD

#### 6. Security Considerations

TBD

#### 7. References

##### 7.1. Normative References

- [RFC2046] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, DOI 10.17487/RFC2046, November 1996, <<https://www.rfc-editor.org/info/rfc2046>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, DOI 10.17487/RFC4122, July 2005, <<https://www.rfc-editor.org/info/rfc4122>>.

- [RFC5646] Phillips, A., Ed. and M. Davis, Ed., "Tags for Identifying Languages", BCP 47, RFC 5646, DOI 10.17487/RFC5646, September 2009, <<https://www.rfc-editor.org/info/rfc5646>>.
- [RFC5870] Mayrhofer, A. and C. Spanring, "A Uniform Resource Identifier for Geographic Locations ('geo' URI)", RFC 5870, DOI 10.17487/RFC5870, June 2010, <<https://www.rfc-editor.org/info/rfc5870>>.
- [RFC6350] Perreault, S., "vCard Format Specification", RFC 6350, DOI 10.17487/RFC6350, August 2011, <<https://www.rfc-editor.org/info/rfc6350>>.
- [RFC6351] Perreault, S., "xCard: vCard XML Representation", RFC 6351, DOI 10.17487/RFC6351, August 2011, <<https://www.rfc-editor.org/info/rfc6351>>.
- [RFC7095] Kewisch, P., "jCard: The JSON Format for vCard", RFC 7095, DOI 10.17487/RFC7095, January 2014, <<https://www.rfc-editor.org/info/rfc7095>>.
- [RFC7493] Bray, T., Ed., "The I-JSON Message Format", RFC 7493, DOI 10.17487/RFC7493, March 2015, <<https://www.rfc-editor.org/info/rfc7493>>.
- [RFC7942] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", BCP 205, RFC 7942, DOI 10.17487/RFC7942, July 2016, <<https://www.rfc-editor.org/info/rfc7942>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <<https://www.rfc-editor.org/info/rfc8259>>.

## 7.2. Informative References

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

- [RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002, <<https://www.rfc-editor.org/info/rfc3339>>.
- [RFC3966] Schulzrinne, H., "The tel URI for Telephone Numbers", RFC 3966, DOI 10.17487/RFC3966, December 2004, <<https://www.rfc-editor.org/info/rfc3966>>.
- [RFC6068] Duerst, M., Masinter, L., and J. Zawinski, "The 'mailto' URI Scheme", RFC 6068, DOI 10.17487/RFC6068, October 2010, <<https://www.rfc-editor.org/info/rfc6068>>.
- [RFC6473] Saint-Andre, P., "vCard KIND:application", RFC 6473, DOI 10.17487/RFC6473, December 2011, <<https://www.rfc-editor.org/info/rfc6473>>.
- [RFC6474] Li, K. and B. Leiba, "vCard Format Extensions: Place of Birth, Place and Date of Death", RFC 6474, DOI 10.17487/RFC6474, December 2011, <<https://www.rfc-editor.org/info/rfc6474>>.
- [RFC6715] Cauchie, D., Leiba, B., and K. Li, "vCard Format Extensions: Representing vCard Extensions Defined by the Open Mobile Alliance (OMA) Converged Address Book (CAB) Group", RFC 6715, DOI 10.17487/RFC6715, August 2012, <<https://www.rfc-editor.org/info/rfc6715>>.
- [RFC6869] Salgueiro, G., Clarke, J., and P. Saint-Andre, "vCard KIND:device", RFC 6869, DOI 10.17487/RFC6869, February 2013, <<https://www.rfc-editor.org/info/rfc6869>>.
- [RFC8605] Hollenbeck, S. and R. Carney, "vCard Format Extensions: ICANN Extensions for the Registration Data Access Protocol (RDAP)", RFC 8605, DOI 10.17487/RFC8605, May 2019, <<https://www.rfc-editor.org/info/rfc8605>>.

### 7.3. URIs

[1] <https://www.iana.org/time-zones>

[2] <https://rdap.pubtest.nic.it/>

### Authors' Addresses

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

Email: [rsto@fastmailteam.com](mailto:rsto@fastmailteam.com)

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

Email: [mario.loffredo@iit.cnr.it](mailto:mario.loffredo@iit.cnr.it)