Network Working Group
Internet-Draft
Intended status: Informational
Expires: February 14, 2019

A. Ivanov
D. Spence
S. Saxena
T. Nadeau
Brocade

August 13, 2018

Application of YANG Modeling to JSON RPCs for Interoperability Purposes

draft-yang-json-rpc-03

Abstract

This document specifies the application of YANG modeling language to JSON RPC 2.0 for the purposes of achieving interoperability between implementations.

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 14, 2019.

Copyright Notice

Table of Contents

1. Introduction

A key use case for JSON encoding of data is the transfer and interchange of such data between applications. While some of the semantics desribed here will be of value for any such interchange, this document will concentrate on two specific use cases - Remote Procedure Calls (RPCs) and Notifications. There is a standard specification for using JSON in both - it is the JSON RPC 2.0 specification maintained by JSON RPC working group at jsonrpc.org [JSONRPC20]. This specification uses JSON as described in [RFC7159] in messages transported over a variety of transports. For example - HTTP [RFC7230], 0MQ [ZEROMQ], AMQP [AMQP] etc.

The JSON RPC 2.0 specification [JSONRPC20] has a number of well known shortcomings. There are no means of describing and documenting a particular API method. There are no means to specify how to interpret data received via an RPC call or notification and what data structures should be generated to accommodate the data. Most implementers have concentrated on the loss of data structure information as a key deficiency, while ignoring the lack of API description which is the actual root cause. This has led them to encode information about the data structures into bespoke extensions

also known as class hints and/or use of special "magic" method notations as implemented in [JABSORB], [PJ0MQ], etc. These break the core assumption of JSON being a language agnostic interchange format and result in a plethora of subtly incompatible JSON RPC implementations.

This document proposes an alternative to class hints and other non-interoperable extensions through the use of "JSON Encoding of Data Modeled with YANG" RFC [RFC7951] and YANG modeling of JSON RPCs.

2. Terminology and Notation

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

YANG is the data modeling language as described in [RFC7950], version 1.1

JSON encoding of YANG modeled data structures is described in [RFC7951]

JSON RPC 2.0 is the Remote Procedure Call and Notifications encoding as described in [JSONRPC20]

3. Modeling of JSON RPC 2.0 Interfaces in YANG

All RPCs in a particular API must be described as RPC statements in its corresponding YANG model in accordance to the rules of RFC7950 [RFC7950], section 7.14, which is expanded to cover JSON RPCs in addition to NETCONF RPCs. The RPC arguments and results must comply to subsections 7.14.2 and 7.14.3 of RFC7950 [RFC7950].

All Notifications provided by a particular API must be described as notification statements in its corresponding YANG model in accordance to the rules of RFC7950 [RFC7950], section 7.15 which is expanded to cover JSON Notifications in addition to NETCONF ones. The JSON Notification payload and substatements must comply to subsection 7.15.1 of RFC7950 [RFC7950].

While all applicable YANG statements as specified by RFC7950 [RFC7950] are supported and acceptable in both RPCs and notifications, the use of must, when, leafref and identityref is discouraged to improve interoperability with implementations which use simplistic serialization/deserialization to parse the messages and do not have a fully featured YANG interpreter. The internal mapping of an RPC name to a function call is an implementation detail outside the scope of this document. RPC method names containing

characters invalid as a part of a function name in most computing languages are discouraged, but not prohibited.

All mappings from application data types to JSON representation types MUST use the conventions defined in "JSON Encoding of Data Modeled with YANG" [RFC7951]. The JSON message payload must use the I-JSON profile according to [RFC7493].

If an application does not wish to specify the model constraints for a particular RPC argument or Notification payload it must specify the corresponding element as anydata in the YANG model.

JSON RPC Specification [JSONRPC20] mandates that positional arguments are always represented as a list even if an RPC call or Notification has a single argument.

For example, if an RPC is modelled as:

```
grouping arg-uri {
        leaf uri {
                description "A test URI.";
                mandatory true;
                type inet:uri;
        }
}
rpc test-uri {
        description "A simple example RPC.";
        input {
                uses arg-uri;
        }
}
```

Figure 1

The following payload is invalid - it violates the JSON RPC Specification [JSONRPC20]:

```
{
        "jsonrpc": "2.0",
        "id": 3,
        "method": "test-uri",
        "params": "http://www.ietf.org"
}
```

Figure 2

The correct parameter encoding is as follows:

```
{
        "jsonrpc": "2.0",
        "id": 3,
        "method": "test-uri",
        "params": [
                "http://www.ietf.org"
        ]
}
```

Figure 3

or

```
{
        "jsonrpc": "2.0",
        "id": 3,
        "method": "test-uri",
        "params": {
                "uri": "http://www.ietf.org"
        }
}
```

Figure 4

YANG models describing different RPCs and Notifications may be grouped together into a YANG module to describe an API or a set of APIs.

3.1. Optionality

Omitted optional parameters for the positional form of JSON RPC 2.0 [JSONRPC20] must be specified as null elements in the argument array.

For example, both arguments in the test-elements RPC are optional.

```
rpc test-elements {
        description "A simple example RPC.";
        input {
                leaf element1 {
                        type string;
                }
                leaf element2 {
                        type string;
                }
        }
}
```

Figure 5

If only element2 is supplied, the resulting JSON RPC payload using the positional form should be:

```
{
    "id": 3,
    "jsonrpc": "2.0",
    "method": "test-elements",
    "params": [
        null,
        "element2 value"
    ]
}
```

Figure 6

The named form should be:

```
{
        "jsonrpc":"2.0",
        "id":3,
        "method":"test-elements",
        "params":{
                "element2": "element2 value"
        }
}
```

Figure 7

While using the named form can allow an implementation to differentiate between a parameter being supplied and a parameter being null, implementers should not rely on this difference in semantics as most computing language compilers and runtimes will obscure this difference from the caller.

Trailing nulls resulting from missing parameters or an implementation supplying a null argument MAY be stripped from a call by position payload. The following encoding:

```
{
        "jsonrpc": "2.0",
        "id": 3,
        "method": "test-elements",
        "params": [
                "element1 value",
                null
        ]
}
```

Figure 8

can also be expressed as:

```
{
        "jsonrpc": "2.0",
        "id": 3,
        "method": "test-elements",
        "params": [
                "element1 value"
        ]
}
```

Figure 9

3.2. Default values

The receiving implementation MUST supply any default values as specified by the model if the sender has omitted them as described in RFC7950 [RFC7950], section 7.14.2 and 7.14.3. Thus, in a JSON RPC the caller may omit supplying a default value. The callee (the server) implementation is obliged to fill it in prior to passing the data to an application. For example, for the following model:

```
rpc test-htg-2 {
        description "A simple example RPC.";
        input {
                leaf question {
                        type string;
                        default "Meaning of the Universe";
                }
        }
        output {
                leaf answer {
                        type int;
                        default 42;
                }
}
```

Figure 10

The test-htg RPC can be invoked with the following payload:

```
{
        "jsonrpc": "2.0",
        "id": 3,
        "method": "test-htg-2",
        "params": []
}
```

Figure 11

The receiver implementation MUST fill the value of "Meaning of the Universe" before passing the parsed payload to the application. The application is not obliged to supply the answer of 42. If the application does not supply an answer, the model aware JSON RPC2.0 implementation must add it to the payload prior to it being sent to the caller as an RPC result.

3.3. The idl:value-type YANG Extension

This document standardizes a new YANG extension idl:value-type compliant to [RFC7950]. This extension specifies additional metadata necessary for a source code generator to produce the correct source code mapping for a specific YANG type. The value provided by the idl:value-type extension is a type hint for the generator and may refine or override the standard type mapping rules specified in "JSON Encoding of Data Modeled with YANG" [RFC7951]. For example:

```
idl:value-type python {
        idl:implemented-by bsct.enforce.value.TypeBoolean;
}
```

Figure 12

3.4. Modeling of JSON RPC 2.0 RPCs

3.4.1. Representing results

In order to simplify integration to existing codebase, RPC calls returning a single result MUST emit a single value for the params key in the RPC message instead of a an array consisting of a one element for the positional form of JSON RPC 2.0 for the cases where it is non-ambiguous, namely lists, leaf-lists and all primitive types.

For example, invoking the test-1 RPC in the following example

```
rpc test-1 {
        description "A simple example RPC.";
        output {
                leaf answer {
                        type int;
                }
}
```

Figure 13

will produce a simplified result:

```
{
    "id": 3,
    "jsonrpc": "2.0",
    "result": 42
}
```

Figure 14

This simplification is ambiguous for anydata and container result types. If an RPC call uses the positional form to represent a container or anydata return value, it MUST represent it as an array with a single element equal to the result value.

For example, invoking the test-2 RPC in the following example

```
rpc test-2 {
        description "A simple example RPC.";
        output {
                leaf answer {
                        type anydata;
                }
}
```

Figure 15

will produce a verbose result:

```
{
    "id": 3,
    "jsonrpc": "2.0",
    "result": [
                {"key":"value"}
        ]
}
```

Figure 16

3.4.2. Modeling and expressing in JSON of data structures fully described by model

All data structures for which the model is known should be fully described in YANG as specified in [RFC7950]. The test-uri RPC in the following example is expecting a uri argument. The uri argument is fully described and modeled:

```
grouping arg-uri {
        leaf uri {
                description "A test URI.";
                mandatory true;
                type inet:uri;
        }
}
rpc test-uri {
        description "A simple example RPC.";
        input {
                uses arg-uri;
        }
        output {
                leaf passes {
                        type boolean;
                }
        }
}
```

Figure 17

The output is a boolean signifying if the test has passed or has failed. For a URI value of "http://www.ietf.org", this corresponds to the following JSON RPC request payload when using positional arguments:

```
{
        "jsonrpc": "2.0",
        "id": 3,
        "method": "test-uri",
        "params": ["http://www.ietf.org"]
}
```

Figure 18

Alternatively, when using named arguments

```
{
        "jsonrpc": "2.0",
        "id": 3,
        "method": "test-uri",
        "params": {
                "uri": "http://www.ietf.org"
        }
}
```

Figure 19

If we assume that the remote procedure call with an argument of "http://www.ietf.org" returns True, we will have the following result. Positional form:

```
{
        "jsonrpc": "2.0",
        "id": 3,
        "result": true
}
```

Figure 20

Named form:

```
{
        "jsonrpc": "2.0",
        "id": 3,
        "result": {
                "passes": true
        }
}
```

Figure 21

3.4.3. Modeling and expressing in JSON of data structures with elements outside model scope

Structures which the implementer does not wish to model in YANG (from here on referred to as "opaque") are specified as anydata. For example the test-object RPC expects an object argument and returns an object output. The argument and the return are opaque to the model and may contain nested structures or structures which the implementer has decided to keep outside the scope of the model.

```
grouping arg-object {
        leaf object {
                description "A test Object.";
                mandatory true;
                type anydata;
        }
}
rpc test-object {
        description "A simple example RPC.";
        input {
                uses arg-object;
        }
        output {
                uses arg-object;
        }
}
```

Figure 22

Objects opaque to the model as in this example are encoded as JSON using the rules for anydata in [RFC7951]

For an object value of {"key":"value"}, this corresponds to the following JSON RPC request payload when using positional arguments:

```
{
        "jsonrpc": "2.0",
        "id": 3,
        "method": "test-object",
        "params":[
                {"key": "value"}
        ]
}
```

Figure 23

Alternatively, when using named arguments

```
{
        "jsonrpc": "2.0",
        "id": 3,
        "method": "test-object",
        "params": {
                "object":{
                        "key": "value"
                }
        }
}
```

Figure 24

If we assume that the procedure call returns the following test-object ["eeny", "meeny", "miny", "moe"], we will have the following result. Positional form:

```
{
        "jsonrpc": "2.0",
        "id": 3,
        "result": [
                "eeny",
                "meeny",
                "miny",
                "moe"
        ]
}
```

Figure 25

Named form:

```
{
        "jsonrpc": "2.0",
        "id": 3,
        "result": {
                "object": [
                        "eeny",
                        "meeny",
                        "miny",
                        "moe"
                ]
        }
}
```

Figure 26

3.5. Modeling of JSON RPC 2.0 Notifications

3.5.1. Modeling and expressing in JSON of notifications with fully modeled data payload

All data structures for which the model is known should be fully described in YANG as specified in [RFC7950]. The testing Notification in the following example contains a uri leaf. The notification server issuing the notification is expected to supply the value and the recipients will expect the value for this leaf:

```
grouping arg-uri {
        leaf uri {
                description "A test URI.";
                mandatory true;
                type inet:uri;
        }
}
notification notify-uri {
        description "A simple notification with URI payload.";
        uses arg-uri;
}
```

Figure 27

This results in the following notification payloads. Positional arguments:

```
{
        "jsonrpc": "2.0",
        "method": "notify-uri",
        "params": [
                "http://www.ietf.org"
        ]
}
```

Figure 28

Note - there is no id member in a notification. Alternatively, when using named arguments

```
{
        "jsonrpc":"2.0",
        "method": "notify-uri",
        "params": {
                "uri": "http://www.ietf.org"
        }
}
```

Figure 29

3.5.2. Modeling and expressing in JSON of notifications with data payload outside model scope

Structures which the implementer does not wish to model in YANG and pass as object are specified as anydata. For example the notify-object notification expects an object argument as its object leaf.

```
grouping arg-object {
        leaf object {
                description "A test Object.";
                mandatory true;
                type anydata;
        }
}
notification notify-object {
        description "A simple Notification with an
                opaque object payload.";
        uses arg-object;
}
```

Figure 30

Objects opaque to the model as in this example are encoded as JSON using the rules for anydata in [RFC7951] resulting in the following example payloads for the {"key":"value"} object payload:

```
{
        "jsonrpc": "2.0",
        "method": "notify-object",
        "params": [
                {
                        "key": "value"
                }
        ]
}
```

Figure 31

Note - there is no id member in a notification. Alternatively, when using named arguments

```
{
        "jsonrpc": "2.0",
        "method": "notify-object",
        "params": {
                "object": {
                        "key": "value"
                }
        }
}
```

Figure 32

3.6. Addressing Modeled Data - Yang Paths

When working with structured data in a tree form it is essential to be able to address parts of the tree and individual elements. Yang uses Instance Identifiers for this purpose. The current normative reference for this is section 6.1. of [RFC7951]. It borrows the representation from Netconf and represents IIds as an uri path. There are issues with this encoding when used in a JSON RPC context:

- Yang IId is intended only for the purposes of addressing data. It does not provide semantics to address RPCs and/or Notifications and their argument which are the primary target of this specification.

- The representation in section 6.11 of [RFC7951] does not match JSON semantics - it uses URI syntax which requires conversion for each addressing operation. This approach has some advantages for implementations which support netconf and/or XML in parallel with JSON as these have existing URI parsers. It is, however, clearly disadvantageous for any non-netconf implementation, because it introduces a single "foreign" object type with a non-JSON serialization in the middle of a JSON specification.

The path specification specified in this draft provides a representation of paths to address Notification and RPC objects as well as paths into data structures. It MUST be explicitly specified as an anydata object in any models and is not a 1:1 replacement representation for an IId.

3.6.1. Addressing anything but an individual list element

Containers, leaves, leaf-lists, lists as a whole (everything but an individual element in a list) have their Path expressed as:

```
{
        "module:top-level-container": {
                "subcontainer":{}
        }
}
```

Figure 33

The path is represented as a single branch tree structure containing nested JSON objects. Each level key is the QName. Path terminates in {} to signify what is being addressed.

Module, revision and namespace qualifiers are optional and can be omitted except module qualifier at top level.

3.6.2. Addressing an individual list element

Individual list elements have their Paths expressed as:

```
{
        "module:top-level-container": {
                "list":[{
                        "key1":"value1",
                        "key2":"value2"
                }]
        }
}
```

Figure 34

Path is represented as a single branch tree structure containing nested JSON objects up to the list level.

At list level the QName is followed by [] to signify a list. The list contains a single object with key-value pairs uniquely identifying the list element.

3.6.3. Addressing inside an individual list element

Individual list elements have their Paths expressed as:

```
{
        "module:top-level-container": {
                "list":[{
                        "key1":"value1",
                        "key2":"value2",
                        "list-element": {}
                }]
        }
}
```

Figure 35

This form is obtained by adding a path as described in Section 3.6.1 section to the list element IId representation described in Section 3.6.2.

4. Normative References

[AMQP] "AMQP Protocol Specification", <https://www.rabbitmq.com/resources/specs/amqp0-9-1.pdf>.

[JABSORB] "Jabsorb JSON RPC Orb and Broker", <http://www.jabsorb.org/>.

[JSONRPC20] Morley, M., "JSON-RPC 2.0 Specification", 2010, <http://www.jsonrpc.org/specification>.

[PJ0MQ] "Python JSON RPC over 0MQ", <https://github.com/dwb/jsonrpc2-zeromq-python>.

[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>.

[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>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <https://www.rfc-editor.org/info/rfc7230>.

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

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <https://www.rfc-editor.org/info/rfc7950>.

[RFC7951] Lhotka, L., "JSON Encoding of Data Modeled with YANG", RFC 7951, DOI 10.17487/RFC7951, August 2016, <https://www.rfc-editor.org/info/rfc7951>.

[ZEROMQ] "Zero MQ", <http://zeromq.org>.

Authors' Addresses

Anton Ivanov
Cambridge Greys Limited

Email: anton.ivanov@cambridgegreys.com

David Spence
Inocybe Technologies

Email: david@roughsketch.co.uk

Shaleen Saxena
Lumina Networks Inc.

Email: shaleen.external@gmail.com

Tom Nadeau
Lucid Vision LLC

Email: tnadeau@lucidvision.com