JSON-RPC 2.0 Specification

Origin Date:
2010-03-26 (based on the 2009-05-24 version)

Updated:
2013-01-04

Author:
JSON-RPC Working Group <json-rpc@googlegroups.com>

Table of Contents

1 Overview

JSON-RPC is a stateless, light-weight remote procedure call (RPC) protocol. Primarily this specification defines several data structures and the rules around their processing. It is transport agnostic in that the concepts can be used within the same process, over sockets, over http, or in many various message passing environments. It uses JSON (RFC 4627) as data format.

It is designed to be simple!

2 Conventions

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 RFC 2119.

Since JSON-RPC utilizes JSON, it has the same type system (see http://www.json.org or RFC 4627). JSON can represent four primitive types (Strings, Numbers, Booleans, and Null) and two structured types (Objects and Arrays). The term "Primitive" in this specification references any of those four primitive JSON types. The term "Structured" references either of the structured JSON types. Whenever this document refers to any JSON type, the first letter is always capitalized: Object, Array, String, Number, Boolean, Null. True and False are also capitalized.

All member names exchanged between the Client and the Server that are considered for matching of any kind should be considered to be case-sensitive. The terms function, method, and procedure can be assumed to be interchangeable.

The Client is defined as the origin of Request objects and the handler of Response objects.
The Server is defined as the origin of Response objects and the handler of Request objects.

One implementation of this specification could easily fill both of those roles, even at the same time, to other different clients or the same client. This specification does not address that layer of complexity.

3 Compatibility

JSON-RPC 2.0 Request objects and Response objects may not work with existing JSON-RPC 1.0 clients or servers. However, it is easy to distinguish between the two versions as 2.0 always has a member named "jsonrpc" with a String value of "2.0" whereas 1.0 does not. Most 2.0 implementations should consider trying to handle 1.0 objects, even if not the peer-to-peer and class hinting aspects of 1.0.

4 Request object

A rpc call is represented by sending a Request object to a Server. The Request object has the following members:

jsonrpc
A String specifying the version of the JSON-RPC protocol. MUST be exactly "2.0".

method
A String containing the name of the method to be invoked. Method names that begin with the word rpc followed by a period character (U+002E or ASCII 46) are reserved for rpc-internal methods and extensions and MUST NOT be used for anything else.

params
A Structured value that holds the parameter values to be used during the invocation of the method. This member MAY be omitted.

id
An identifier established by the Client that MUST contain a String, Number, or NULL value if included. If it is not included it is assumed to be a notification. The value SHOULD normally not be Null [1] and Numbers SHOULD NOT contain fractional parts [2]

The Server MUST reply with the same value in the Response object if included. This member is used to correlate the context between the two objects.

[1] The use of Null as a value for the id member in a Request object is discouraged, because this specification uses a value of Null for Responses with an unknown id. Also, because JSON-RPC 1.0 uses an id value of Null for Notifications this could cause confusion in handling.

[2] Fractional parts may be problematic, since many decimal fractions cannot be represented exactly as binary fractions.

4.1 Notification

A Notification is a Request object without an "id" member. A Request object that is a Notification signifies the Client's lack of interest in the corresponding Response object, and as such no Response object needs to be returned to the client. The Server MUST NOT reply to a Notification, including those that are within a batch request.

Notifications are not confirmable by definition, since they do not have a Response object to be returned. As such, the Client would not be aware of any errors (like e.g. "Invalid params","Internal error").

4.2 Parameter Structures

If present, parameters for the rpc call MUST be provided as a Structured value. Either by-position through an Array or by-name through an Object.

- by-position: params MUST be an Array, containing the values in the Server expected order.
- by-name: params MUST be an Object, with member names that match the Server expected parameter names. The absence of expected names MAY result in an error being generated. The names MUST match exactly, including case, to the method's expected parameters.

5 Response object

When a rpc call is made, the Server MUST reply with a Response, except for in the case of Notifications. The Response is expressed as a single JSON Object, with the following members:

jsonrpc
A String specifying the version of the JSON-RPC protocol. MUST be exactly "2.0".

result
This member is REQUIRED on success.
This member MUST NOT exist if there was an error invoking the method.
The value of this member is determined by the method invoked on the Server.

error
This member is REQUIRED on error.
This member MUST NOT exist if there was no error triggered during invocation.
The value for this member MUST be an Object as defined in section 5.1.

id
This member is REQUIRED.
It MUST be the same as the value of the id member in the Request Object.
If there was an error in detecting the id in the Request object (e.g. Parse error/Invalid Request), it MUST be Null.

Either the result member or error member MUST be included, but both members MUST NOT be included.

5.1 Error object

When a rpc call encounters an error, the Response Object MUST contain the error member with a value that is a Object with the following members:

code
A Number that indicates the error type that occurred.
This MUST be an integer.

message
A String providing a short description of the error.
The message SHOULD be limited to a concise single sentence.

data
A Primitive or Structured value that contains additional information about the error.
This may be omitted.
The value of this member is defined by the Server (e.g. detailed error information, nested errors etc.).

The error codes from and including -32768 to -32000 are reserved for pre-defined errors. Any code within this range, but not defined explicitly below is reserved for future use. The error codes are nearly the same as those suggested for XML-RPC at the following url: http://xmlrpc-epi.sourceforge.net/specs/rfc.fault_codes.php

code	message	meaning
-32700	Parse error	Invalid JSON was received by the server. An error occurred on the server while parsing the JSON text.
-32600	Invalid Request	The JSON sent is not a valid Request object.
-32601	Method not found	The method does not exist / is not available.
-32602	Invalid params	Invalid method parameter(s).
-32603	Internal error	Internal JSON-RPC error.
-32000 to -32099	Server error	Reserved for implementation-defined server-errors.

The remainder of the space is available for application defined errors.

6 Batch

To send several Request objects at the same time, the Client MAY send an Array filled with Request objects.

The Server should respond with an Array containing the corresponding Response objects, after all of the batch Request objects have been processed. A Response object SHOULD exist for each Request object, except that there SHOULD NOT be any Response objects for notifications. The Server MAY process a batch rpc call as a set of concurrent tasks, processing them in any order and with any width of parallelism.

The Response objects being returned from a batch call MAY be returned in any order within the Array. The Client SHOULD match contexts between the set of Request objects and the resulting set of Response objects based on the id member within each Object.

If the batch rpc call itself fails to be recognized as an valid JSON or as an Array with at least one value, the response from the Server MUST be a single Response object. If there are no Response objects contained within the Response array as it is to be sent to the client, the server MUST NOT return an empty Array and should return nothing at all.

7 Examples

Syntax:

```
--> data sent to Server
<-- data sent to Client
```

rpc call with positional parameters:

```
--> {"jsonrpc": "2.0", "method": "subtract", "params": [42, 23], "id": 1}
<-- {"jsonrpc": "2.0", "result": 19, "id": 1}
```

```
--> {"jsonrpc": "2.0", "method": "subtract", "params": [23, 42], "id": 2}
<-- {"jsonrpc": "2.0", "result": -19, "id": 2}
```

rpc call with named parameters:

```
--> {"jsonrpc": "2.0", "method": "subtract", "params": {"subtrahend": 23, "minuend": 42},
"id": 3}
<-- {"jsonrpc": "2.0", "result": 19, "id": 3}

--> {"jsonrpc": "2.0", "method": "subtract", "params": {"minuend": 42, "subtrahend": 23},
"id": 4}
<-- {"jsonrpc": "2.0", "result": 19, "id": 4}
```

a Notification:

```
--> {"jsonrpc": "2.0", "method": "update", "params": [1,2,3,4,5]}
--> {"jsonrpc": "2.0", "method": "foobar"}
```

rpc call of non-existent method:

```
--> {"jsonrpc": "2.0", "method": "foobar", "id": "1"}
<-- {"jsonrpc": "2.0", "error": {"code": -32601, "message": "Method not found"}, "id": "1"}
```

rpc call with invalid JSON:

```
--> {"jsonrpc": "2.0", "method": "foobar, "params": "bar", "baz]
<-- {"jsonrpc": "2.0", "error": {"code": -32700, "message": "Parse error"}, "id": null}
```

rpc call with invalid Request object:

```
--> {"jsonrpc": "2.0", "method": 1, "params": "bar"}
<-- {"jsonrpc": "2.0", "error": {"code": -32600, "message": "Invalid Request"}, "id": null}
```

rpc call Batch, invalid JSON:

```
--> [
  {"jsonrpc": "2.0", "method": "sum", "params": [1,2,4], "id": "1"},
  {"jsonrpc": "2.0", "method"
]
<-- {"jsonrpc": "2.0", "error": {"code": -32700, "message": "Parse error"}, "id": null}
```

rpc call with an empty Array:

```
--> []
<-- {"jsonrpc": "2.0", "error": {"code": -32600, "message": "Invalid Request"}, "id": null}
```

rpc call with an invalid Batch (but not empty):

```
--> [1]
<-- [
  {"jsonrpc": "2.0", "error": {"code": -32600, "message": "Invalid Request"}, "id": null}
]
```

rpc call with invalid Batch:

```
--> [1,2,3]
<-- [
  {"jsonrpc": "2.0", "error": {"code": -32600, "message": "Invalid Request"}, "id": null},
```

```
  {"jsonrpc": "2.0", "error": {"code": -32600, "message": "Invalid Request"}, "id": null},
  {"jsonrpc": "2.0", "error": {"code": -32600, "message": "Invalid Request"}, "id": null}
]
```

rpc call Batch:

```
--> [
        {"jsonrpc": "2.0", "method": "sum", "params": [1,2,4], "id": "1"},
        {"jsonrpc": "2.0", "method": "notify_hello", "params": [7]},
        {"jsonrpc": "2.0", "method": "subtract", "params": [42,23], "id": "2"},
        {"foo": "boo"},
        {"jsonrpc": "2.0", "method": "foo.get", "params": {"name": "myself"}, "id": "5"},
        {"jsonrpc": "2.0", "method": "get_data", "id": "9"}
    ]
<-- [
        {"jsonrpc": "2.0", "result": 7, "id": "1"},
        {"jsonrpc": "2.0", "result": 19, "id": "2"},
        {"jsonrpc": "2.0", "error": {"code": -32600, "message": "Invalid Request"}, "id": n
ull},
        {"jsonrpc": "2.0", "error": {"code": -32601, "message": "Method not found"}, "id":
 "5"},
        {"jsonrpc": "2.0", "result": ["hello", 5], "id": "9"}
    ]
```

rpc call Batch (all notifications):

```
--> [
        {"jsonrpc": "2.0", "method": "notify_sum", "params": [1,2,4]},
        {"jsonrpc": "2.0", "method": "notify_hello", "params": [7]}
    ]
<-- //Nothing is returned for all notification batches
```

8 Extensions

Method names that begin with rpc. are reserved for system extensions, and MUST NOT be used for anything else. Each system extension is defined in a related specification. All system extensions are OPTIONAL.

Site by Matt Morley of MPCM Technologies LLC, a manager of the JSON-RPC google group.