JSON-RPC-M1-Specification

Specification of protocol JSON-RPC M1

Setting	Value
Field	Information Technology
Subject	Protocol
Name	JSON-RPC M1
Document Type	Specification
Revision	1 (One)
Origin Date	2024-01-02 *(Second of January, year 2024)*
Author	Vault Thirteen (https://github.com/vault-thirteen)
Changes	None
Last Edit Date	None

Table of contents

1. Overview

JSON-RPC M1 is a remote procedure call protocol.

It is a modification of the original *JSON-RPC 2.0* protocol which can be found on its website – https://www.jsonrpc.org.

JSON-RPC M1 remote procedure call protocol is stateless. It is light-weight and very simple. It is even more simple than the original *JSON-RPC 2.0* protocol. As many people know, the simpler the thing is – the more reliable it is.

JSON-RPC M1 remote procedure call protocol is based on *JavaScript Object Notation* also known as *JSON* which is described in RFC 4627 and on its website located at http://www.json.org.

For additional robustness, *JSON-RPC M1* protocol uses more strict typing than in the original *JSON* Specification.

2. Conventions

JSON-RPC M1 protocol specification, further referred to as the ***M1 Specification***, describes the process of communication between a server and a client. One or more clients make requests to a server and a server must serve all the requests of each client. To serve a client's request, the server reads client's request, performs an action stated in the request and responds to the client. This means that process of communication consists of requests being sent from a client to a server and responses being sent from a server to a client. Requests and responses are described below.

Requests and responses must be encoded with the same text encoding. Text encoding of requests and responses is not limited by this specification, but it is advised to be the *UTF-8* encoding of the *Unicode* standard while it is one of the most popular encodings in the World Wide Web.

3. Request

Each request is a textual message encoded in a standard *JSON* format.
Request is a *JSON* object which must have exactly four (4) mandatory fields:

- `jsonrpc`
- `id`
- `method`
- `params`

These four fields of a client request and their meanings are described below.

Field	Type	Meaning	Example
jsonrpc	String	A *JSON* string containing protocol version. A typical version is `M1`.	`"jsonrpc": "M1"`
id	String	A *JSON* string identifying the request.	`"id": "12345"`
method	String	A *JSON* string containing a name of the requested function.	`"method": "setXY"`
params	Object	A *JSON* object containing named arguments for the requested function.	`"params": {"x": 6, "y": 9}`

Notes on Request

3.1. Mandatory Fields

Being a mandatory field means that *JSON* `Null` value is not accepted as a field value.

3.2. Protocol Version

A typical protocol version is `M1`.

If a client request contains an invalid or unsupported protocol version, an error must be returned.

3.3. Request Identifier

As opposed to the original *JSON-RPC 2.0* protocol, requests made with *M1 Specification* require the `id` field to be set. This also means that *M1 Specification* does not support one-side messages of the *JSON-RPC 2.0* protocol. All client requests of the *M1 Specification* must have a corresponding server response, *JSON-RPC 2.0* notifications are not supported.

It is a good practice to use unique values for request identifiers where possible.

If a client request contains an invalid request identifier, an error must be returned.

3.4. Method Name

Method name must contain only allowed characters. The list of allowed characters consists of the following symbols:

- small and capital latin letters of the *ASCII* standard
  - small letters from `a` to `z`
  - capital letters from `A` to `Z`
- *ASCII* numbers
  - numbers from `0` to `9`
- *ASCII* `Low Line` symbol, also known as underscore.

If a client request contains either invalid or non-existent function name, an error must be returned.

3.5. Function Parameters

If the called function or procedure does not accept any parameters, an empty *JSON* object must be provided as `params` value (`"params": {}`).

If a client request contains function parameters different from parameters of the called function, an error must be returned. For example, if a function has a single parameter named `x` and client provides two parameters named `x` and `y` , the server must respond with an error stating the unsupported parameter `y` .

If a client does not provide a required function parameter `x` , the server, if possible, should respond with an error stating the missing `x` parameter which was not provided. Unfortunately, a lot of libraries for parsing *JSON* format do not provide this functionality, that is why this rule says *SHOULD* instead of *MUST*.

Passing function parameters as an array instead of an object is not allowed by the *M1 Specification*. If an array must be passed to a function, it should be placed into a named argument, i.e. into a named field of an object.

4. Response

Each client request must be followed by a server response.
Each response is a textual message encoded in a standard *JSON* format.
Response is a *JSON* object which must have at least five (5) following fields:

- `jsonrpc`
- `id`

- `result`
- `error`
- `ok`

These five required fields of a server response and their meanings are described below.

Field	Type	Meaning	Example
jsonrpc	String	A *JSON* string containing protocol version. A typical version is `M1`.	`"jsonrpc": "M1"`
id	String	A *JSON* string identifying the request and response to it.	`"id": "12345"`
result	Object	A *JSON* object containing the result returned by the requested function.	`"result": {"answer": 42}`
error	Object	A *JSON* object containing error returned by the requested function.	`"error": {...}`
ok	Boolean	A *JSON* boolean indicating the successful execution of the requested function.	`"ok": true`

Notes on Response

4.1. Response Fields

All the above-mentioned five response fields must be set explicitly, even if they contain the *JSON* `Null` value.

4.2. Protocol Version

Protocol version in a response must be the same as in a normal request. Server must respond only with a valid protocol version even if the protocol version in the client request is wrong. If a client request contains an invalid or unsupported protocol version, an error must be returned.

4.3. Request & Response Identifier

Request identifier in response must be the same as in a normal request to which a server is responding. If the server can not obtain the request identifier, it must respond with an error and a request identifier set to *JSON* `Null`.

4.4. Result

The result returned by the called function or procedure must be a *JSON* object with named fields being the result values returned by the called function or procedure. An example – `"result":`

`{"x": 1, "y": 2}`. If the called function or procedure does not return anything, the server must respond with an empty *JSON* object (`"result": {}`). If execution of the requested function or procedure fails or ends with an error, server must set the result to *JSON* `Null` (`"result": null`).

4.5. Error

The error returned by the called function or procedure must be a *JSON* object with three following fields.

Field	Type	Meaning	Example
code	Integer	A *JSON* integer number with a numeric error code.	`"code": 101`
message	String	A *JSON* string with a textual description of the error.	`"message": "oops"`
data	Value	A *JSON* value providing error details.	`"data": ...`

Notes on Error

4.5.1. Error Code

Error code must be an integer number. Positive error code numbers may be used for errors defined by user functions. Negative error code numbers are used by errors defined by an RPC server. Zero number is never used as an error code. Error code is a required field when an error is set.

4.5.2. Error Message

Error message may be anything in textual format. Error message is a required field when an error is set.

4.5.3. Error Data

Error data is an optional *JSON* value providing additional details about the error. If no details are available, it must be *JSON* `Null` (`"data": null`).

4.5.4. Error Object

If execution of the called function or procedure finishes successfully without any errors, server must return the *JSON* `Null` instead of the error object (`"error": null`).

4.6. Success Marker

If execution of the called function or procedure finishes successfully without any error, the server must set the success marker to *JSON* `True` value. (`"ok": true`) If execution of the called

function or procedure fails or does not finish successfully, the server must set the success marker to *JSON* `False` value (`"ok": false`).

4.7. Meta Information

For purposes of transmission of information not directly related to the called function or procedure, the response object may have an additional field named `meta` . This field must be of *JSON* object type and may be used to store additional information about the call, such as information about the time and duration of the call, environmental setup and other parameters.

The `meta` field can not be an empty object. It means that if no meta information is provided, `meta` field must be *JSON* `Null` (`"meta": null`). An empty `meta` field (*JSON* `Null`) should be omitted in response.

Field	Type	Meaning	Example
meta	Object	A *JSON* object containing meta information about the call.	`"meta": {...}`

5. General Notes

5.1. Object Field Order

Field order in request and response objects is not limited by the *M1 Specification*, however it is advised that libraries implementing this specification use the strict field order similar to the one stated here. The reason for this is very simple. It is recommended to have the boolean success marker as the last field in a response to indicate the correctness of the message delivery.

5.2. Object Field Names

Request and response objects must contain only those root fields which are described in this specification. It means that request object can not have a root field named e.g. `time` , response object can not have a root field named e.g. `weather` . All the additional information should be transmitted with the help of meta-data fields.

5.3. Error Codes

Each implementation of an RPC server using the *JSON-RPC M1* specification must support following error codes.

Code	Short Name	Description
-1	Request is not readable.	Request text is not parsable as a valid *JSON* message.

Code	Short Name	Description
-2	Invalid request.	Received *JSON* is valid, but request object is not valid.
-4	Unsupported protocol.	Requested RPC protocol is not supported by server.
-8	Unknown method.	Requested RPC method is not known to server.
-16	Invalid parameters.	Method parameters are invalid or contain an error.
-32	Internal RPC error.	An internal RPC server error or an exception during method call.

Error codes `-64`, `-128` and `-256` are reserved for possible future revisions of this document and may not be used by an RPC server.

Error codes of `1` and all greater positive integer numbers may be used by errors generated by RPC user functions.

5.4. Software Implementation

To implement the described logics of request and response objects it is advised to use pointers to object while they can implement all three states of object fields. Following is an example how to implement the `result` field of a response object.

- **Case I.** When the called function returns some values, the `result` is a pointer to an object which is fully set.

```
{
  "result":
  {
      "name": "John",
      "age": 100
  }
}
```

- **Case II.** When the called function returns nothing, the `result` is a pointer to an empty object, i.e. an object with no fields.

```
{
  "result": {}
}
```

- **Case III.** When function call fails, the result is a null pointer.

```
{
  "result": null
```

```
}
```

5.5. Effectiveness

This specification of protocol *JSON-RPC M1* is compiled in a single copy uploaded to the *GitHub* repository named *Vault Thirteen* which is located at the Internet address `github.com/vault-thirteen`. No other copies of this document are effective and not one of the copies can be made effective while this original copy exists at the specified place.
