

# JSON-RPC

JSON REMOTE PROCEDURE CALL

OVERVIEW OF JSON-RPC, A VERY SIMPLE AND  
LIGHTWEIGHT RPC PROTOCOL  
FOR DISTRIBUTED APPLICATIONS

Peter R. Egli  
INDIGOO.COM

## Contents

1. What is JSON-RPC?
2. JSON-RPC interactions
3. Transport options for JSON-RPC
4. JSON serialization format
5. JSON-RPC 2.0 examples
6. When to use JSON-RPC

## 1. What is JSON-RPC?

JSON-RPC is a simple RPC mechanism, similar to XML-RPC.

### Protocol:

Unlike XML-RPC which is a client-server protocol, JSON-RPC is a *peer-to-peer* protocol. It uses JSON (Javascript Object Notation, RFC4627) as the serialization format and plain TCP streams or HTTP as transport mechanism.

### JSON message types:

JSON-RPC defines 3 message types:

#### Request:

Method invocation with arguments encoded in JSON.

#### Response:

Reply to method invocation containing the return argument encoded in JSON.

#### Notification:

Asynchronous request without response.

### Specification:

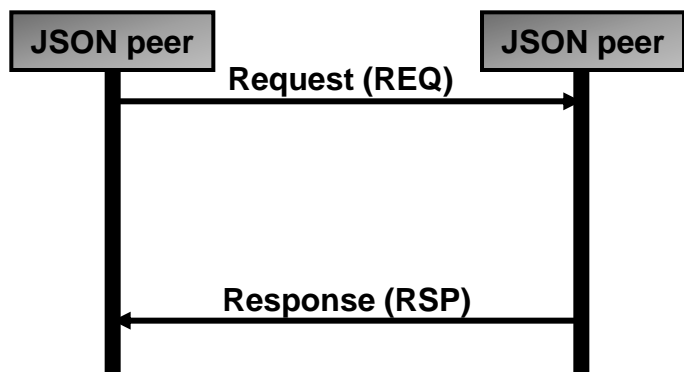
JSON-RPC is very simple, i.e. the JSON-RPC specification is very short (ca. 5 pages). See <http://json-rpc.org/>.

## 2. JSON-RPC interactions

JSON-RPC defines 2 message exchange patterns that support most of the usual peer2peer interaction schemes.

### A. Request-Response:

The sending JSON peer invokes a method on the remote JSON peer with a JSON request. The remote peer sends back a JSON response message.



Request message:

method	Method to be invoked.
params	Arguments to be passed to method as an array of objects.
id	Request ID to match request and response.

Response message:

result	Object returned by the called method.
error	Error object if an error occurred.
id	Request ID to match request and response.

### B. Notification:

The sending peer sends a single notification message. There is no response message.



Notification message:

method	Method to be invoked.
params	Arguments to be passed to method as an array of objects.
id	Must be null (nothing to match).

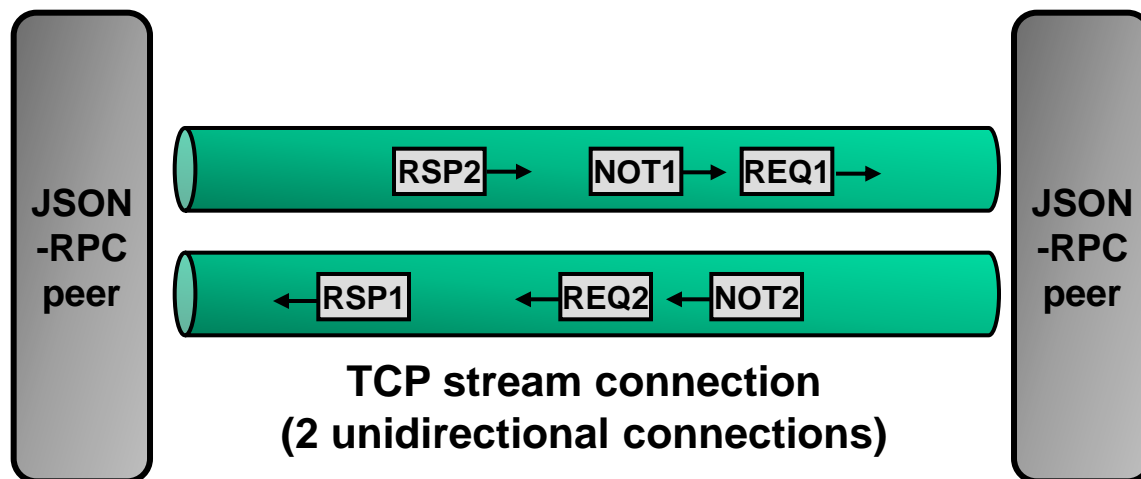
## 3. Transport options for JSON-RPC (1/2)

JSON does not require a specific transport protocol.

The JSON-RPC standard defines 2 transport protocols for conveying JSON messages.

### A. TCP stream (default transport):

The JSON-RPC default transport is a simple TCP stream (JSON-RPC exchanges serialized objects over plain TCP sockets).



## 3. Transport options for JSON-RPC (2/2)

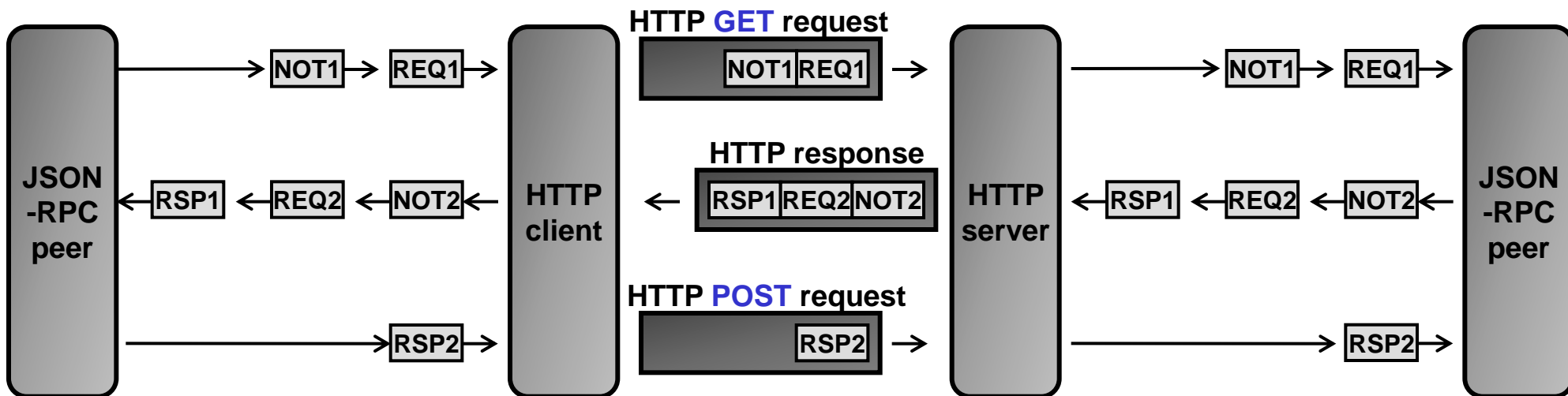
### B. HTTP connection:

Here an HTTP connection is used as a transport channel.

The HTTP-request is used as transport container for JSON-request (REQ), JSON-responses (RSP) and JSON-notification (NOT) messages that are sent from the JSON-peer with the HTTP client. Likewise the HTTP-response is used for transporting JSON-messages from the JSON-peer with the HTTP server.

N.B.: There is no mapping of JSON request to HTTP-request. HTTP is merely the transport mechanism.

The HTTP client and server will need to use a **long polling scheme** so the server always has a response ready to use for a RSP, REQ or NOT message.



## 4. JSON serialization format (1/2)

JSON (Javascript Object Notation, RFC4627) is a lightweight, text-based, language-independent data exchange format. JSON text is a sequence of tokens.

### JSON types:

#### 1. Primitive types:

string	Sequence of 0..n Unicode characters, enclosed in quotation marks. Example: „hello world“
number	Numerical value (representation as used in most programming languages). Examples: 3.45, 5E3
boolean	true / false value
null	Null value (= no object or no value)

#### 2. Structured types:

Array	Ordered sequence of 0..n values. Example: [1,3,4]
Object	Unordered collection of 0..n name:value pairs. Name = string Value = string, number, boolean, null, object, array. Example: {"jsonrpc": "2.0", "method": "update", "params": [1,2,3,4,5]}

### Encoding:

JSON text is encoded in Unicode. The default encoding is UTF-8.

## 4. JSON serialization format (2/2)

JSON grammar expressed in ABNF (excerpt only, ABNF syntax see [RFC5234](#)):

JSON-text = object / array;

begin-array = ws %x5B ws ; [ left square bracket

begin-object = ws %x7B ws ; { left curly bracket

end-array = ws %x5D ws ; ] right square bracket

end-object = ws %x7D ws ; } right curly bracket

name-separator = ws %x3A ws ; : colon

value-separator = ws %x2C ws ; , comma

whitespace = \*{%x20 / %x09 / %x0A / %x0d}

value = false / null / true / object / array / number / string

false = %x66.61.6c.73.65 ; false

null = %x6e.75.6c.6c ; null

true = %x74.72.75.65 ; true

object = begin-object [ member \*( value-separator member ) ] end-object

member = string name-separator value

array = begin-array [ value \*( value-separator value ) ] end-array

number = [ minus ] int [ frac ] [ exp ]

string = quotation-mark \*char quotation-mark



## 5. JSON-RPC 2.0 examples (1/2)

### Notation:

--> Data sent to JSON service

<-- Data coming from JSON service

### RPC call with parameters:

--> {"jsonrpc": "2.0", "method": "subtract", "params": [84, 42], "id": 1}

<-- {"jsonrpc": "2.0", "result": 42, "id": 1}

### RPC call with named parameters:

--> {"jsonrpc": "2.0", "method": "subtract", "params": {"subtrahend": 42, "minuend": 84}, "id": 3}

<-- {"jsonrpc": "2.0", "result": 42, "id": 3}

### Notification:

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

### RPC call with invalid JSON:

--> {"jsonrpc": "2.0", "method": "foobar, "params": "bar", "baz]

<-- {"jsonrpc": "2.0", "error": {"code": -12345, "message": "Parse error."}, "id": null}

## 5. JSON-RPC 2.0 examples (2/2)

### RPC call batch (multiple JSON requests mapped to one JSON packet):

```
--> [
  {"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"}
]
```

## 6. When to use JSON-RPC

### Applicability:

JSON-RPC is well suited for web service applications with the need for bidirectional interaction (peer2peer), but where the complexity of SOAP is not required.

### Example 1:

Remote management of devices over the Internet. SNMP (Simple Network Management Protocol) would be the standard management protocol, but it is difficult to get through the Internet due to the presence of firewalls.

### Example 2:

Web application where the web server needs to update the client (server push). JSON-RPC, as its name implies, was derived from Javascript. The client side of the application is usually Javascript based (e.g. AJAX).