
comodojo/rpcclient Documentation

Release 2.0.0

Marco Giovinazzi

Jul 31, 2018

Contents:

1	Installation	3
1.1	Requirements	3
2	Basic Usage	5
3	Composing requests	7
3.1	Handling special types	7
3.2	About request id	8
4	Using the client	9
4.1	The autoclean switch	9
4.2	Selecting RPC protocol	10
4.3	Change default encoding	10
4.4	Transport	10
4.5	Encryption	10

This library provides a framework (and transport) independent XML and JSON(2.0) RPC client.

Main features are:

- full [XMLRPC](#) and [JSONRPC \(2.0\)](#) protocols support, including multicall and batch requests
- [PSR-3](#) compliant logging
- configurable content encoding
- content encryption (if used in combination with [comodojo/rpcserver](#))

CHAPTER 1

Installation

First install composer, then:

```
composer require comodojo/rpcclient
```

1.1 Requirements

To work properly, comodojo/rpcclient requires PHP $\geq 5.6.0$.

CHAPTER 2

Basic Usage

Following a quick and dirty example of lib basic usage.

Note: For more detailed informations, please see *Using the client* and *Composing requests* pages.

```
1 <?php
2
3 use \Comodojo\RpcClient\RpcClient;
4 use \Comodojo\RpcClient\RpcRequest;
5 use \Exception;
6
7 try {
8
9     // create a RpcClient instance (default XML)
10    $client = new RpcClient( "http://phpxmlrpc.sourceforge.net/server.php" );
11
12    // create and inject a request
13    $client->addRequest( RpcRequest::create("echo", ['Hello Comodojo!']) );
14
15    // send the request
16    $result = $client->send();
17
18 } catch (Exception $e) {
19
20     /* something did not work :( */
21     throw $e;
22
23 }
24
25 echo $result;
```

Composing requests

Each RPC request, regardless of the protocol, should be composed as an instance of the class `\Comodojo\RpcClient\RpcRequest`.

Each request, once created, generate it's unique id that will never be transmitted to the server but is used to discriminate requests in case of batches.

For example, to invoke the `echo` method on the server side that may accept a parameter:

```
1 <?php
2
3 use \Comodojo\RpcClient\RpcRequest;
4
5 $request = new RpcRequest;
6 $request
7     ->setMethod('echo')
8     ->setParameters([
9         "Hello world!"
10    ]);
```

Or using the static `RpcRequest::create` constructor:

```
1 <?php
2
3 use \Comodojo\RpcClient\RpcRequest;
4
5 $request = RpcRequest::create('echo', ["Hello world!"]);
```

3.1 Handling special types

When using XML-RPC protocol, `base64`, `datetime` and `cdata` parameters must me explicitly declared using the `RpcRequest::setSpecialType` method, in order to produce a well formatted xml output.

```
1 <?php
2
3 use \Comodojo\RpcClient\RpcRequest;
4
5 $request = new RpcRequest;
6
7 // base64 of: "I checked it very thoroughly", said the computer,
8 // "and that quite definitely is the answer. I think the problem,
9 // to be quite honest with you, is that you've never actually
10 // known what the question is."
11 $parameters = [
12     "Ikkgy2hlY2t1lZCBpdCB2ZXJ5IHRob3JvdWdob".
13     "HkiLCBzYWlkIHRoZSBjb21wdXRlciwgImFuZC".
14     "B0aGF0IHFlaXRlIGRlZmluaXRlbHkggaXMgdGh".
15     "lIGFuc3dlci4gSSB0aGluyaYB0aGUgcHJvYmxl".
16     "bSwgdG8gYmUgcXVpdGUgaG9uZXN0IHdpdGgge".
17     "W91LCBpcyB0aGF0IHlvdSd2ZSBuZXZlciBhY3".
18     "RlYWxseSBrbm93biB3aGF0IHRoZSBxdWVzdG1".
19     "vbiBpcy4i"
20 ];
21
22 $request
23     ->setMethod('echo')
24     ->setParameters($parameters)
25     ->setSpecialType($parameters[0], "base64");
```

3.2 About request id

When using a JSON-RPC protocol, each request should have it's own id, otherwise it is assumed to be a notification.

Note: Please see [jsonrpc specs](#) for more information about how request are structured and interpreted.

By default, this library will assign a random id to each request. The method `RpcRequest::setId` is available to override this behaviour:

- if id is set to true, the lib will generate a random id (default behaviour);
- if id is an integer or a string, it will be left intact;
- if id is set to null, the request will be treated as a notification.

Using the client

Once a request is composed, the `\Comodojo\RpcClient\RpcClient` class can be used to send it to the server and retrieve the response. It will seamlessly format the message according to selected protocol, send it to the server, read and decode the response.

```
1 <?php
2
3 use \Comodojo\RpcClient\RpcClient;
4 use \Comodojo\RpcClient\RpcRequest;
5
6 $client = new RpcClient("http://phpxmlrpc.sourceforge.net/server.php");
7 $client->addRequest( RpcRequest::create("echo", ['Hello Comodojo!']) );
8 $result = $client->send();
```

Optionally, the client can be configured to use a PSR-3 compliant logger and/or a custom transport (second and third arguments, see below).

4.1 The autoclean switch

By default, the client will hold one or more requests until the `RpcClient::send` method is invoked and it cleans the request's stack when a response is received.

This behaviour can be overridden using the `RpcClient::setAutoclean` method that will force the client to keep each request and reply it on next iteration.

```
1 <?php
2
3 use \Comodojo\RpcClient\RpcClient;
4 use \Comodojo\RpcClient\RpcRequest;
5
6 $client = new RpcClient("http://phpxmlrpc.sourceforge.net/server.php");
7 $client->setAutoclean(false);
8 $client->addRequest( RpcRequest::create("echo", ['Hello Comodojo!']) );
```

(continues on next page)

(continued from previous page)

```
9  $result = $client->send();
10 $client->addRequest( RpcRequest::create("echo", ['Hello Comodojo...2!']) );
11
12 // client will send two requests to the server and result will consequently contain
13 // two responses.
14 $result = $client->send();
```

4.2 Selecting RPC protocol

To select XMLRPC (default) or JSONRPC protocols:

```
1  <?php
2
3  use \Comodojo\RpcClient\RpcClient;
4
5  $client = new RpcClient("http://phpxmlrpc.sourceforge.net/server.php");
6  $client->setProtocol(RpcClient::JSONRPC);
7  // or
8  // $client->setProtocol(RpcClient::XMLRPC);
```

4.3 Change default encoding

By default, client will encode requests in *utf-8*. To select a different encoding:

```
1  <?php
2
3  use \Comodojo\RpcClient\RpcClient;
4
5  $client = new RpcClient("http://phpxmlrpc.sourceforge.net/server.php");
6  $client->setEncoding('ISO-8859-2');
```

4.4 Transport

The `RpcClient` comes with an embedded HTTP transport manager that makes use of `comodojo/httprequest` lightweight library.

If class is not inited with a custom transport, this one will be used.

To access transport instance, the `RpcClient::getTransport` method can be used before invoking `RpcClient::send`.

Custom transport classes should implement the `\Comodojo\RpcClient\Interfaces\Transport`. The `SocketTransport` of `comodojo/daemon` library is a good example to start with.

4.5 Encryption

When used in combination with `comodojo/rpcserver`, the `RpcClient` can be configured to seamlessly encrypt messages and decrypt responses using a pre shared key.

To enable this feature, a key can be passed to `RpcClient::getTransport` method:

```
1 <?php
2
3 use \Comodojo\RpcClient\RpcClient;
4
5 $client = new RpcClient("http://example.com/rpcserver");
6 $client->setEncryption('this is my super secret key');
```