comodojo/rpcclient Documentation

Release 2.0.0

Marco Giovinazzi

Apr 06, 2019

Contents:

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 >=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.

```php
<?php

use \Comodojo\RpcClient\RpcClient;
use \Comodojo\RpcClient\RpcRequest;
use \Exception;

try {

    // create a RpcClient instance (default XML)
    $client = new RpcClient( "http://phpxmlrpc.sourceforge.net/server.php" );

    // create and inject a request
    $client->addRequest( RpcRequest::create("echo", ['Hello Comodojo!']) );

    // send the request
    $result = $client->send();

} catch (Exception $e) {

    /* something did not work :( */
    throw $e;

}

echo $result;
```

CHAPTER 3

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:

```php
<?php

use \Comodojo\RpcClient\RpcRequest;

$request = new RpcRequest;
$request
    ->setMethod('echo')
    ->setParameters([
        "Hello world!"
    ]);
```

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

```php
<?php

use \Comodojo\RpcClient\RpcRequest;

$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.

```php
<?php

use \Comodojo\RpcClient\RpcRequest;

$request = new RpcRequest;

// base64 of: "I checked it very thoroughly", said the computer,
//  "and that quite definitely is the answer. I think the problem,
//  to be quite honest with you, is that you've never actually
//  known what the question is."
$parameters = [
    "IkkgY2hlY2tlZCBpdCB2ZXJ5IHRob3JvdWdob".
    "HkiLCBzYWlkIHRoZSBjb21wdXRlciwgImFuZC".
    "B0aGF0IHF1aXRlIGRlZmluaXRlbHkgaXMgdGh".
    "lIGFuc3dlci4gSSB0aGluayB0aGUgcHJvYmxl".
    "bSwgdG8gYmUgcXVpdGUgaG9uZXN0IHdpdGgge".
    "W91LCBpcyB0aGF0IHlvdSd2ZSBuZXZlciBhY3".
    "R1YWxseSBrbm93biB3aGF0IHRoZSBxdWVzdGl".
    "vbiBpcy4i"
];

$request
    ->setMethod('echo')
    ->setParameters($parameters)
    ->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 availabe 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.

CHAPTER 4

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.

```
<?php

use \Comodojo\RpcClient\RpcClient;
use \Comodojo\RpcClient\RpcRequest;

$client = new RpcClient("http://phpxmlrpc.sourceforge.net/server.php");
$client->addRequest( RpcRequest::create("echo", ['Hello Comodojo!']) );
$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.

```
<?php

use \Comodojo\RpcClient\RpcClient;
use \Comodojo\RpcClient\RpcRequest;

$client = new RpcClient("http://phpxmlrpc.sourceforge.net/server.php");
$client->setAutoclean(false);
$client->addRequest( RpcRequest::create("echo", ['Hello Comodojo!']) );
```

(continues on next page)

(continued from previous page)

```
$result = $client->send();
$client->addRequest( RpcRequest::create("echo", ['Hello Comodojo...2!']) );

// client will send two requests to the server and result will consequently contain two responses.
$result = $client->send();
```

4.2 Selecting RPC protocol

To select XMLRPC (default) or JSONRPC protocols:

```
<?php

use \Comodojo\RpcClient\RpcClient;

$client = new RpcClient("http://phpxmlrpc.sourceforge.net/server.php");
$client->setProtocol(RpcClient::JSONRPC);
// or
// $client->setProtocol(RpcClient::XMLRPC);
```

4.3 Change default encoding

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

```
<?php

use \Comodojo\RpcClient\RpcClient;

$client = new RpcClient("http://phpxmlrpc.sourceforge.net/server.php");
$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 reponses using a pre shared key.

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

```php
<?php

use \Comodojo\RpcClient\RpcClient;

$client = new RpcClient("http://example.com/rpcserver");
$client->setEncryption('this is my super secret key');
```