comodojo/rpcserver documentation

Release 2.0.0

Marco Giovinazzi

Jul 03, 2018

Contents:

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

It is designed to work in combination with a REST framework that could handle the transport side (such as comodojo/dispatcher).

Main features are:

- full XMLRPC and JSONRPC(2.0) protocols support, including multicall and batch requests
- embedded introspection methods
- PSR-3 compliant logging
- payload decoding/econding and encryption
- support for multiple signatures per method

Following capabilities are supported out of the box:

- xmlrpc
- system.multicall
- introspection
- nil
- faults_interop
- json-rpc

Additional capabilities could be implemented *Extending the library*.

CHAPTER 1

Installation

First install composer, then:

```
composer require comodojo/rpcserver
```

1.1 Requirements

To work properly, comodojo/rpcserver requires PHP >=5.6.0.

CHAPTER 2

Basic Usage

Following a quick and dirty example of lib basic usage, without a framework that mediates RPC requests.

Note: For more detailed informations, please see *Using the RPC Server* and *Creating methods* pages.

```php
<?php

use \Comodojo\RpcServer\RpcServer;
use \Comodojo\RpcServer\RpcMethod;
use \Exception;

// get the raw request payload (using, for example, an HTTP::POST)
$payload = file_get_contents('php://input');

try {

    // create a RpcServer instance (i.e. JSON)
    $server = new RpcServer(RpcServer::JSONRPC);

    // create a method (using a lambda functions)
    $method = RpcMethod::create("example.sum", function($params) {
            $a = $params->get('a');
            $b = $params->get('b');
            return intval($a) + intval($b);
    })
    ->setDescription("Sum two integers")
    ->setReturnType('int')
    ->addParameter('int','a')
    ->addParameter('int','b');

    // register newly created method into server
    $server->getMethods()->add($method);

```

(continues on next page)

(continued from previous page)

```
29      // set the payload
30      $server->setPayload($request);
31
32      // serve the request
33      $result = $server->serve();
34
35  } catch (Exception $e) {
36
37      /* something did not work :( */
38      throw $e;
39
40  }
41
42  echo $result;
```

CHAPTER 3

Creating methods

The `\Comodojo\RpcServer\RpcMethod` class should can be used to create custom RPC methods to inject into server.

It requires basically a method name and a callable, provided as lambda function, named function or couple class::method.

Parameters can be added using `RpcMethod::addParameter`; multiple signatures can be specified using the `RpcMethod::addSignature` method.

For example, to create a *my.method* RPC method mapped to `\My\RpcClass::mymethod()` that supports two different signatures:

```
<?php

use \Comodojo\RpcServer\RpcMethod;

// create a method using class::method pattern
$method = RpcMethod::create('my.method', '\My\RpcClass::mymethod')
    // provide a description for the method
    ->setDescription("My method")

    // for now on, parameters and return type will be associated
    // to the first (default) signature, until next addSignature() marker

    // set the return type (default: undefined)
    ->setReturnType('boolean')

    // start another signature, the second one
    ->addSignature()

    // set the return type for second signature
    ->setReturnType('boolean')

    // add expected parameters (if any) for second signature
```

(continues on next page)

(continued from previous page)

```
23      ->addParameter('int','a')
24      ->addParameter('int','b');
```

Note: Signatures are automatically matched by the server as well as received parameters.

If a request does not match any valid signature, an *Invalid params* (-32602) error is returned to the client.

3.1 Defining Callbacks

As in previous example, the `\My\RpcClass::mymethod()` has to be created to handle the request.

This method should expect a `\Comodojo\RpcServer\Request\Parameters` object that provides:

- received parameters (`Parameters::get`)
- **server properties**
  - capabilities (`Parameters::getCapabilities`)
  - methods (`Parameters::getMethods`)
  - errors (`Parameters::getErrors`)
- RPC protocol in use (`Parameters::getProtocol`)
- logging interface (`Parameters::getLogger`)

```
1   <?php
2
3   use \Comodojo\RpcServer\Request\Parameters;
4
5   class RpcClass {
6
7       public static function mymethod(Parameters $params) {
8
9           // retrieve 'a' param
10          $a = $params->get('a');
11
12          // retrieve 'b' param
13          $b = $params->get('b');
14
15          // get current PSR-3 logger
16          $logger = $params->getLogger();
17
18          // get current protocol
19          $current_rpc_protocol = $params->getProtocol();
20
21          // log something...
22          $logger->info("mymethod called, current protocol: $current_rpc_protocol,␣
    ↪parameters in context", [$a, $b]);
23
24          return $a === $b;
25
26      }
27
28  }
```

3.2 Injecting extra arguments

In case the callback method needs extra arguments in input, they should be specified as additional arguments in method declaration.

Server will transfer them when callback is fired.

As an example, a method declaration like:

```php
<?php

use \Comodojo\RpcServer\RpcMethod;

// create a method that transfer two additional arguments
$method = RpcMethod::create(
    'my.method',
    '\My\RpcClass::mymethod',
    \My\Extra\Attribute $attribute,
    $another_attribute
)
->setDescription("My method")
->setReturnType('boolean');
```

Will invoke a callback like:

```php
<?php

use \Comodojo\RpcServer\Request\Parameters;
use \My\Extra\Attribute;

class RpcClass {

    public static function mymethod(
        Parameters $params,
        Attribute $attribute,
        $another_attribute
    ) {

        // ... method internals

    }

}
```

CHAPTER 4

Using the RPC Server

The class `\Comodojo\RpcServer\RpcServer` realizes the core server component.

Note: As already mentioned, the server component does not provide transport management but just the logic to understand and serve RPC requests.

Other frameworks or custom implementations can be used to mediates requests using HTTP, sockets or any other message-delivery transport.

An RPC Server should be created specifying the desired RPC protocol; constants `RpcServer::JSONRPC` and `RpcServer::XMLRPC` are available to setup server correctly.

The optional parameter *$logger* expects an implementation of `\Psr\Log\LoggerInterface` and implicitly enable internal logging.

Once created, server expects a payload, and starts processing it when `RpcServer::serve` method is invoked:

```
<?php

use \Comodojo\RpcServer\RpcServer;

// create the server
$server = new RpcServer(RpcServer::XMLRPC);

// (optional) set encoding (default to *utf-8*)
$server->setEncoding('ISO-8859-1');

// feed server with request's payload and start serving the request
$result = $server->setPayload($request)->serve();
```

4.1 Encrypting communications

This package provides a *non-standard* PSK message-level encryption (using AES).

This working mode could be enabled specifying the pre shared key using `RpcServer::setEncryption` method.

Note: The only client that supports this communication mode is the one provided by comodojo/rpcclient package.

CHAPTER 5

Extending the library

Beside RPC methods, server capabilities and errors can be added or removed using dedicated server methods.

5.1 Server Capabilities

The `RpcServer::getCapabilites` allows access to capabilities manager that can be used to modify standard supported capabilities.

For example, to add a new capability:

```php
<?php

use \Comodojo\RpcServer\RpcServer;

// init the server
$server = new RpcServer(RpcServer::JSONRPC);

// ad a new capability
$capabilities = $server->getCapabilities();
$capabilities->add("my.capability", "http://url.to.my/capability", 1.0);
```

5.2 Custom Errors

Errors can be managed using the `RpcServer::getErrors` method.

For example, to add a new error:

```php
<?php

use \Comodojo\RpcServer\RpcServer;

```

(continues on next page)

(continued from previous page)

```
// init the server
$server = new RpcServer(RpcServer::JSONRPC);

// ad a new capability
$errors = $server->getErrors();
$errors->add(-31010, "Transphasic torpedo was banned by the United Federation of
↪Planets");
```