# An introduction to XML-RPC in PHP

**By Dennis Pallett**

## Introduction

In this tutorial you'll learn all about XML-RPC and how to "open" your PHP scripts to the rest of the world, effectively creating a web service.

According to Google a webservice "is any piece of software that makes itself available over the Internet and uses a standardized XML messaging system", which exactly describes XML-RPC. Basically, a web service allows other developers to query your web site, and perform certain tasks or get specific information, without having to go to your website, and this includes desktop applications.

XML-RPC web services are commonly included with blogging tools (such as WordPress) and content management systems, but many other websites benefit from having a public web service, especially web sites that serve data, for example a weather website.

In this tutorial I will show you how to create your own XML-RPC web service, allowing other people to connect to your website. I will also show you how to create your own XML-RPC client, which means you can query other web services.

Although PHP comes with inbuilt XML-RPC functionality, we won't be using it in this tutorial, and instead we'll opt for the excellent XML-RPC Library by Simon Willison, available at http://scripts.incutio.com/xmlrpc/. This library includes both the ability to create a XML-RPC server and client, which is exactly what we need.

Let's start by creating our own XML-RPC server first, and after that we'll create our own client.

## Creating a simple XML-RPC server

With the XML-RPC library, creating a XML-RPC server takes very little work, and most of it gets done automatically for you. If you have a look at the manual you will read that all it takes is the following code:

```
<?php

include('xmlrpc.php');

function sayHello($args) {
    return 'Hello!';
 }

function addTwoNumbers($args) {
    $number1 = $args[];
    $number2 = $args[1];
    return $number1 + $number2;
 }

$server = new IXR_Server(array(
    'demo.sayHello' => 'sayHello',
    'demo.addTwoNumbers' => 'addTwoNumbers'
 ));

?>
```

The above code first includes the XML-RPC library (called xmlrpc.php), and then defines two functions, which will be part of our server. After that it creates the actual server by creating a new instance of the IXR_Server() class, and passing the server functions as an argument in the constructor.

That's basically how the XML-RPC server works, and all you have to do is extend it with your own functions and methods. The methods in the above code are called 'demo.sayHello' and 'demo.addTwoNumbers', but you can call them anything you'd like, e.g. 'myapp.MyFunc' and link them to any function you'd like.

Every server method only accepts one argument, which is an array of arguments passed by the client. For example, if the client sent two arguments to the sayHello method in the above code, the $args variable would be an array consisting of the two arguments, i.e.

```
Array
 (
    [0] => Argument 1

   [1] => Argument 2
 )
```

If there's only one argument, then the $args variable will be that argument.

Let's have a look at creating a XML-RPC server that can describe its methods, by using the IXR_IntrospectionServer() class.

## A self-describing server

The XML-RPC library comes with a few standard methods which are built in, and are part of the XML-RPC specification, like for example system.listMethods. This method will return an array to the client of all the methods that are supported by the server, but it won't return any specific details, like the number of arguments each method takes, or its return value.

For this you need the system.methodHelp method, but this is only supported by the IXR_IntrospectionServer() class, and not the regular server class (which we just used). Let's re-create our server, with the IXR_IntrospectionServer class:

```
<?php

include ('xmlrpc.php');

function sayHello($name) {
    return 'Hello, ' . $name;
 }

function addTwoNumbers($args) {
    $number1 = $args[];
    $number2 = $args[1];
    return $number1 + $number2;
 }

$server = new IXR_IntrospectionServer();
 // Now add the callbacks along with their introspection details
 $server->addCallback(
    'demo.sayHello',  // XML-RPC method name
    'sayHello',       // function to calback
    array('string', 'string'), // Array specifying the method signature
    'Returns the current date as a string'  // Documentation string
 );
 $server->addCallback(
    'demo.addTwoNumbers',
    'addTwoNumbers',
    array('int', 'int', 'int'),
    'Adds up two numbers and returns the result'
 );

// And serve the request
 $server->serve();

?>
```

This XML-RPC server has the exact same methods, except it now supports the system.methodHelp and system.methodSignature methods. As you can see in the above code, for each method, I've added a small blurb of information, and the 'method signature'. The method signature is an array consisting of at least one item, which describes the type of return value. Optionally, you can also add the types of arguments (which I've done). See the library manual for all the valid types.

The advantage of using this self-describing server, instead of the regular server, is that other people can easily understand what your web service does, without having to read any documentation or FAQ'.s.

Now let's create a simple XML-RPC client that can query any XML-RPC web service, and get ready to be surprised about how easy it is.

## A simple XML-RPC client

If you have a look at the manual again, you'll see that all it takes to query a web service is the following code:

```
<?php

include ('xmlrpc.php');

// Create new XML-RPC client
 $client = new IXR_Client(
'http://projects/phpit/content/introduction%20xml-rpc%20in%20php/demos/server1.php');
```

```
// Try the sayHello function
 if (!$client->query('demo.sayHello')) {
     die('Something went wrong - '.$client->getErrorCode().' : '.$client->getErrorMessage());
 }

// Show response
 echo $client->getResponse();

?>
```

(View Live Demo)

Is that easy or what?! What the above code does is first create the XML-RPC client, and passes the URL of the server as an argument. As you can see I'm using a server that's located on my own computer (http://projects points to a folder on my computer). After that the code sends the query ($client->query), and immediately checks if there weren't any errors. If everything is alright, it displays the response of the server.

Let's write a simple client that queries a web service running on PHPit, which simply returns the number of times it's been queried:

```
<?php

include ('xmlrpc.php');

// Create new XML-RPC client
 $client = new IXR_Client('http://www.phpit.net/demo/introduction%20xml-rpc%20in%20php/phpitserver.php');

// Get query count
 if (!$client->query('phpit.getCounter')) {
     die('Something went wrong - '.$client->getErrorCode().' : '.$client->getErrorMessage());
 }

// Show response
 echo 'Number of times queried: ' . $client->getResponse();

?>
```

(View Live Demo)

The above code is very similar to our previous code, and uses the "phpit.getCounter" method to get the number of times the web service has been queried.

## Multiple queries at once

In the previous examples we've only sent one query, but it's possible that you want to send multiple XML-RPC queries in one script, which can be done using the following code:

```
// Try the sayHello function
 if (!$client->query('demo.sayHello', 'Dennis')) {
     die('Something went wrong - '.$client->getErrorCode().' : '.$client->getErrorMessage());
 }
```

```
// Show response
 echo $client->getResponse();

echo '';

// Now try adding up two numbers
 if (!$client->query('demo.addTwoNumbers', 4, 5)) {
     die('Something went wrong - '.$client->getErrorCode().' : '.$client->getErrorMessage());
 }

echo 'Result: ' . $client->getResponse();

echo '';
```

But there is one big disadvantage to the above code, because of the way the internet works. Even though both requests are very small in size (not even a kilobyte), it still carries overhead, because each query is a separate HTTP request, which means a connection must be made for each query. This can easily add up to an extra half a second of load time, which is clearly unacceptable for any PHP script.

That's why the XML-RPC library comes with built-in 'multicall' support, allowing you to send multiple queries at once, but only if the server supports it. The code to do a multicall query looks like this:

```
<?php

include ('xmlrpc.php');

// Create new XML-RPC client
 $client = new IXR_ClientMulticall(
'http://projects/phpit/content/introduction%20xml-rpc%20in%20php/demos/server1.php');

// Add method calls
 $client->addCall('demo.sayHello');
 $client->addCall('demo.addTwoNumbers', 3, 4);

// Send query
 if ($client->query()) {
     $response = $client->getResponse();
 } else {
     echo "<h2>Error! ".$client->getErrorCode().":".$client->getErrorMessage().'</h2>';
 }

// Show response
 echo '<pre>';
 print_r ($response);
 echo '</pre>';

?>
```

(View Live Demo)

As you can see the above code sends two queries to my own server, and then shows both responses. Since you're doing multiple queries at once, each response of each query is located in a new array item.

That's it for XML-RPC clients, and thanks to the XML library, it takes very little code.

## Conclusion

In this tutorial I have taken you through the steps necessary to create your own XML-RPC server and client. Without the XML-RPC library by Simon Willison, it'd would've been 10x times harder, but thanks to his library, most of the hard work is done for us.

I hope I've inspired you to do some really neat things, and if you have any interesting uses of XML-RPC, please e-mail me at dennis [A - T] PHPit [dot] net. There are so many things you can do, and there are already so many public web services (see the list at XMLRPC.com).

Click here to download all the demos in this tutorial.

If you have any comments on this tutorial drop them below, or if you would like to discuss it further, join us at PHPit Forums to talk about anything PHP.

## About this PDF