# Description of PHP-RPC protocol

Ivan Voras

January, 2004.

**Abstract**

Because of the need for a truly light-weight RPC mechanism in PHP, a new protocol is formed: PHP-RPC, to be used instead of XML-RPC when communicating to purely PHP applications. This protocol is faster and has a smaller overhead.

## Contents

**1 Overview** **1**

**2 Making PHP-RPC calls** **1**

**3 Reference implementation** **2**

**4 Final notes** **3**

## 1 Overview

The primary goal of this protocol is to provide a simple, light-weight and most imortantly, fast mechanisam for making remote procedure calls (RPC) for pure PHP applications. Consequently, portability to non-PHP applications is not of significance. The protocol uses PHP's SERIALIZE() and UNSERIALIZE() calls, with BASE64_ENCODE() and BASE64_DECODE() for transport-armour when needed. Because of this, any non-resource PHP data type can be used with no overhead for type-juggling. The underlaying transport mechanism is HTTP.

The dynamics of protocol usage is purely call-and-response, with no innate state tracking (which is left to higher-level systems to implement if needed). Thus it is very simple and at the same time very powerful for real-world usage.

## 2 Making PHP-RPC calls

Firstly, we'll introduce the notion of a PHP-RPC *function identifier*. It is a URI which completely and uniquely identifies one PHP-RPC function. It is of the

form *http://web.site.com/somedir/phpfile.php?function_name(par1,par2)*. This URI references PHP function FUNCTION_NAME() in file phpfile.php, that resides on web site www.site.com/somedir, and specifies that the function will receive PAR1 and PAR2 as it's parameters.

There are two forms of PHP-RPC calls and they are distinguished by the function name part of the URI. The first one is when the function name ends in underline ("_"): then each of the parameters in the URI is base64-encoded output from serialize() of the PHP value intended to be passed as the parameter (the "_" is stripped from the function name before making the actual call). The second form is specified by function name ending without "_", in which case the parameters are in human-readable form (e.g. "string", 1234, true). The functions intended to be called by PHP-RPC can never have names ending in "_". Note that in the first case, any PHP serializable value can be passed, but in the second case, only string, numeric and boolean values can be used as parameters. String values must be enclosed in double quotes and in the case they themselves contain double quotes, the inner quotes must be backslash-escaped. There must not be any extra whitespace in any of the call forms.

Result of the function is always returned as a serialized PHP value, in the body of HTTP response. The result is *not* automatically base64 encoded, rather it depends of the Transfer-Encoding and other HTTP reply headers.

A raw-text dump of HTTP protocol traffic for an example PHP-RPC call is:

```
GET /somedir/phpfile.php?multiply(2,5) HTTP/1.0
]
HTTP/1.0 200 OK
i:10;
```

And this is all there is to it. Notice that the protocol accomplishes its goals:

- It has a much smaller footprint than XML-RPC
- It is also much faster, as it avoids complex creation and parsing of XML structures

# 3 Reference implementation

The reference implementation (found at http://geri.cc.fer.hr/~ivoras/) contains these functions:

```
prpc($fun_uri,[$arg1,...])
prpc_($fun_uri,[$arg1,...])
```

The above functions will perform a PHP-RPC call to function identified by URI in $fun_uri, with variable number of parameters, and will return its result. The first function will make the call using the human-readable form (the first case described in section 2), and the second case will use base64-encoded serialized values (the second case). These are the only client-side functions of the reference implementation. Another function is used at the server side:

```
prpc_server()
```

This function will read the URI by which the current PHP script was called and attempt to call the specified function. It will return TRUE if the URI is valid and the function was successfully called, and FALSE otherwise (e.g. the URI is invalid, the URI is not a PHP-RPC function identifier, there is no such function at the current point in the execution of PHP script, etc. It is valid to have the same PHP script file for generating HTML web pages, and serve PHP-RPC calls, although its not particulary good software design). The function will service both forms of calls.

The reference implementation is made to be as fast as possible, and does not perform any error checking (e.g. for the validity of function arguments). Error checking can be added, but the code will grow much in complexity and execution time.

# 4 Final notes

There are some things that should be kept in consideration when using PHP-RPC:

1. All PHP values can be serialized, but object and resource types are a special case: serializing resource variables makes no sense (and actually results in a bogus serialized value) and proper restoration of objects requires that the class implementation be present at the side that does the unserializing of an object.

2. Due to the limitation of PHP's unserialize() function, it is impossible to differentiate between unserialized boolean FALSE value, and an error in unserialization process.

3. There are subtle bugs in serialize/unserialize code in the early PHP 4 versions, and you should probably not use any version $< 4.2$ for this purpose. The reference code is PHP 4.3-specific.

4. The PHP-RPC mechanism is reasonably portable - the only thing required to use it with other programming languages is implementation of the compatible serialize() and unserialize() functions, which is of near-trivial complexity.