EXTRAS packages for the PHP-XMLRPC library

version 0.5

Gaetano Giunta

EXTRAS packages for the PHP-XMLRPC library: version 0.5

Gaetano Giunta

Table of Contents

Chapter 1. Introduction

This collection of utilities provides many additional features to the PHP classes that are part of the PHP-XMLRPC library.

Altough very useful for specific purposes, these files have been omitted from the base distribution in order to keep it small and focused.

PHP-XMLRPC [???] is a php library which implements the XMLRPC protocol.

XMLRPC is a format devised by Userland Software [http://www.userland.com/] for achieving remote procedure call via XML using HTTP as the transport. XML-RPC has its own web site, www.xmlrpc.com [http://www.xmlrpc.com/]

Support for JSON and JSONRPC is part of this distribution.

JSON [???] is a format devised to ease serialization and deserialization of common data types without incurring the overhead that is normally associated with XML. It is a subset of the Javascript language, and as such it is easily manipulated within web browsers.

JSONRPC [http://www.jsonrpc.org] is a remote procedure call protocol that uses HTTP for transport and a json syntax extremely similar to the xmlrpc one.

Chapter 2. What's new

rev. 0.5

This is the first release of the library to only support PHP 5. The corresponding version of the base library is 3.0.0 beta

Quite a few changes and bugfixes have been made in the are of json support:

improved: catch exceptions thrown during execution of php functions exposed as methods by the jsonrpc server

- improved: catch exceptions thrown during execution of php functions exposed as methods by the jsonrpc server
- fixed: fix bad encoding if same object is encoded twice using php_jsonrpc_encode
- improved: json_extension_api.php: added json_last_error() function and the constants defined in php 5.3.0
- improved: allow 'mixed type' jsonrpc servers (see description in the xmlrpc server docs)
- improved: allow 'phpvals' jsonrpc servers to do type checking on incoming requests
- fixed: a missing 'new' call when building string vals in php_jsonrpc_encode
- fixed: encoding of UTF8 chars outside of the BMP
- fixed: encoding of '/' chars in jsonrpc when source is in UTF8

rev. 0.4

- DOCXMLRPCS got a bugfix for php installs where magic_quotes_gpc is set and an option to take advantage of a visual editor for xmlrpc values (the editor being part of the jsxmlrpc package, it has to be downloaded separately)
- the usual lot of bugfixes for JSONRPC: slightly faster handling of data which is internally encoded as UTF-8; support for booleans, strings, null as valid id in requests and responses; correctly add quotes when serializing datetimes; correct handling of charset declaration in http headers; switched the declared content-type from 'text/plain' to 'application/json' as per the proposed rfc; modified json_extension_api.inc to follow php 445/521 semantics
- added in AJAXMLRPC a new server class that takes advantage of ths jsxmlrpc lib for the generated javascript code (instead of jsolait)
- one bugfix in XMLRPC_EXTENSION_API

rev. 0.3

- This manual was, albeit in complete, added to the distribution
- a new php file has been included in the JsonRPC package, which implements the same API as the PHP native JSON extension: it can be used to replace the extension in those situations where modifications to the application server are not feasible, such as websites on shared hosting etc...
- bugfixes in the JsonRPC and ADODB packages

rev. 0.2

See the NEWS file for a detailed list of changes in release 0.2

Chapter 3. System requirements

The PHP-XMLRPC library base package, version 2.1 or later, is a prerequisite for the extras packages.

A working php setup is of course needed.

The base library is tested to work with php versions ranging from 4.0.5 to 5.2. Please note that not all library features are available for every PHP version. The "curl" php extension is needed to support https calls. The "json" and "xmlrpc" php extensions do not interfere with the operations of either the base library or extras.

A complete testsuite has not yet been implemented for every component that is part of the extras. This prevents complete regression tests to be carried out against all php versions. If you encounter any quirk, especially with old php installs, please report it on the project development pages [http://sourceforge.net/projects/phpxmlrpc].

Chapter 4. List of packages in the distribution

The 'extras' library is formed by many separate components, grouped together for ease of download. The terms 'component' and package' are used interchangeably within this manual to indicate groups of files that provide a particular capability. Please note that the term 'package' is used with a different meaning on the library download pages on the www.sourceforge.net website, where the complete extras collection is in fact called a package in itself.

Included packages as of the current release are:

- ADODB: designed to provide a flexible and easy-to-use database-to-webservice conversion mechanism. Based on the adodb [http://adodb.sourceforge.net] library by John Lim, it includes conversion functions, server component and meta-db-driver that allows transparent access to remote databases over http.
- AJAX: demo of an ajaxified version of the php-xmlrpc lib: supports executing xmlrpc/jsonrpc calls directly from the client browser. Needs the excellent jsolait lib from http://jsolait.net/ (thanks Jan Kollhof) or the jsxmlrpc lib.
- DOCXMLRPCS: subclass of xmlrpc server that auto-generates HTML documentation of exposed services. Easy as a breeze to use, and extremely user-friendly (it is used on the main php-xmlrpc website, too).
- JSONRPC: support for this brand new protocol, 100% buzzword-compliant and ajax-ready. Client and server classes provided. Makes it very easy to build a server that supports both xmlrpc and jsonrpc protocols at the same time. The original JSON parsing code was from Michal Migurski (whose lib is now officially part of PEAR).
- PROXY: subclass of xmlrpc server that can act as remote (transparent) xmlrpc proxy to forward calls to a remote server. Can either forward any received call or probe the remote server first for existing methods.
- WSDL: the completely UNOFFICIAL DTD and RELAX NG schemas to validate your xmlrpc against. Might be useful in defining some wsdl file describing xmlrpc services (good luck!!!). The DTD is not quite accurate, due to inherent limitations in the definition language. The RELAX NG schema should be on the other hand 100% precise and accurate.
- XMLRPC_EXTENSION_API: a replacement for the php native xmlrpc extension, written in 100% pure php. Typical use case: enabling a php application, written taking advantage of the php native xmlrpc extension, to run also on webservers where the extension is not / can not be installed.

Chapter 5. ADODB package

Introduction

The main purpose of this package is to provide a flexible and easy-to-use database-to-webservice conversion mechanism. In layman terms, we want to provide an easy mean of connecting applications to remote databases using a web service protocol instead of a native database driver.

Current features include:

- written in 100% PHP, runs on Windows, Linux and many variants of UNIX
- use of ADODB as database backend enables connection to a great number of different databases (even LDAP!)
- uses XML over HTTP as data transportation layer: can be deployed in different setups and will work across proxies, firewalls etc...
- provides a SOAP inbterface as well as an XMLRPC one
- uses standard web-service protocols with libraries available in many programming languages, allowing coders to write new clients for integration within existing platforms
- includes a generic SQL-to-HTTP proxy that can be used to build two or three-tier application architectures eliminating the need for installation of database connectivity on clients
- can be used to save money on database licensing costs
- if properly coded, clients can become 100% database platform agnostic, i.e. you could switch databases with zero impact on clients and limited impact on the middle-tier
- many, many times slower than accessing databases using native driver interfaces
- does not take advantage of many particular programming features of advanced databases

files

- `client_driver_examples.php`: php demo code showcasing usage of this package to connect to remote databases via the xmlrpc ADODB driver
- `client_examples.php`: php demo code showcasing usage of this package to connect to remote databases via xmlrpc
- `readme.txt`: a short description of the purpose of this package
- `schema.svg`: a graphical representation of the typical deployment scenario
- `drivers/adodb-soap.inc`, `drivers/adodb-xmlrpc.inc`: adodb database driver files, to be used together with the server components for transparent db access
- `lib/tosoap.inc`, `lib/toxmlrpc.inc`: the files containing the core conversion routines
- `server/server.php`, `server/server_config.inc`, `server/server_functions.inc`: the server component: it receives incoming xmlrpc calls and maps them to database operations. Server.php is the webservice endpoint, while server_config.inc is the con figuration file that has to be tailored for every installation

Usage

This distribution consists of 3 main modules:

1. an adodb-recordset to xmlrpc-value (and viceversa) conversion module for adodb, forming the core of the library
2. an xmlrpc web-service server that provides the functionality of a SQL-to-HTTP proxy
3. an xmlrpc driver for ADODB that can be used to transparently connect applications to remote databases using the xmlrpc web-service server

Module 1 can be used stand-alone in any application; module 2 depends on module 1 and module 3 depends on both 1 and 2.

The above schema documents a typical usage scenario. We will refer to that image throughout the rest of the documentation.

- part A is the database to be accessed by the client app. It can be any database supported by adodb
- part B is the PHP-powered web service that lets remote clients access the database using xmlrpc function calls. It needs a php-supporting web server to run
- part C is the client application, and is the most varying of the 3 parts:
  - it can be written in PHP and make use of the adodb xmlrpc driver (module 3 above). The programmer only needs to know the adodb api.
  - it can be written in PHP and make direct use of xmlrpc calls. The programmer needs to know the phpxmlrpc api and decode 'by hand' the recordset received from the db
  - it can be written in any other language than PHP, making use of an appropriate xmlrpc library
  - it can be part of a website as well as a stand-alone application

Needless to say, all 3 parts can reside on the same as well as on 3 completely separate physical servers (any combination is allowed).

A few usage cases:

- A is SQLserver, B is IIS. They both reside on the same Windows 200 server. C is a PHP application running in Apache on Linux. Advantage: no need of ODBC drivers on the linux server
- A is Sybase, B is Apache. They both reside on the same Solaris server. C is is a PHP application running as a service on a SCO Unix. Advantage: no need for recent Sybase client libraries when compiling PHP on SCO
- A is Oracle, B is Netscape Fasttrack. They reside on different servers in the intranet. C is a PHP application running in Apache on a hardened Solaris in the DMZ. Advantage: instead of giving direct access to the internal database through the firewall separating the intranet and DMZ networks, only a (custom) port is opened for HTTP request

Installation

Server side (tier B of schema)

- unpack the php xmlrpc lib and put it into a directory where it can be included by php scripts (see directive include in php.ini). NB: only the two files xmlrpc.inc and xmlrpcs.inc a re actually needed.

NNB: to avoid potential security problems possibly put them in a directory outside of the web server root!

- unpack the adodb lib and put it into a directory where it can be included by php scripts NNB: to avoid potential security problems possibly put it in a directory outside of the web server root!
- unpack in a temp dir the contents of this distribution, then
  - copy toxmlrpc.inc.php in the main adodb dir
  - copy adodb-xmlrpc.inc.php in ADODB/drivers
  - choose a directory inside the web server root where the web service will be active. Copy in there the files server.php, server_config.inc.php and server_functions.inc.php
- edit the configuration of the web service: edit the file server_config.inc.php

Client side (tier C of schema)

- unpack the php xmlrpc lib and put it into a directory where it can be included by php scripts (see directive include in php.ini). NB: only the two files xmlrpc.inc and xmlrpcs.inc a re actually needed. NNB: to avoid potential security problems possibly put them in a directory outside of the web server root!
- unpack the adodb lib and put it into a directory where it can be included by php scripts NNB: to avoid potential security problems possibly put it in a directory outside of the web server root!
- unpack in a temp dir the contents of this distribution, then
  - copy toxmlrpc.inc.php in the main adodb dir
  - copy adodb-xmlrpc.inc.php in ADODB/drivers

API documentation

To be documented...

SOAP version

To be documented... suffice to say that we provide a wsdl file and a nusoap-based implementation, too

Road map

Add support in the SQL-to-XMLRPC proxy for the missing 90% functionality of adodb, such as sql bind parameters; create a stand-alone web-service server making use of a continuously running php script (e.g. nanoweb or pear::http) instead of using an external web server; benchmark for speed / bandwidth; see all todo items in the code

Chapter 6. Ajax package

files

- `axjaxdemo.php`: demo file showcasing construction of a complete ajax client/server solution in a single php file, based on xmlrpc and jsolait
- `ajaxdemo2.php`: demo file showcasing construction of a complete ajax client/server solution in a single php file, based on xmlrpc and jsxmlrpc
- `ajaxmlrpc.inc`: file containing core logic to wrap xmlrpc function calls and exposed methods into js objects
- `sonofajax.php`: demo file showcasing construction of a complete ajax client/server solution in a single php file, based on jsonrpc

usage

To be documented...

Chapter 7. DocXmlRpc package

files

- `docxmlrpcs.inc`: contains the class extending `xmlrpc_server` with the capability to generate html documentation of exposed methods
- `docxmlrpcs.css`: the stylesheet used by defalt by the self-generated html documentation

usage

The purpose of this package is to let users have human-readable documentation automatically generated for all xmlrpc methods that are implemented by a given xmlrpc server.

The documentation produced will be in HTML format, and it will exactly match the information that the xmlrpc_server class makes available to clients via usage of the `system.methodHelp`, `system.methodSignature` and `system.listMethods` xmlrpc calls. As an extra feature, documentation for single parameters of xmlrpc methods can be added. Html forms will be included with every method synopsis description page, to help the developer do quick'n'dirty debugging.

The simplest way to make usage of the extra capabilities of this package is to take an existing xmlrpc_server and swap the php class used with `documenting_xmlrpc_server`:

```
// define the dispatch map describing all of the xmlrpc methods exposed by this server and the php
$dmap = array(
  '' => array(),
  ...
);

// include the php code implementing the xmlrpc methods
...

// build the server and let it do its job: that's it!
$server = new documenting_xmlrpc_server($dmap);
```

Since version 0.4, the html forms that are generated by the server class can take advantage of the javascript-based visual xmlrpc value editor that is part of the jsxmlrpc library (downloadable as a separate package from the sourceforge.net project pages), making it even easier to invoke the implemented webservices via a browser interface. This optional feature can be enabled by setting the `editorpath` member of the server:

```
$s = new documenting_xmlrpc_server($dmap, false);
$s->editorpath = '../javascript/'; // enable link to js visual editor of content: set this to the d
$s->setdebug(3); // enable maximum debugging level, just in case
$s->service();
```

Chapter 8. JsonRPC package

files

- `jsonrpc.inc`: contains the classes implementing the JSONRPC client and the JSON encoding/decoding functionality. `include()` this and `xmlrpc.inc` in your PHP files to use the classes
- `jsonrpcs.inc`: contains the JSONRPC server class. `include()` this in addition to `xmlrpc.inc`, `xmlrpcs.inc` and `jsonrpc.inc` to get server functionality
- `json_extension_api.php`: contains the functions that emulate the php JSON extension API. `include()` this in addition to `xmlrpc.inc` and `jsonrpc.inc` in your PHP files to have access to the required functionality
- `server.php`:
- `benchmark.php`:
- `testsuite.php`: testsuite used to verify adherence of this implementation to the native JSON extension

JSONRPC functionality

The API is basically the same as the PHP-XMLRPC base library: all the classes have "jsonrpc" instead of "xmlrpc" in their names, but method names and members do not vary.

To be documented...

Native JSON extension emulation layer

The `json_extension_api.php` file has been created with the goal of letting developers write applications taking advantage of the functionality offered by the php JSON extension, and allow them to deploy on php installations where that extension is not /cannot be installed. Since the reimplementation is written in 100% php, it can be deployed on virtually any php setup by simply copying three php files. The API of the native library is reproduced feature for feature and bug for bug, with the following exceptions:

- when trying to `json_encode()` a php string value that contains characters outside of the UTF-8 encoding range, this implementation will return a json string containing different characters from the ones intended (a.k.a. garbage), while the native extension discards invalid characters and any following. Please note that if you are trying to do this your code is most probably wrong.

```
var_dump(json_encode("Günter, Elène"));

// this implenentation
string(19) ""Gnter, El\u8ba5ne""

// native extension
string(3) ""G""
```

- when calling `json_encode()` on a php float value, the json representation generated by this implementation will always contain a fractional part, whereas the native extension does not. This feature allow us, when using this implementation to both encode and decode, to distinguish between int and float values and rebuild them with the correct php type, despite the JSON protocol lacks two separate types.

```
var_dump(json_encode(1.0);
```

```
// this implementation
string(3) "1.0"

// native extension
string(1) "1"
```

An interesting feature of this implementation, not offered by the native implementation, is that it allows to transparently encode/decode php strings to/from json using the ISO-8859-1 character set for the php representation, instead of UTF-8. ISO-8859-1 is the native character set of php, so this is the behaviour that is most likely to best fit the majority of usages.

API documentation

Please note that if you `include()` the `json_extension_api.php` file on a php installation where the native JSON extension is loaded, the functions provided by this implementation will be automatically renamed to `json_alt_encode` and `json_alt_decode`. This prevents function name clashes and makes it easy to compare the output of the two implementations.

json_encode

Encode a php value into a JSON formatted string.

See the description in the php manual for more details.

json_decode

Decode a a JSON formatted string into the corresponding php value(s). Returns NULL on error.

See the description in the php manual for more details.

$xmlrpc_internalencoding

The character set to which the JSON strings will be converted by `json_decode()`. Defaults to "UTF-8" for compatibility with the php json extension, but can be set to "ISO-8859-1" or "ASCII".

$json_extension_api_120_behaviour

The behaviour of the php json extension was changed in between versions. Up to version 1.2.0, calling `json_decode()` on strings representing scalar json values (ie. strings, booleans, numbers and null) not encapsulated within an array or struct would return false. From version 1.2.1 onwards, decoding single scalar json values is allowed. By setting the value of the global variable `$json_extension_api_120_behaviour` to true or false, the behaviour of the emulation layer will be altered to follow the different versions of the extension.

Chapter 9. Proxy package

files

- `proxyserver.php`: demo of a simple xmlrpc service acting as proxy to the webservices of phpxmlrpc.sf.net
- `proxyxmlrpc.inc`: xmlrpc proxy server class definition

usage

To be documented...

Chapter 10. WSDL package

files

- `xmlrpc.xsd`: a XML Schema Definition attempting to describe XMLRPC
- `xmlrpc.wsdl`: a XML Schema Definition attempting to describe XMLRPC method calls and responses
- `schema.rng`: a RELAX NG schema definition describing XMLRPC (using xml notation)
- `schema.rnc`: the same RELAX NG schema definition describing XMLRPC (using ... notation)

rationale

The XMLRPC protocol has been specified a long time ago, at the dawn of XML and is described only in natural language. Attempts have been made to "reverse specify" it with DTD or XSD, with little success. Both definition languages are in fact too limited in their capability of expressing valid xml constructs to capture in a complete and correct fashion the XMLRPC semantics (see http://www.cafeconleche.org/books/xmljava/chapters/ch02s05.html for more details. Suffice to say that XSD does not allow to specify two consecutive xml elements with the same name but of different types, such as the faultCode and faultReason response members happen to be).

The RELAX-NG definition language on the other hand has been found capable enough to fully describe XMLRPC, and the schema that is part of this distribution can be used to validate any XMLRPC request or response conforming to the spec. IT IS NOT AUTHORITATIVE because the XMLRPC specification is copyright by Dave Winer, and he seems to have lost interest in any further development of the protocol, but it should be considered accurate and production ready (if you find any bugs, please let the authors know, using the project's web site [???]).

This situation is very unfortunate, since WSDL has emerged as a widely accepted web services standard, and a plethora of tools exist to parse existing WSDL files and automatically generate code stubs or generate WSDL definitions from existing code. If a WSDL representation of XMLRPC webservices could be generated, the net result would be greatly inreased interoperability between SOAP clients and toolkits and XMLRPC server implementations.

Even though the apparent goal of the WSDL language design was to allow description of every possible message exchange, using every conceivable xml serialization over any transport (and a lot of the redundancy, verbosity and complication of the language are direct consequence of the flexibility sought), existing implementations de facto only support the SOAP binding for webservices, and XSD as language definition schema. RELAX-NG, while nominally accepted, has yet to find widespread usage in webservices toolkits. This leaves us with almost-bot-not-quite interoperable solutions.

Chapter 11. XMLRPC extension API package

files

- `xmlrpc_extension_api.inc`: contains the classes and functions that emulate the php XMLRPC extension API. `include()` this in your PHP files to have access to the required functionality
- `testsuite.php`: testsuite used to verify adherence of this implementation to the native extension

usage

To be documented...

API documentation

Please note that if you `include()` the `xmlrpc_extension_api.php` file on a php installation where the native XMLRPC extension is loaded, the functions provided by this implementation will be automatically renamed to `xmlrpc_alt_encode`, `xmlrpc_alt_decode`, etc. This prevents function name clashes and makes it easy to compare the output of the two implementations.

To be documented... suffice to say that you can read the documentation for the php xmlrpc extension on the php manual.