Zend_XmlRpc_Server Proposal - Matthew Weier O'Phinney

<ac:macro ac:name="unmigrated-inline-wiki-markup"><ac:plain-text-body><![CDATA[

<ac:macro ac:name="unmigrated-inline-wiki-markup"><ac:plain-text-body><![CDATA[

Zend Framework: Zend_XmlRpc_Server Component Proposal

Proposed Component Name	Zend_XmlRpc_Server
Developer Notes	http://framework.zend.com/wiki/display/ZFDEV/Zend_XmlRpc_Server
Proposers	Matthew Weier O'Phinney
Revision	$Id:$ (wiki revision: 4)

Table of Contents

1. Overview
2. References
3. Component Requirements, Constraints, and Acceptance Criteria
4. Dependencies on Other Framework Components
5. Theory of Operation
6. Milestones / Tasks
7. Class Index
8. Use Cases
9. Class Skeletons

1. Overview

Zend_XmlRpc_Server provides a flexible, performant, UTF-8 aware, and easily implemented XMLRPC server following PHP's SOAP Server API. It features the ability to attach functions, static methods, and object instance methods as XMLRPC method callbacks; the ability to cache a server definition to accelerate future calls to the server; and auto-creation of method help text and signatures via introspection of the callbacks used.

All required system.* methods are implemented in the server, including system.listMethods, system.methodHelp, and system.methodSignature. Additionally, the system.multicall method has been included to allow method boxcarring in a single request.

This component was developed internally at Zend prior to the original preview releases and as such has not followed the now established community process. It is being opened to public comment after initial commit to the incubator.

2. References

http://www.xmlrpc.com/spec http://www.xmlrpc.com/discuss/msgReader$1208 http://php.net/soap

3. Component Requirements, Constraints, and Acceptance Criteria

*PHP5
**PHP5 Reflection extension
**PHP5 DOM extension
**PHP5 SimpleXML extension
*Zend Framework
**Zend_XmlRpc_Value family of classes
**Zend class

4. Dependencies on Other Framework Components

- Zend_Exception
- Zend_XmlRpc_Value
- Zend

5. Theory of Operation

Zend_XmlRpc_Server allows the developer to attach any combination of functions, classes, and objects to the server as XMLRPC method callbacks, each with an optional namespace. The developer need only provide the function name, class name, or object to the appropriate methods (addFunction() and setClass()); the resource is then examined via PHP 5's Reflection API to determine the signature and method help (if a docblock is present). (Note: this introspection has been moved to Zend_Server_Reflection.)

Method boxcarring is allowed via the system.multicall method.

After methods are attached, the handle() method should be called. It accepts an optional request object representing the XMLRPC request; if none is passed, the server instantiates a Zend_XmlRpc_Request_Http object to retrieve it via php://input. Request objects should provide accessors for retrieving the method (getMethod()) and arguments (getParams()).

A response object is returned by handle(); the response object will be either a Zend_XmlRpc_Response or a Zend_XmlRpc_Server_Fault, either of which may be directly echoed in order to generate the XML response, or further manipulated by the instance script (for instance, to log returns). The developer may set the type of response object returned by using setResponseType(); by default, a Zend_XmlRpc_Response_Http is returned, which will set the appropriate HTTP content type when echoed.

Optionally, the developer may cache the dispatch table for future invocations, thus eliminating the need to attach and inspect method callbacks. Zend_XmlRpc_Server_Cache provides basic functionality for this, currently serializing the dispatch table and saving it to disk. When caching, one must either load all classes that could be called, or rely on __autoload().

By default, only Zend_XmlRpc_Server_Exceptions may be used to provide messages and codes for use with fault responses. However, the developer may register additional exception types for use with faults using Zend_XmlRpc_Server_Fault::attachFaultException(). Additionally, the developer may observe faults for the purpose of logging or messaging by attaching observers via Zend_XmlRpc_Server_Fault::attachObserver().

6. Milestones / Tasks

zone: Missing {zone-data:milestones}

7. Class Index

- Zend_XmlRpc_Fault
- Zend_XmlRpc_Request
- Zend_XmlRpc_Request_Http
- Zend_XmlRpc_Request_Stdin
- Zend_XmlRpc_Response
- Zend_XmlRpc_Response_Http
- Zend_XmlRpc_Server
- Zend_XmlRpc_Server_Cache
- Zend_XmlRpc_Server_Exception
- Zend_XmlRpc_Server_Fault

8. Use Cases

```
require_once 'Zend/XmlRpc/Server.php';
    require_once 'Zend/XmlRpc/Server/Cache.php';
    require_once 'Zend/XmlRpc/Server/Fault.php';
    require_once 'My/Exception.php';
    require_once 'My/Fault/Observer.php';

    // Instantiate server
    $server = new Zend_XmlRpc_Server();

    // Allow some exceptions to report as fault responses:
    Zend_XmlRpc_Server_Fault::attachFaultException('My_Exception');
    Zend_XmlRpc_Server_Fault::attachObserver('My_Fault_Observer');

    // Get or build dispatch table:
    if (!Zend_XmlRpc_Server_Cache::get($filename, $server)) {
        require_once 'Some/Service/Class.php';
        require_once 'Another/Service/Class.php';

        // Attach Some_Service_Class in 'some' namespace
        $server->setClass('Some_Service_Class', 'some');

        // Attach Another_Service_Class in 'another' namespace; use only static
        // methods
        $server->setClass('Another_Service_Class', 'another', false);

        // Create dispatch table cache file
        Zend_XmlRpc_Server_Cache::save($filename, $server);
    }

    // Handle request
    echo $server->handle();
```

9. Class Skeletons

zone: Missing {zone-data:skeletons}
]]></ac:plain-text-body></ac:macro>
]]></ac:plain-text-body></ac:macro>