Zend_Service_Server Proposal - Davey Shafik

<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_Service_Server Component Proposal

Proposed Component Name	Zend_Service_Server
Developer Notes	http://framework.zend.com/wiki/display/ZFDEV/Zend_Service_Server
Proposers	Davey Shafik
Revision	1.0 - 1 August 2006: Intial Proposal. (wiki revision: 6)

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
Front facing file
testClass.php
Example Exchanges
Request
Response
Request
Response
Request
Response
9. Class Skeletons

1. Overview

An automated SOAP/XML-RPC/REST server that will expose any code using any of the above protocols. Zend_Service_Server will also generate human-readable documentation.

2. References

Initial idea is based on my original efforts on WSDL generation for Cerebral Cortex. It has since come to my attention, there is also (again, SOAP specific) PEAR's Net_Service.

- Crtx_SOAP_AutoDiscover
- PEAR::Service_Webservice

3. Component Requirements, Constraints, and Acceptance Criteria

Must allow the user to expose any class with no special requirements placed on the author. The only exception to this is that all methods must return values, they cannot simply output them.

All servers must provide a SoapServer-like interface allowing for instantiation outside of this automated process and registering of both single class and multiple functions. They *should* also provide a way to generate documentation specific to the implementation.

4. Dependencies on Other Framework Components

- Zend_Uri
- Zend_XmlRpc_Server (not yet created)
- Zend_Service_Exception

5. Theory of Operation

Zend_Service_Server will require that a file containing a class be given as the constructor argument. This is required because the tokenizer extension is used for PHP 5.0.x compatibility (though, I forget the /exact/ problem with ext/reflection that prompted this decision).

The operation from here depends on the GET arguments.

The operations allowed are as follows:

?wsdl = WSDL Generation
?soap = SOAP Server (using ext/soap's SoapServer)
?rest = REST Server (using Zend_Service_Server_Rest)
?rpc = XML-RPC Server (using Zend_XmlRpc_Server)

Passing no argument will output user-readable documentation, detailing the methods exposed. If you supply one of the above **and** the 'docs' argument, it will output implementation specific documentation, and possibly allow for submission of requests using a form (a la .NET)

6. Milestones / Tasks

zone: Missing {zone-data:milestones}

7. Class Index

- Zend_Service_Server - Class that handles user request and forwards request to implementation specific server

- Zend_Service_Server_Interface - Interface that all (except ext/soap) must implement to be accepted as backends.
- Zend_Service_Server_Rest - REST server; this will either create a nice markup based on reflection, or will allow the user to return SimpleXML, DomDocument or DomNode objects.
- Zend_Service_Server_Soap_WsdlCreator - Generates WSDL files for classes automatically.
- Zend_Service_Server_Wsdl - WSDL Toolkit for creating WSDL files (not automated)
- Zend_Service_Server_Wsdl_Parser - WSDL Parser
- Zend_Service_Server_Wsdl_Parser_Result - WSDL Parser Result container
- Zend_Service_Server_Wsdl_CodeGenerator - WSDL -> PHP Skeleton creator

In addition to these, there will be the documentation creators:

- Zend_Service_Server_Documentor
- Zend_Service_Server_Documentor_REST
- Zend_Service_Server_Documentor_XmlRpc
- Zend_Service_Server_Documentor_Soap

Furthermore, depending on the implementation of Zend_XmlRpc_Server there may be a requirement to wrap it with a Zend_Service_Server_XmlRpc class to make it conform to the Zend_Service_Server_Interface API.

Finally, there will need to be a REST client class, whether this is implemented within Zend_Service_Rest or as another entity is undecided.

8. Use Cases

The ultimate use-case for this is anybody wishing to provide webservices to their users and not wanting to be tied to a specific technology (i.e. SOAP). It also allows for additional backends to be plugged in at will for custom formats.

Front facing file

```
<?php
require_once 'Zend/Service/Server.php';

$server = new Zend_Service_Server('./testClass.php');

$server->handle();
?>
```

testClass.php

```
<?php
class TestClass {
 /**
  * Class Constructor
  */
 public function __construct()
 {
  // Some Constructor
 }

 /**
  * Say Hello to Somebody
  *
  * @return string
  */
 public function sayHello($who, $when)
 {
  return "Hello $who, Good $when";
 }

 /**
  * Something to return an array
  *
  * @return array
  */
 public function returnArray()
 {
  return array("foo" => "bar", "baz" => "bat");
 }


 private function doSomething()
 {
  // Some Private Method
 }
}
?>
```

Example Exchanges

Request

```
?rest&method=SayHello&wHo=Davey&WHEN=Day
```

Response

```
<TestClass>
    <sayHello>
        <response>
            Hello World, Good Day
        </response>
        <status>
            success
        </status>
    </sayHello>
</TestClass>
```

Request

```
?rest&method=returnArray
```

Response

```
<TestClass>
    <returnArray>
        <foo>bar</foo>
        <baz>bat</baz>
        <status>success</status>
    </returnArray>
</TestClass>
```

Request

```
?wsdl
```

Response

```


<?xml version="1.0"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:tns="http://example.org/Zend/Services/example.php?soap"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap-enc="http://schemas.xmlsoap.org/soap/encoding/"
name="TestClass"
targetNamespace="http://example.org/tests/Zend/Services/example.php?soap">

  <documentation>Zend_Service_Server Test Class</documentation>
  <portType name="TestClassPort">
    <operation name="sayHello">
      <input message="tns:sayHelloRequest" />
      <output message="tns:sayHelloResponse" />
    </operation>
    <operation name="returnArray">

      <input message="tns:returnArrayRequest" />
      <output message="tns:returnArrayResponse" />
    </operation>
  </portType>
  <binding name="TestClassBinding" type="tns:TestClassPort">
    <soap:binding style="rpc"
    transport="http://schemas.xmlsoap.org/soap/http" />
    <operation name="sayHello">
      <input>
        <soap:body use="encoded"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />

      </input>
      <output>
        <soap:body use="encoded"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </output>
      <soap:operation soapAction="http://example.org/Zend/Services/example.php?soap#sayHello" />
    </operation>
    <operation name="returnArray">
      <input>
        <soap:body use="encoded"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
```

```
      </input>
      <output>
        <soap:body use="encoded"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </output>
      <soap:operation soapAction="http://example.org/Zend/Services/example.php?soap#returnArray" />
    </operation>
  </binding>
  <service name="TestClassService">
    <port name="tns:TestClassPort" binding="tns:TestClassBinding">

      <soap:address location="http://example.org/Zend/Services/example.php?soap" />
    </port>
  </service>
  <message name="sayHelloRequest">
    <part name="longdesc" type="xsd:anyType" />
    <documentation>Say Hello to Somebody</documentation>
  </message>
  <message name="sayHelloResponse">

    <part name="sayHelloReturn" type="xsd:string" />
  </message>
  <message name="returnArrayRequest">
    <part name="longdesc" type="xsd:anyType" />
    <documentation>Something to return an array</documentation>
  </message>
  <message name="returnArrayResponse">
    <part name="returnArrayReturn" type="soap-enc:Array" />
```

```
  </message>
</definitions>
```

9. Class Skeletons

```
class Zend_Service_Server {
    public function __construct() { }

    public function handle() { }
}
```

```
interface Zend_Service_Server_Interface {
 public function addFunction($funcname);

 public function getFunctions();

 public function handle();

 public function setClass($class);
}
```

```
class Zend_Service_Rest implements Zend_Service_Server_Interface { }
```

```
class Zend_Service_Server_Wsdl {
    public function __construct() { }

    public function addBinding() { }

    public function addBindingOperation() { }

    public function addDocumentation() { }

    public function addMessage() { }

    public function addPortOperation() { }

    public function addPortType() { }

    public function addService() { }

    public function addSoapBinding() { }

    public function addSoapOperation() { }

    public function addTypes() { }

    public function dump() { }

    public function toDomDocument()  { }

    public function toXML() { }
}
```

```
class Zend_Service_Server_Wsdl_Parser {
    static public function getDocs() { }

    static public function getOperationInputs() { }

    static public function getOperationOutput() { }

    static public function parse() { }
}
// The static-ness of this class is left over from its previous implementation in
// Cerebral Cortex and could be removed quite easily.
```

```
class Zend_Service_Server_Wsdl_Parser_Result { }
// Contains on named properties for accessing parser results
```

```
class Zend_Service_Server_Wsdl_CodeGenerator {
    static public function parse($wsdl, $output_file = null) { }

    static private generatorPhp() { }
}
```

```
class Zend_Service_Server_Soap_WsdlCreator {
    public function __construct($code_file, $service_uri, $output_file = null, $ignore_methods =
array()) { }

    public function createWsdl() { }

    static public function getType() { }

    private function saveToFile() { } // Should be made public and $output_file removed from
constructor...

    public function toXML() { }
}
```

Documentation classes are seen as specialized HTML responding servers.

```
class Zend_Service_Server_Documentor implements Zend_Service_Server_Interface { }
```

```
class Zend_Service_Server_Documentor_Rest extends Zend_Service_Server_Documentor { }
```

```
class Zend_Service_Server_Documentor_Soap extends Zend_Service_Server_Documentor { }
```

```
class Zend_Service_Server_Documentor_XmlRpc extends Zend_Service_Server_Documentor { }
```

ToDo:

REST client that works with Zend_Service_Server_Rest automatically.
]]></ac:plain-text-body></ac:macro>
]]></ac:plain-text-body></ac:macro>