Jimmy Zimmerman

APIs, Family History, Star Wars, and Other Random Writings

SOAP Server with PHP5 – part 2: fun with wsdl

Part 1 of this series of blog posts should help you get started with setting up a simple soap server with one function, a stockquote request. It points you to the soap extension tutorial on the Zend Developer Zone.

In this part, I'm going to explain some of the different parts of the wsdl so that the wsdl isn't so intimidating. So let's begin by examining the wsdl found on the tutorial here.

I'm going to start from the bottom of the wsdl and work my way to the top.

The 'service' block

```
<service name='StockQuoteService'>
<port name='StockQuotePort' binding='StockQuoteBinding'>
<soap:address location='http://[insert real path here]/server1.php'/>
</port>
</service>
```

This 'service' block of the wsdl gives some basic definitions for your soap server. You'll want to pick a service name and stay consistent throughout the wsdl. This example uses 'StockQuote' for all its naming but you could use something like 'Weather' or 'Inventory.' Just stay consistent. So, if you were to make an 'Inventory' service, your service block of the wsdl would look something like this:

```
<service name='InventoryService'>
<port name='InventoryPort' binding='InventoryBinding'>
```

```
<soap:address location='http://[insert real path here]/server1.php'/>
</port>
</service>
```

The port name and binding connect this block with the other parts of the wsdl, so just keep to your naming convention and you'll be fine.

The address location points to the location of what I call the "glue code" or the php script that glues your function to the soap extensions server libraries. The glue code is what handles all of your xml messages and passes the right parameters to your functions.

The 'binding' block

```
<binding name='StockQuoteBinding' type='tns:StockQuotePortType'>
<soap:binding style='rpc'
transport='http://schemas.xmlsoap.org/soap/http'/>
<operation name='getQuote'>
<soap:operation soapAction='urn:xmethods-delayed-quotes#getQuote'/>
<input>
<soap:body use='encoded' namespace='urn:xmethods-delayed-quotes'
encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'/>
</input>
<output>
<soap:body use='encoded' namespace='urn:xmethods-delayed-quotes'
encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'/>
</output>
</operation>
</binding>
```

The 'binding' block has a lot of stuff in it, but most of it is just defining how the soap client and server will communicate. There is very little code here that you'll need to change to fit your service. There are 4 parts that you'll need to modify in order to map the service to the function that you wish to put in the service. Those parts have been bolded and italicized above. Change the binding

name and type, the operation functionName and the #functionName at the end of the soap:operation.

So, if we were to modify this for our inventory example we might change the getQuote to getItemCount and our binding block would be as follows:

```
<binding name='InventoryBinding' type='tns:InventoryPortType'>
<soap:binding style='rpc'
transport='http://schemas.xmlsoap.org/soap/http'/>
<operation name='getItemCount'>
<soap:operation soapAction='urn:xmethods-delayed-quotes#getItemCount'/>
<input>
<soap:body use='encoded' namespace='urn:xmethods-delayed-quotes'
encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'/>
</input>
<output>
<soap:body use='encoded' namespace='urn:xmethods-delayed-quotes'
encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'/>
</output>
</operation>
</binding>
```

If you were to add another function to your service, you would just copy the 'operation' section of this block, and paste it right below, and change the function name. Pretty simple.

The 'portType' block

```
<portType name='StockQuotePortType'>
<operation name='getQuote'>
<input message='tns:getQuoteRequest'/>
<output message='tns:getQuoteResponse'/>
</operation>
</portType>
```

The ‘portType’ block connects your binding block to the message blocks which define your method parameter and return types. The name should match the type from the binding block. You’ll also need to change the operation name and the input and output message attributes. The message attribute can be named however you wish as long as it matches the name of its corresponding message block, which we’ll talk about next.

For the inventory example, we’d change our portType block to the following:

```
<portType name='InventoryPortType'>
<operation name='getItemCount'>
<input message='tns:getItemCountRequest'/>
<output message='tns:getItemCountResponse'/>
</operation>
</portType>
```

Again, if you wish to add another method to your service, you would just have two operation blocks within the portType block.

The ‘message’ block(s)

```
<message name='getQuoteRequest'>
<part name='symbol' type='xsd:string'/>
</message>
<message name='getQuoteResponse'>
<part name='Result' type='xsd:float'/>
</message>
```

Here we have 2 message blocks that correspond to the message attributes found in the portType block of the stockquote.wsdl. This is where the meat of your wsdl resides.

The part element in the document specifies parameters for the requests messages, and the return structure for the variable returned by your function. For parameters, make sure you mape the name with the parameter name, and

specify the type. Here is a list of primitive data types accepted: http://www.w3.org/TR/xmlschema-2/#built-in-primitive-datatypes.

For our inventory example, we might modify this section like so:

```
<message name='getItemCountRequest'>
<part name='upc' type='xsd:string'/>
</message>
<message name='getItemCountResponse'>
<part name='Result' type='xsd:integer'/>
</message>
```

The 'definitions' header

The definitions header is what defines the namespaces of your wsdl document. You can mostly leave this as it is from the example, but you should change a few parts. Here is how I've modified my inventory service header:

```
<definitions name='Inventory'
targetNamespace='urn:JimmyzInventory'
xmlns:tns='urn:JimmyzInventory'
xmlns:soap='http://schemas.xmlsoap.org/wsdl/soap/'
xmlns:xsd='http://www.w3.org/2001/XMLSchema'
xmlns:soapenc='http://schemas.xmlsoap.org/soap/encoding/'
xmlns:wsdl='http://schemas.xmlsoap.org/wsdl/'
xmlns='http://schemas.xmlsoap.org/wsdl/'>
```

And that's pretty much it. Creating your own wsdl isn't really as bad as it appears.

My final inventory wsdl is located here.

See part 3 for implementing the web service.

Jimmy Z / February 20, 2007 / Web Services / php5, soap, wsdl

2 thoughts on “SOAP Server with PHP5 – part 2: fun with wsdl”

Edwilson

June 1, 2012 at 6:06 am

Link for part 3 is broken as link from part 1 to part 2..

Gerrit

October 12, 2012 at 4:44 am

Nice tutorial. Soap is documented far too less on the web and in the PHP docs.
The link to the Soap extension tutorial on the zend webpages is broken.
I get a 404 error. Probably it is replaced by a new framework with different urls because i found this link : http://devzone.zend.com/25/php-soap-extension/

Comments are closed.