SOAP Servers in PHP

Disposition

- Necessary steps when constructing a simple server
- Example server
- Error handling
- Returning the WSDL to clients
- Necessary steps when constructing a complex server
- Mapping complex types
- Example server
- Concluding remarks

SOAP Servers in PHP

Necessary steps when constructing a simple server

1. Write the WSDL
2. Write the service class
3. Add this class to the server: $server->setClass(“class”);
4. Activate the server: $server->handle()

SOAP Servers in PHP

```
<?xml version ='1.0' encoding ='UTF-8' ?>
<definitions name='SayHello'  targetNamespace='urn:SayHello'
 xmlns:tns='urn:SayHello'
  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/'>

<message name='getHelloRequest'>
  <part name='name' type='xsd:string'/>
</message>
<message name='getHelloResponse'>
  <part name='Result' type='xsd:string'/>
</message>

<portType name='SayHelloPortType'>
  <operation name='getHello'>
    <input message='tns:getHelloRequest'/>
    <output message='tns:getHelloResponse'/>
  </operation>
</portType>
```

SOAP Servers in PHP

```
<binding name='SayHelloBinding' type='tns:SayHelloPortType'>
  <soap:binding style='rpc'
    transport='http://schemas.xmlsoap.org/soap/http'/>
  <operation name='getHello'>
    <soap:operation soapAction='urn:SayHello#getQuote'/>
    <input>
      <soap:body use='encoded' namespace='urn:SayHello'
        encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'/>
    </input>
    <output>
      <soap:body use='encoded' namespace='urn:SayHello'
        encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'/>
    </output>
  </operation>
</binding>

<service name='SayHelloService'>
  <port name='SayHelloPort' binding='SayHelloBinding'>
    <soap:address location='http://localhost/~mir/soap/php5/day5/helloserver.php'/>
  </port>
</service>
</definitions>
```

SOAP Servers in PHP

Simple server

```
<?php
    class SayHello {
        public function getHello($inmessage) {
            return "Hello $inmessage!";
        }
    } //end class

    ini_set("soap.wsdl_cache_enabled", "0");
    $server = new SoapServer("SayHello.wsdl");
    $server->setClass("SayHello");
    $server->handle();
?>
```

SOAP Servers in PHP

Error handling

```
$faultcode = 'The actor coursing the fault [client | server |
            intermediary]';
$faultstring =  'Some human readable text';
$faultactor = 'The namespace';
$detail = 'Some machine readable text.';

throw new SoapFault(
            $faultstring,
            $faultcode,
            $faultactor,
            $detail
);
```

SOAP Servers in PHP

Returning the WSDL

```
if (isset($_SERVER['REQUEST_METHOD'])  &&
        $_SERVER['REQUEST_METHOD']=='POST') {
        $soapserver->handle();
}
else {
        if (isset($_SERVER['QUERY_STRING']) &&
                strcasecmp($_SERVER['QUERY_STRING'],'wsdl') == 0) {
                // Return the WSDL
                $wsdl = @implode ('', @file ('some.wsdl'));
                if (strlen($wsdl) > 1) {
                        header("Content-type: text/xml");
                        echo $wsdl;
                }
                else {
                        header("Status: 500 Internal Server Error");
                        header("Content-type: text/plain");
                        echo "HTTP/1.0 500 Internal Server Error";
                }
        else {Some other non-related SOAP error}
}
```

SOAP Servers in PHP

Necessary steps when constructing a complex server

1. Write the WSDL for the service
2. Map complex types from WSDL
3. Write the service class
4. Add this class to the server: $server->setClass(“class”);
5. Activate the server: $server->handle()

SOAP Servers in PHP

The WSDL

```
<types xmlns="http://schemas.xmlsoap.org/wsdl/">
      <schema xmlns="http://www.w3.org/2001/XMLSchema"
            targetNamespace="urn:HostInfo">
            <complexType name="hostInfo">
                  <sequence>
                        <element name="hostname" type="xsd:string" />
                        <element name="ip" type="xsd:string" />
                        <element name="port" type="xsd:int" />
                        <element name="datetime" type="xsd:string" />
                        <element name="PHP_version" type="xsd:string" />
                  </sequence>
            </complexType>
      </schema>
</types>

<message name='getHostInfoRequest' />

<message name='getHostInfoResponse'>
  <part name='Result' type='tns:hostInfo'/>
</message>
```

SOAP Servers in PHP

Map complex types

- Create an associative array for each complex type
- Use the class SoapParam to convert the array to the soap type specified in the WSDL: SoapParam(mixed data, 'type name from WSDL')

```
$HostInfo = array(
      'hostname' => “”,
      'ip' => “”,
      'port' => “”,
      'datetime' => “”,
      'PHP_version' => “”
);
return new SoapParam ( $HostInfo, 'hostInfo');
```

SOAP Servers in PHP

Write the service class

```
class HostInfo {
    public function getHostInfo() {
        $response = array(
            'hostname' => $_SERVER['SERVER_NAME'],
            'ip' => $_SERVER['SERVER_ADDR'],
            'port' => $_SERVER['SERVER_PORT'],
            'datetime' => date("l dS of F Y H:i:s T"),
            'PHP_version' => PHP_VERSION
        );
        return new SoapParam ( $response, 'hostInfo');
    }
}
```

SOAP Servers in PHP

Add class to server and activate server

```
ini_set("soap.wsdl_cache_enabled", "0");
$server = new SoapServer("HostInfo.wsdl");
$server->setClass("HostInfo");
if (isset($_SERVER['REQUEST_METHOD'])  &&
        $_SERVER['REQUEST_METHOD']=='POST')
    $server->handle();
else {
    $wsdl = @implode ('', @file ('HostInfo.wsdl'));
    if (strlen($wsdl) > 1) {
        header("Content-type: text/xml");
        echo $wsdl;
    }
}
```

@: suppresses error and warnings

SOAP Servers in PHP

Concluding remarks

- All input parameters will be of type stdObject
- Output of any kind is not permitted
- Always provide WSDL for clients
- Input and output are passed in clear text – XML
- If security is an issue use HTTPS – requires the curl module in PHP
- Encryption is not a part of the standard yet, but work is in progress
- Always use SoapParam to return complex types
- Provide a class for each usecase – simplifies debugging and gives higher maintainability