PHP & Web Services / PHP & XML

Unix IDs:
baxter
muhammad
sonu
walberg

PHP & Web Services

PHP is an ideal language to implement and use a web service as the available packages simplify the implementation details. The syntax is similar to Perl, so it should not be difficult to read and understand the sample code provided in this tutorial. While PHP does not bundle classes/packages/modules to make use of SOAP, two major SOAP implementations exist that can be downloaded and easily installed. They are: NuSOAP, and PHP:PEAR::SOAP. PEAR refers to the "**P**HP **E**xtension and **A**pplication **R**epository." It is a distribution system for reusable Open-Source PHP classes/packages/modules.

This example based report is divided into 3 major parts:

- Server/Client using NuSOAP
- DOM Model XML parsing
- SAX/Event Based XML Parsing

NuSOAP – A SOAP Implementation for PHP

You can download both the toolkit and the documentation for NuSOAP at this link: http://dietrich.ganx4.com/nusoap/index.php

One of the nice features of using the NuSOAP module is the built-in WSDL support. Recall that in Java, a deployment descriptor XML file is required to describe the web

service and register it for world use, even when the .wsdl file is provided. In NuSOAP, this step is conveniently avoided. NuSOAP allows the .wsdl file to be loaded and registered in very few lines of code, which we will explain later on.

Server Using NuSOAP

The following code is an example of a server built in PHP that returns the GST for a given amount. It is assumed that the nusoap.php package has been downloaded and installed, and that the web server is configured to handle PHP scripts.

```
1.  <?
2.  require_once("nusoap.php");
3.  $ns="http://www.yourserver.com/";
4.  $server = new soap_server();
5.  $server->configureWSDL('CanadaTaxCalculator',$ns);
6.  $server->wsdl->schemaTargetNamespace=$ns;
7.  $server->register('CalculateTax',
8.  array('amount' => 'xsd:string'),
9.  array('return'=>'xsd:string'),
10. $ns);
11. function CalculateTax($amount){
12.   $taxcalc=$amount*.07;
13.   return new soapval('return','xsd:string',$taxcalc);
14. }
15. $server->service($HTTP_RAW_POST_DATA);
16. ?>
```

Code Explanation:

We assume that the reader has general programming experience and is able to understand to understand the code flows, functions and other such coding basics in this example as well as all that follow.

Lines 1 and 16 denote the start and end of a PHP script.

Line 2 includes the NuSOAP package.

Line 3 designates the URI for the web service.

Lines 4-6 create a new instance of a soap_server object, and configure the service name and namespace for the wsdl.

Lines 7-10 make the server aware of the function/method 'CalculateTax' which takes in a string and returns a string.

Lines 11-14 define the 'CalculateTax' method. Notice the return value.

Line 15 simply invokes the service.

Once the above code is deployed on the server, the service springs to life.

NuSOAP – Built-in WSDL Support:

As mentioned earlier, the built-in WSDL support is where NuSOAP's strength can be seen. With any server built using NuSOAP and PHP, adding "?wsdl" to the end of the server's URL will dynamically generate and display the WSDL. Assuming the above code was saved to a file called server.php and we were running it on yourserver.com, the WSDL can easily be displayed by opening a web browser and navigating to the following URL:

http://www.yourserver.com/server.php?wsdl

Client using NuSOAP

Creating a client using NuSOAP is very straightforward because NuSOAP's built-in WSDL support simplifies the task.

```
1. <?
2. require_once('nusoap.php');
3. $wsdl = "http://www.yourserver.com/server.php?wsdl";
4. $client=new soapclient($wsdl, 'wsdl');
5. $param=array('amount'=>'15.00',);
6. echo $client->call('CalculateTax', $param);
7. ?>
```

Code Explanation:

Lines 1 and 7 denote the start and end of the PHP script.

Line 2 includes the NuSOAP package.

Line 3 assigns a variable to a WSDL. Note that the WSDL is dynamically generated by appending "?wsdl" to the server's URL.

Line 4 creates a new instance of a soap client. The client object is given the WSDL describing the service.

Line 5 creates the parameter to be passed to the web service. In this example, the client is requesting the web service to return the GST value for a $15.00 purchase (This can easily be dynamic of course, like getting the value from a form. We have kept it simple as this is designed as a basic introduction).

Line 6 makes a call to the web service and displays the return result via the 'echo' call.

As the examples above demonstrate, creating servers and clients for web services is very straightforward in PHP using a package such as NuSOAP. Implementation is not as tedious as in Java, and extra web-server components such as Tomcat are required.

PHP & XML

Similar to the style above, we have taken an example based approach to this section as well. Note that what follows assumes basic knowledge of PHP. For an introduction to PHP, we recommend you look at the group that covered the basics of PHP as their main presentation topic.

DOM Based XML Parsing

PHP 5 has a built in extension that allows XML parsing using the DOM model. This extension fully conforms to the World Wide Web Consortium's standards (Hooray for standards compliance). One down side to looking at PHP 5 with respect to PHP functionality is that although PHP in general aims to be backwards compatible, there are however many issues in the new release when it comes to compatibility with older versions (something to keep in mind if you are working on PHP 4x). DOM parsing is one of the casualties, and thus some of the following code may not work under anything older than PHP 5.

The first step to manipulating XML in PHP using the DOM model is to create a DomDocument Object.

```
$dom = new DomDocument();
```

With this object created you can then give it an XML file to use as a source.

```
$dom->load("xmlfile.xml");
```

Now of course, when dealing with XML parsing, validation is an important concern. PHP certainly has not let us down in this respect. Validation can be done using DTD, XML Schema, and RelaxNG documents (another XML schema language, see details at www.relaxng.org). Simple one line functions can validate based on these documents and return boolean values based on the outcome of the validation.

```
//using the DomDocument created above
$dom->validate("mydtd.dtd");
$dom->schemaValidate("myschema.xsd");
$dom->relaxNGValidate("myRNG.rng");
```

As well as the boolean return values, any errors are also returned as PHP warnings. Now that we have a DomDocument loaded with a valid XML document, we want to be able to traverse it. This can be done in two different ways (on top of the simple standard way of iterating over the tags/nodes of the document one by one):

1. Getting a list of all nodes that correspond to a named Tag

   ```
   $mytags = $dom->getElementsByTagName("myTag");
   ```

2. Getting a unique node

   ```
   $myID = $dom->getElementByID("myID");
   ```

Accessing child nodes for any unique node is simple as well:

```
$children = $myNode->childNodes;
```

Another way in which the PHP DomDocument is very useful is its ability to also write out XML files. This functionality can be used to easily create and/or modify XML documents. To add new elements to an XML Document:

```
$myElement = $dom->createElement("myElement");
```

```
$myText = $dom->createTextNode("A Text Node");
$myElement->appendChild($myText);
$dom->documentElement->appendChild($myElement);
```

In the code snippet above, the nodes are created and chained together, and then the new element is inserted into the root element of the document.

Documents (XML files) created or modified using the two codes above can then be generated in two different ways.

1. Either as direct output (to stdout or a browser)

   ```
   print $dom->saveXML();
   ```

2. Or output to a file

   ```
   print $dom->save("myXMLfile.xml");
   ```

DOM Model - Benefits

The benefits of the DOM model in PHP can be summarized as follows:

- Ease of use, as can be seen from the code above. The code is fairly straight forward; much can be accomplished with minimum lines of code
- Provides a simple interface for a variety of tasks such as validation, querying, and modifying XML
- In memory parsing implies fast, non-sequential access
- Handy for visualizing data and transforming data on the fly

DOM Model - Drawbacks

The major drawbacks include:

- Parallel parsing of large documents can hog a lot of system memory

- Building the in-memory tree representation of the document takes time
- This implementation does not support partial parsing of XML documents

SAX/Event Based XML Parsing

PHP includes an event-based XML parser as well. As with the DOM parser, its use is straight forward and simple to implement. We will describe the use of the event based parser with an example that also takes advantage of PHP's object oriented features.

RSS: RSS (Really Simple Syndication, Rich Site Summary, take your pick) is a XML based document that is common for headline news aggregation. Such feeds are available from a wide variety of sources such as BBC, CNN, Disney, Slashdot, etc. The RSS document describes and summarizes content of a web page(s) in distinct channels.

RSS Example:

```
<rdf:RDF>
  <channel rdf:about="http://www.sitepoint.com/rss.php">
    <title>SitePoint.com</title>
    <description>Master the Web!</description>
    <link>http://www.sitepoint.com/</link>
    <items>
      <rdf:Seq>
        <rdf:li
rdf:resource="http://www.PromotionBase.com/article/551"/>
       <rdf:li
rdf:resource="http://www.WebmasterBase.com/article/541"/>
      </rdf:Seq>
    </items>
  </channel>
  <item rdf:about="http://www.PromotionBase.com/article/551">
    <title>Escape Search Engine Caching</title>
    <description>Did you know that many search engines cache your
pages? While this practice can speed up a search, users might not
see your most recent site updates! Ralph shows how you can stop
search engines caching your pages.
    </description>
    <link>http://www.PromotionBase.com/article/551</link>
```

```
  </item>
  .
  .
  .
  <item rdf:about="http://www.WebmasterBase.com/article/541">
    <title>Add JavaScript to Fireworks</title>
    <description>Does your design need more pizazz? Add
interactivity to your site without learning JavaScript! Matt
explains the creation of JavaScript effects in Fireworks, and
explores in detail the use of this program's tools.
    </description>
    <link>http://www.WebmasterBase.com/article/541</link>
  </item>
</rdf:RDF>
```

Now if you wanted to display the title, description, and link on your web site from one of these RSS feeds, you cannot just display it as is, since most browsers are not very discriminating in what they display when encountering XML. Also note that you may be fetching multiple feeds every hour, every 10 minutes, or some other time frame so you do not want to expend too much effort in extracting the information you want. This being the case, you would not need or even want a DOM based parser for this purpose as the overhead would be excessive and you would be parsing every single tag, including the ones you know you do not need. The Event based XML Parser allows us to efficiently parse and look at selective tags within the document, thus greatly minimizing the server load and speeding up the process.

Following is an intermediate object oriented agent built with PHP that could be used to fetch many RSS feeds and simply display the title, description and link.

Our first step is to create a class that will be used to keep track of where we are in the XML document and output the information that we want. This class is based on the RSS

feed shown as an example above and will be parsing the title, description and link from that RSS feed.

```
Class RSSParser{
	//these variable will be used to test if we are in an ITEM
tag
	//and if so hold the pertinent information
	var $insideitem = false;
	var $tag = “”;
	var $title = “”;
	var $description = “”;
	var $link = “”;

	function startElement($parser, $tagName, $attrs){
		if($this->insideitem){
			$this->tag = $tagName;
		}
		elseif($tagName == “ITEM”){
			$this->insideitem = true;
		}
	}

	function characterData($parser, $data){
		if($this->insideitem){
			switch($this->tag){
				case “TITLE”:
					$this->title .= $data;
					break;
				case “DESCRIPTION”:
					$this->description .= $data;
					break;
				case “LINK”:
					$this->link .= $data
					break;
			}
		}
	}

	function endElement($parser, $tagName){
		if($tagName == “ITEM”){
			printf(“<p><b><a href=’%s’>%s</a></b></p>”,
				trim($this->link),
				htmlspecialchars(trim($this->title)));
			printf(“<p>%s</p>”,
				htmlspecialchars(trim($this-
			>description)));
			$this->title = “”;
			$this->description = “”;
```

```
            $this->link = “”;
            $this->insideitem = false;
        }
    }
}
```

Without knowing yet how these functions are called it is fairly simple to see what they do. The `startElement` function tests to see if we are in an “**ITEM**” tag and if so, records which tag within an item we are currently in. The `characterData` function fills our class variables when we are inside an “ITEM” tag with the proper values based on the tag. Finally the `endElement` function prints the class variables in HTML when we exit an ITEM tag. Notice how this class can be easily overhauled to parse any other actions based on the XML that you are parsing. If you were to do so, the process shown below would be identical once your class (as the example above) is written.

Now we shall see how this class (or any other) is used.

```
1.  $xml_parser = xml_parser_create();
2.  $rss_parser = new RSSParser();
3.  xml_set_object($xml_parser,&$rss_parser);
4.  xml_set_element_handler($xml_parser, "startElement",
"endElement");
5.  xml_set_character_data_handler($xml_parser,
"characterData");
6.  $fp = fopen("http://www.sitepoint.com/rss.php","r")
          or die("Error reading RSS data.");
7.  while ($data = fread($fp, 4096)) //read in 4K chunks
8.          xml_parse($xml_parser, $data, feof($fp))
            or die(sprintf("XML error: %s at line %d",

      xml_error_string(xml_get_error_code($xml_parser)),
                xml_get_current_line_number($xml_parser)));
9.  fclose($fp);
10. xml_parser_free($xml_parser);
```

How simple is that? Line 1 and 2 create instances of the PHP event based XML Parser and our class respectively. Line 3 tells the PHP parser that our object is going to be handling the events for it. Line 4 and 5 tell the parser which functions in our object are going to be handling the events fired by entering a tag, seeing the text within a tag and leaving a tag. Line 6 open our RSS feed, and line 7 starts reading it in, at most 4K at a time. Now line 8 (the while loop) is the interesting one. This loop is what reads through the portion of the document that has been read and fires the events as tags are encountered. The final two lines are simply closing the file and freeing the memory. Like Java, PHP will free this memory on its own, but why wait?

SAX/Event Based Model - Benefits

The benefits of the event driven parsing method include:

- Easy readability of RSS feeds (or any other XML documents that you wish to see in a particular format)
- Selective parsing of the XML document
- Extremely light on memory usage, especially compared to the DOM model
- Easy as 1-2-3: Create the parser, set the handlers, and pass the feed

SAX/Event Based Model - Drawbacks

Drawbacks include:

- None of the advanced functionality of the DOM based parser such as non-sequential access

- Very complex searches or formatting can be difficult to implement (although once implemented are encapsulated within the class)
- In an example like this there is no way a DTD can be used for validation

References:

NuSOAP Implementation - http://dietrich.ganx4.com/nusoap/index.php

Web services – http://webservices.xml.com

DOM Model – http://www.zend.com/php5/articles/php5-xmlphp.php

DOM Model - http://www.w3.org/DOM/

SAX/Event Based XML Parsing - http://www.sitepoint.com/article/php-xml-parsing-rss-1-0/2

SAX/Event Based XML Parsing - http://www.php.net