PHP and XML

Brian J. Stafford, Mark McIntyre and Fraser Gallop

What is PHP?

PHP is a server-side tool for creating dynamic web pages. PHP pages consist of both HTML and program logic.

One of the advantages of PHP is that it will run with almost any web server. It installs as an executable, so the only configuration needed is to setup a filter on the web server telling it to process requests for PHP pages through the executable.

When a client browser makes a request to the web server for a PHP document, the server first needs to recognize it as PHP. The PHP engine is used to process the document, and will output a page consisting of just HTML. The web server will then deliver this page to the client. The flow of logic here is very similar to that of a CGI script.

History

The first version of PHP, called *Personal Home Page/Forms Interpreter* was written by Rasmus Lerdorf in 1995. He originally built it to help him track hits to his on-line resume. It was written as a series of Perl scripts. After releasing it to the public domain, many people started using it, and soon requests were coming in for more features to be added. A larger C implementation was soon written to add functionality to PHP.

In 1997, Andi Gutmans and Zeev Suraski worked with Rasmus to make a complete rewrite of PHP/FI 2.0. It was renamed to *PHP: Hypertext Preprocessor* (a recursive acronym). This rewrite became the basis for the PHP that exists today. Probably the most important feature that was added to PHP at this point was the ability for others to write extension modules that could be used with PHP. This allowed other web developers to make their own enhancements and additions to the language. Along with extension modules, improvements were made to the language syntax.

Version 4 of PHP was released in May 2000 with a new engine, support for new web servers, and support for tracking user sessions.

Using PHP

Hello World

A PHP document will contain a mixture of code and HTML. The HTML part of the document is written as plain HTML. Code is contained within sets of <? and ?>. The most basic of examples to demonstrate this follows:

```
<html>
  <head>
    <title>PHP Test</title>
  </head>
  <body>
    <?php echo “Hello World<p>”; ?>
  </body>
</html>
```

PHP Syntax

Lines of PHP code will always end with a semicolon (;). Like Perl, variables are marked with a $. Much of the language was based on C and Perl, so for the most part, the syntax copies those languages.

The following is an example that will display a message showing which browser the user is running:

```
<?php
  if(strstr($HTTP_USER_AGENT,"MSIE")) {
?>
      <center><b>You are using Internet Explorer</b></center>
<?
  } else {
?>
      <center><b>You are not using Internet Explorer</b></center>
<?
  }
?>
```

Other language features that mimic Perl are the user of a hash type and a foreach loop. Regular expressions are supported using *ereg* and *pereg_match*. PHP follows the C++ convention for comments, allowing both single and multi-line.

Variables in PHP are weakly (dynamically) typed. It is possible to change the type of variable using `settype(variable, new_type)`. There is no need to speficy the size of arrays as they will grow dynamically to fit the data being stored in them.

Forms

One of the more powerful features in PHP is the way in which it handles forms. Variables are created for input values automatically in target forms. This means that, unlike other languages like Perl, JSP, and ASP, the web designer does not have to do anything to request the data from those values, they can simply reference them just like any other variable.

The following is a HTML page containing a form that will be submitted to PHP:

```
<form action="action.php" method="post">
  Your name: <input type="text" name="name">
  You age: <input type="text" name="age">
  <input type="submit">
</form>
```

Once the form is submitted, the values of name and age can be accessed in the PHP:

```
Hi <?php echo $name; ?>.
You are <?php echo $age; ?> years old.
```

PHP and XML

There is an extension to PHP, called expat, which allows a PHP application to parse XML documents. However, this extension does not provide the ability to validate XML documents with respect to DTDs.

The parsing model is similar to that of SAX; it's an event-based model in which you register handlers for various events that can occur during the parse operation. These events include readings start and end tags, character data, and encountering errors.

In order to keep track of where you are in the XML document tree, you typically need to use a stack. This allows you to record which elements have been seen, which provides the necessary context to make appropriate decisions while the document is being parsed.

A parser is created with the following call:

```
int xml_parser_create ( [string encoding])
```

'encoding' refers to the type of character encoding the parser should use, and can be one of the following:

- ISO-8859-1 (default)
- US-ASCII
- UTF-8

When finished with the parser, the PHP code must release it, as follows:

```
string xml_parser_free ( int parser)
```

The main handlers handle events associated with start and end tags. They are set as follows:

```
int xml_set_element_handler ( int parser, string startElementHandler, string
endElementHandler)
```

startElementHandler and endElementHandler are the names of the handlers to be used for handling start tags and end tags, respectively.
A valid handler for start tags must accept three parameters: the parser, the name of the tag, and a hash containing the attributes of the tag.
A valid handler for end tags must accept two parameters: the parser, and the name of the tag.

In order to handle character data, the PHP application will need to set a handler for that purpose:

```
int xml_set_character_data_handler ( int parser, string handler)
```

This function takes as parameters the parser as well as the name of the function that will handle character data. That function must also accept the parser as a parameter, as well as a string containing the character data.

There are other handlers for such things as external entities, processing instructions, and namespaces. A PHP application can also set a default handler to handle any events that would otherwise go unhandled:

```
int xml_set_default_handler ( int parser, string handler)
```

A valid default handler must take as parameters the parser and a string containing the data associated with the event.

To parse the document, the following function is called:

```
int xml_parse ( int parser, string data [, int isFinal])
```

This function takes as parameters the parser, the data to parse (read from the XML file), and an optional flag to indicate whether the data contains the remainder of the file. The XML file is parsed in pieces (blocks of data), and this flag indicates when the last call is being made to xml_parse. It is during these repeated calls to xml_parse that events are fired and handled by the registered event handlers.

To demonstrate the XML parsing ability of this PHP extension, we re-implemented Assignment #5 using PHP and XML (and exchanging spy products for plants). This was easily accomplished without using a stack; a more complicated example would typically require one. The code listing for this example appears in Appendix A.

PHP and WDDX

PHP has a powerful set of functions to serialize and de-serialize WDDX documents. WDDX was developed by Allaire (later acquired by Macromedia) and is similar to XML-RPC, except it is designed for the transfer of structures without a specific procedure call. This makes it extremely simple to use, while being equally as flexible.

```
<wddxPacket version=”1.0”>
      <header comment=”PHP”/>
      <data>
            <struct>
                  <var name=”pi”>
                        <number>3.1415926</number>
                  </var>
                  <var name=”cities”>
                        <array length=”3”>
                              <string>Austin</string>
                              <string>Novato</string>
                              <string>Seattle</string>
                        </array>
                  </var>
            </struct>
      </data>
</wddxPacket>
```

The following is an example of a simple WDDX document:

Rather than manually parsing and assembling a structure out of such a document, PHP can access the following functions to automatically take these documents and return structures containing the appropriate data.

- wddx_serialize_value – Takes a single variable and serializes it to a WDDX packet.
- wddx_serialize_vars – Takes a series of variables and serializes them to a WDDX packet.
- wddx_packet_start - Initializes a new WDDX packet with no current values.
- wddx_packet_end - Ends (terminates) a WDDX packet after the values have been added.
- wddx_add_vars - Adds a variable to the specified WDDX packet.
- wddx_deserialize – De-serializes a WDDX packet into a PHP structure.

An example of de-serializing an external packet is provided below. It requests a WDDX document from the site http://www.listology.com/. This site provides thousands of lists of data for viewing online, or for export for use by other applications. By requesting one of these documents as a WDDX packet, PHP can instantly make use of these lists without the need to explicitly parse the WDDX document itself.

```
// Requests a WDDX document from listology.com and outputs
// the list data as HTML formatted text.

// create the file handle
$fhandle =
fopen("http://www.listology.com/syndicate_content.cfm?content_id=".
       $id . "&mode=wddx","r");

// if the handle was successfully created…
if ($fhandle) {
      // read the document into a buffer
      $page = fread($fhandle,10000);

      // close the file handle
      fclose($fhandle);

      // deserialize the WDDX document and store
      // the structure in a variable called 'value'
      $value = wddx_deserialize($page);

      // Print the title and comments
      print "<h1>" . $value["TITLE"] . "</h1>";
      print "<p>" . $value["COMMENTS"] . "</p>";

      // store the list data in an array called 'data'
      $data = $value["DATA"];

      // print out the list data as an ordered list
      print "<ol>";
      foreach ( $data as $val ) {
            print "<li>" . $val . "</li>";
      }
      print "</ol>";
}
```

By directly supporting WDDX, PHP enables developers to take advantage of WDDX technology without knowing the WDDX syntax itself. WDDX is an open standard, so PHP will be able to communicate with many other web applications. Supporting products already include Flash and Director, and a third party JavaScript WDDX de-serializer is available.

Appendix A

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
      <head>
            <style type="text/css">
            body {
                  color: white;
            }
      </style>
            <title>Spy Store</title>
      </head>
      <body bgcolor="black">

<?php
$file = "catalog.xml" ;
$curElem = array() ;

function startElement($parser, $name, $attrs)
{
      $curElem[] = $name ;

    if ($name == "item")
    {
            ?><tr><td><table border="0" ID="Table1"><?
    }
    elseif ($name == "name")
    {
            ?><tr><h2><?
    }
    elseif ($name == "image")
    {
            ?><tr><td><img src="<?
    }
    elseif ($name == "description")
    {
            ?><td><?
    }
    elseif ($name == "price")
    {
            ?><h3>Price: <?
    }
    elseif ($name == "model_number")
    {
            ?><h3>Model Number: <?
    }
}

function endElement($parser, $name)
{
      if ($name == "item") {
            ?></table></td></tr><?
      }
      elseif ($name == "name") {
            ?></h2></tr><?
      }
    elseif ($name == "image") {
            ?>"/></td><?
    }
    elseif ($name == "description") {
            ?><p><?
    }
    elseif ($name == "price") {
            ?></h3><?
    }
    elseif ($name == "model_number") {
            ?></h3></td></tr><?
    }
}

function cdataHandler($parser, $data) {
      print $data ;
}

$xml_parser = xml_parser_create() ;
xml_parser_set_option($xml_parser, XML_OPTION_CASE_FOLDING, FALSE) ;
xml_set_element_handler($xml_parser, "startElement", "endElement") ;
xml_set_character_data_handler($xml_parser, "cdataHandler") ;

?>
            <table border="0">
                  <tr>
                        <td><a href="http://www.ibuyspy.com"><img
src="http://www.ibuyspystore.com/images/logo.gif" border="0"/></a></td>
                        <td><h1>Spy Store</h1></td>
                  </tr>
            </table>
            <h3>Welcome to the Spy Store! We have an excellent selection of
items to help you
                  with your espionage!</h3>
            <table border="0">
<?

if (!($fp = fopen($file, "r")))
{
    die("could not open XML input");
}

while ($data = fread($fp, 4096))
{
    if (!xml_parse($xml_parser, $data, feof($fp)))
    {
        die(sprintf("XML error: %s at line %d",
                    xml_error_string(xml_get_error_code($xml_parser)),
                    xml_get_current_line_number($xml_parser))) ;
    }
}

xml_parser_free($xml_parser) ;
?>
            </table>
      </body>
</html>
```