PHP 101 (Part 1) – Secret Agent Man

By Vikram Vaswani and Harish Kamath

Table of Contents

The PHP Story

Ever since Web designers found out about the tag, the Internet has seen an explosion in the number of Web sites that depend heavily on user response and interactivity. For a long time, the primary language used to develop such Web sites was Perl. But ask any novice programmer, and he'll tell you that learning Perl isn't exactly a bed of roses...

As a result, there has been a proliferation of alternative server−side scripting languages, which perform many of the tasks previously handled by Perl, but have a shorter learning curve. The most well−known of these are ASP and PHP; while the former works primarily on the Windows platform in combination with a clutch of proprietary products, the latter has the unique distinction of being an open−source server−side scripting language that's both fun and easy to learn. Today, it is estimated that more than 1,000,000 Web sites use PHP as a server side scripting language.

PHP was first developed by Rasmus Lerdorf as a means of monitoring page views for his online resumé, and slowly started making a mark when PHP/FI was released in mid−1995. This version of PHP had support for some basic Web functions − the ability to handle form data, support for the mSQL database, and more.

As PHP's popularity grew, the development of the language shifted from Rasmus to a team of dedicated programmers who took upon themselves the onus of rewriting the PHP parser from scratch. The result of the efforts was PHP 3.0, which included support for a wider range of databases, including MySQL and Oracle. And PHP 4.0, which was released a few weeks ago, uses the powerful new Zend scripting engine to deliver better performance, supports Web servers other than Apache, and comes with in−built support for session management.

Our goal in this series of articles is very simple − we'll be teaching you the basics of using PHP to power your Web site, and related Web development efforts. The only assumptions we're going to make throughout this series are that you know the basics of HTML, are using a properly configured Web server running PHP4, and have a sense of humour.

If you're running Apache, detailed instructions on configuring it to work with PHP are available at http://www.devshed.com/Server_Side/PHP/SoothinglySeamless/ . Alternatively, download the latest distribution of PHP4 from the official PHP Web site at http://www.php.net and take a look at the installation instructions.

Bond...James Bond

Once you've configured your Web server to parse PHP pages, it's time to test it and see if everything's working as advertised. The simplest way to do this is to pop open your favourite text editor and create a file containing these lines of code:

```
<?php phpinfo(); ?>
```

Save the file with the extension .php − for example, "test.php"

Now, start up your Web browser and point it to the file you just saved − for example, http://localhost/test.php − and you'll see a page filled with what at first glance appears to be gibberish, but on closer inspection will reveal itself to be a list of internal PHP variables. The values of most of these variables can be modified by altering the "php.ini" file that ships with every distribution of PHP. For beginners, the default values are more than sufficient to work with.

There's one essential concept that you need to get your mind around before we proceed further. Unlike CGI scripts, which require you to write code to output HTML, PHP lets you create embed PHP code in regular HTML pages, and execute the embedded PHP code when the page is requested.

These embedded PHP commands are enclosed within special start and end tags − here's what they look like:

```
<?php ... PHP code ... ?> or the shorter version <? ... PHP
code ... ?>
```

Here's a simple example which demonstrates how PHP and HTML can be combined:

```
<html> <head> <title>Bonding With PHP</title> </head> <body>
So who do you think you are, anyhow? <br> <?php // this is all
PHP code echo "<b>The name's Bond...James Bond!</b>"; ?>
</body> </html>
```

And if you browse to this page through your browser and take a look at the HTML source, this is what you'll see:

```
<html> <head> <title>Bonding With PHP</title> </head> <body>
So who do you think you are, anyhow? <br> <b>The name's
Bond...James Bond!</b> </body> </html>
```

Every PHP statement ends in a semi−colon − this convention is identical to that used in Perl, and omitting the semi−colon is one of the most common mistakes newbies make. It's also possible to add comments to your PHP code, as we've done in the example above. PHP supports both single−line and multi−line comment blocks − take a look:

```
<?php // this is a single−line comment /* and this is a
multi−line comment */ ?>
```

A Case Of Identity

Variables are the bread and butter of every programming language...and PHP has them too. A variable can be thought of as a programming construct used to store both numeric and non−numeric data; this data can then be used in different places in your PHP scripts.

PHP supports a number of different variable types: integers, floating point numbers, strings and arrays. In many languages, it's essential to specify the variable type before using it; for example, a variable may need to be specified as type "integer" or type "array". Give PHP credit for a little intelligence, though − the language can automagically determine variable type by the context in which it is being used.

Every variable has a name − in PHP, a variable name is preceded by a dollar [$] sign and must begin with a letter, optionally followed by more letters and numbers. For example,

$popeye $one $INCOME are all valid PHP variables.

Note that variable names in PHP are case sensitive − so

$me

is different from

$Me

or

$ME

Here's a simple example which demonstrates PHP's variables:

```
<html> <head> <title>Bonding With PHP</title> </head> <body>
So who do you think you are, anyhow? <br> <? // set up some
variables $fname = "James"; $lname = "Bond"; ?> <b><? echo
"The name's $lname...$fname $lname!"; ?></b> </body> </html>
```

In this case, the variables $fname and $lname are first defined with string values, and then substituted in the echo() function call. Just as an aside...the echo() function is another important PHP function, and one that you'll be using a great deal over the next few lessons. It is commonly used to display output.

Synonymous to echo() is print(), which does exactly the same thing − take a look at the example below, which demonstrates how to use it.

```
<html> <head> <title>Bonding With PHP</title> </head> <body>
So who do you think you are, anyhow? <br> <? // set up some
variables $fname = "James"; $lname = "Bond"; ?> <?
```

```
print("<b>The name's $lname...$fname $lname! </b>"); ?>
</body> </html>
```

Note how we've included the HTML <b> tag within the string to be displayed in this example...you can do this too. Really.

The Toy Shop

Once you've got the basics of variables down, it's time to start doing something with them. We'll start with PHP's most useful mathematical operators − here's an example which demonstrates them:

```
<?php // set up some variables // the toys $item1 = "X−ray
specs"; $item2 = "Watch with built−in poison gas canister";
$item3 = "Exploding chewing gum"; // the price $item1_cost =
100; $item2_cost = 250; $item3_cost = 32; // the amount
$item1_qty = 1; $item2_qty = 2; $item3_qty = 15; // calculate
cost for each item $item1_total = $item1_cost * $item1_qty;
$item2_total = $item2_cost * $item2_qty; $item3_total =
$item3_cost * $item3_qty; // calculate grand total
$grand_total = $item1_total + $item2_total + $item3_total; //
special secret agent discount − 10% $discount = 10; // which
reduces total bill amount $amount = ($grand_total * 10)/100;
// the bottom line $net_total = $grand_total − $amount; ?>
<html> <head> <title>Boys And Their Toys</title> <style
type="text/css"> td {font−family:Verdana;} </style> </head>
<body> <center> <table border="3" cellspacing="0"
cellpadding="4"> <tr> <td colspan="4" align="center"
valign="middle"> <b>The Secret Agent Store</b> </td> </tr>
<tr> <td> <b>Description</b> </td> <td> <b>Unit Cost<b> </td>
<td> <b>Quantity</b> </td> <td> <b>Total</b> </td> </tr> <?php
// Start echo the values echo "<tr> <td> $item1 </td> <td
align=right> $$item1_cost </td> <td align=right> $item1_qty
</td> <td align=right> $$item1_total </td> </tr>"; echo "<tr>
<td> $item2 </td> <td align=right> $$item2_cost </td> <td
align=right> $item2_qty </td> <td align=right> $$item2_total
</td> </tr>"; echo "<tr> <td> $item3 </td> <td align=right>
$$item3_cost </td> <td align=right> $item3_qty </td> <td
align=right> $$item3_total </td> </tr>"; echo "<tr> <td
colspan=3 align=right> <b>Gross</b> </td> <td align=right>
<b>$$grand_total</b> </td> </tr>"; echo "<tr> <td colspan=3
align=right> <b>Less discount [$discount%]</b> </td> <td
align=right> <b>$$amount</b> </td> </tr>"; echo "<tr> <td
colspan=3 align=right> <b>Net</b> </td> <td align=right>
<b>$$net_total</b> </td> </tr>"; ?> </table> <font size=−2
color=silver face=Verdana>Thank you for shopping at the Secret
Agent Store!</font> </center> </body> </html>
```

Looks complex? Don't be afraid − it's actually pretty simple. The meat of the script is at the top, where we've set up variables for the various items, the unit cost and the quantity. Next, we've performed a bunch of calculations using PHP's various mathematical operators, and stored the results of those calculations in different variables. The rest of the script is related to the layout and alignment of the various items on the bill, with PHP variables embedded within the HTML code.

Obviously, just as you can add and subtract numbers, PHP also allows you to concatenate strings with the string concatenation operator, represented by a period[.] The next example will make this clear:

```
<?php // set up some variables $a = "Where"; $b = "Are"; $c =
"You"; // first alternative $gamma = $a . " " . $b . " " . $c;
// second alternative $delta = $a . " " . $c . " " . $b; ?>
<html> <head> <title>All Tied Up</title> </head> <body> The
first string is <? echo $gamma; ?> <br> The second string is
<? echo $delta; ?> </body> </html>
```

And here's what you'll see:

```
The first string is Where Are You The second string is Where
You Are
```

Weapons To Die For

If you've used C before, you're probably already familiar with the "include" directive that appears near the beginning of every C program. PHP supports two functions which have a similar job to do − the include() function and the require() function. Take a look at a simple example:

```
<html> <head> <title>Weapons Worth Dying For</title> <style>
h1,h3,li { font-family:Verdana; } </style> </head> <?php //
this time, Bond's going to need the cigarette-lighter gun...
require("./gun.php4"); // the new car-copter...
include("./car.php4"); // and of course, the gold watch with
the hidden GPS locator require("./watch.php4"); ?> <body>
<h3>So, James, here's your check list for the mission.</h3>
<ol type="a"> <li>The gun: <?php echo "$gun"; ?> <li>The car:
<?php echo "$car"; ?> <li>The watch: <?php echo "$watch"; ?>
</ol> <br> <h3>Oh yes...and remember never to let them see you
cry. Good luck, 007!</h3> </body> </html>
```

Now, if you try to access this page as it, you'll get a bunch of error messages warning you about missing files. So you need to create the files "gun.php4", "car.php4" and "watch.php4":

[gun.php4]

```
<?php $gun = "AK-47"; ?>
```

[car.php4]

```
<?php $car = "BMW G8"; ?>
```

[watch.php4]

```
<?php $watch = "Rolex SAW-007"; ?>
```

And this time, when you access the primary page, PHP should automatically include the specified files, read the variables $gun, $watch and $car from them, and display them on the page.

A quick note on the difference between the include() and require() functions − the require() function is always replaced by the contents of the file it points to, and therefore cannot be used in a conditional statement ["if this is true, require that file"] since the file will be read in regardless. However, the include() function allows you to optionally include or exclude files on the basis of a conditional test. Also, a require()d file cannot return values to the main PHP script, while an included file can.

An important point to be noted is that when a file is require()d or include()d, the PHP parser leaves "PHP mode" and goes back to regular "HTML mode". Therefore, all PHP code within the included external files needs to be enclosed within regular PHP <?...?> tags.

A very useful and practical application of the include() function is to use it to include a standard footer or copyright notice across all the pages of your Web site, like this:

```
<html> <head> <title></title> </head> <body> ...your HTML
page... <br> <? include("footer.html"); ?> </body> </html>
```

where "footer.html" contains

This material copyright Melonfire, 2000. All rights reserved.

Now, this footer will appear on each and every page that contains the include() statement above − and, if you need to change the message, you only need to edit the single file named "footer.html"!

And that's about it for this week. We've shown you the basic building blocks of PHP − its variables and operators − and next time, we'll be using those fundamental concepts to demonstrate PHP's powerful form processing capabilities. Don't miss it!