Getting Started

Introduction

What is PHP?

PHP (recursive acronym for "PHP: Hypertext Preprocessor") is a widely-used open source general-purpose scripting language that is especially suited for web development and can be embedded into HTML.

Nice, but what does that mean? An example:

Example #1 - An introductory example

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">
<html>
   <head>
       <title>Example</title>
   </head>
   <body>

       <?php
           echo "Hi, I'm a PHP script!";
       ?>

   </body>
</html>
```

Instead of lots of commands to output HTML (as seen in C or Perl), PHP pages contain HTML with embedded code that does "something" (in this case, output "Hi, I'm a PHP script!"). The PHP code is enclosed in special start and end processing instructions `<?php` and `?>` that allow you to jump into and out of "PHP mode."

What distinguishes PHP from something like client-side JavaScript is that the code is executed on the server, generating HTML which is then sent to the client. The client would receive the results of running that script, but would not know what the underlying code was. You can even configure your web server to process all your HTML files with PHP, and then there's really no way that users can tell what you have up your sleeve.

The best things in using PHP are that it is extremely simple for a newcomer, but offers many advanced features for a professional programmer. Don't be afraid reading the long list of PHP's features. You can jump in, in a short time, and start writing simple scripts in a few hours.

Although PHP's development is focused on server-side scripting, you can do much more with it. Read on, and see more in the What can PHP do? section, or go right to the introductory tutorial if you are only interested in web programming.

What can PHP do?

Anything. PHP is mainly focused on server-side scripting, so you can do anything any other CGI program can do, such as collect form data, generate dynamic page content, or send and receive cookies. But PHP can do much more.

There are three main areas where PHP scripts are used.

- Server-side scripting. This is the most traditional and main target field for PHP. You need three things to make this work. The PHP parser (CGI or server module), a web server and a web browser. You need to run the web server, with a connected PHP installation. You can access the PHP program output with a web browser, viewing the PHP page through the server. All these can run on your home machine if you are just experimenting with PHP programming. See the installation instructions section for more information.
- Command line scripting. You can make a PHP script to run it without any server or browser. You only need the PHP parser to use it this way. This type of usage is ideal for scripts regularly executed using cron (on *nix or Linux) or Task Scheduler (on Windows). These scripts can also be used for simple text processing tasks. See the section about Command line usage of PHP for more information.
- Writing desktop applications. PHP is probably not the very best language to create a desktop application with a graphical user interface, but if you know PHP very well, and would like to use some advanced PHP features in your client-side applications you can also use PHP-GTK to write such programs. You also have the ability to write cross-platform applications this way. PHP-GTK is an extension to PHP, not available in the main distribution. If you are interested in PHP-GTK, visit » its own website.

PHP can be used on all major operating systems, including Linux, many Unix variants (including HP-UX, Solaris and OpenBSD), Microsoft Windows, Mac OS X, RISC OS, and probably others. PHP has also support for most of the web servers today. This includes Apache, Microsoft Internet Information Server, Personal Web Server, Netscape and iPlanet servers, Oreilly Website Pro server, Caudium, Xitami, OmniHTTPd, and many others. For the majority of the servers PHP has a module, for the others supporting the CGI standard, PHP can work as a CGI processor.

So with PHP, you have the freedom of choosing an operating system and a web server. Furthermore, you also have the choice of using procedural programming or object oriented programming, or a mixture of them. Although not every standard OOP feature is implemented in PHP 4, many code libraries and large applications (including the PEAR library) are written only using OOP code. PHP 5 fixes the OOP related weaknesses of PHP 4, and introduces a complete object model.

With PHP you are not limited to output HTML. PHP's abilities includes outputting images, PDF files and even Flash movies (using libswf and Ming) generated on the fly. You can also output easily any text, such as XHTML and any other XML file. PHP can autogenerate these files, and save them in the file system, instead of printing it out, forming a server-side cache for your dynamic content.

One of the strongest and most significant features in PHP is its support for a wide range of databases. Writing a database-enabled web page is incredibly simple. The following databases are currently supported:

- Adabas D
- dBase
- Empress
- FilePro (read-only)
- Hyperwave
- IBM DB2
- Informix
- Ingres
- InterBase
- FrontBase
- mSQL
- Direct MS-SQL
- MySQL
- ODBC
- Oracle (OCI7 and OCI8)
- Ovrimos
- PostgreSQL
- SQLite
- Solid
- Sybase
- Velocis
- Unix dbm

We also have a database abstraction extension (named PDO) allowing you to transparently use any database supported by that extension. Additionally PHP supports ODBC, the Open Database Connection standard, so you can connect to any other database supporting this world standard.

PHP also has support for talking to other services using protocols such as LDAP, IMAP, SNMP, NNTP, POP3, HTTP, COM (on Windows) and countless others. You can also open raw network sockets and interact using any other protocol. PHP has support for the WDDX complex data exchange between virtually all Web programming languages. Talking about interconnection, PHP has support for instantiation of Java objects and using them transparently as PHP objects. You can also use our CORBA extension to access remote objects.

PHP has extremely useful text processing features, from the POSIX Extended or Perl regular expressions to parsing XML documents. For parsing and accessing XML documents, PHP 4 supports the SAX and DOM standards, and you can also use the XSLT extension to transform XML documents. PHP 5 standardizes all the XML extensions on the solid base of libxml2 and extends the feature set adding SimpleXML and XMLReader support.

At last but not least, we have many other interesting extensions, the mnoGoSearch search engine functions, the IRC Gateway functions, many compression utilities (gzip, bz2, zip), calendar conversion, translation...

As you can see this page is not enough to list all the features and benefits PHP can offer. Read on in the sections about installing PHP, and see the function reference part for explanation of the extensions mentioned here.

A simple tutorial

Here we would like to show the very basics of PHP in a short, simple tutorial. This text only deals with dynamic web page creation with PHP, though PHP is not only capable of creating web pages. See the section titled What can PHP do for more information.

PHP-enabled web pages are treated just like regular HTML pages and you can create and edit them the same way you normally create regular HTML pages.

What do I need?

In this tutorial we assume that your server has activated support for PHP and that all files ending in *.php* are handled by PHP. On most servers, this is the default extension for PHP files, but ask your server administrator to be sure. If your server supports PHP, then you do not need to do anything. Just create your *.php* files, put them in your web directory and the server will automatically parse them for you. There is no need to compile anything nor do you need to install any extra tools. Think of these PHP-enabled files as simple HTML files with a whole new family of magical tags that let you do all sorts of things. Most web hosts offer PHP support, but if your host does not, consider reading the » PHP Links section for resources on finding PHP enabled web hosts.

Let us say you want to save precious bandwidth and develop locally. In this case, you will want to install a web server, such as » Apache, and of course » PHP. You will most likely want to install a database as well, such as » MySQL.

You can either install these individually or choose a simpler way. Our manual has installation instructions for PHP (assuming you already have some web server set up). In case you have problems with installing PHP yourself, we would suggest you ask your questions on our » installation mailing list. If you choose to go on the simpler route, then » locate a pre-configured package for your operating system, which automatically installs all of these with just a few mouse clicks. It is easy to setup a web server with PHP support on any operating system, including MacOSX, Linux and Windows. On Linux, you may find » rpmfind and » PBone helpful for locating RPMs. You may also want to visit » apt-get to find packages for Debian.

Your first PHP-enabled page

Create a file named *hello.php* and put it in your web server's root directory (*DOCUMENT_ROOT*) with the following content:

Example #2 - Our first PHP script: *hello.php*

```
<html>
<head>
```

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

Use your browser to access the file with your web server's URL, ending with the "/hello.php" file reference. When developing locally this URL will be something like *http://localhost/hello.php* or *http://127.0.0.1/hello.php* but this depends on the web server's configuration. If everything is configured correctly, this file will be parsed by PHP and the following output will be sent to your browser:

```
<html>
<head>
 <title>PHP Test</title>
</head>
<body>
<p>Hello World</p>
</body>
</html>
```

This program is extremely simple and you really did not need to use PHP to create a page like this. All it does is display: *Hello World* using the PHP echo() statement. Note that the file *does not need to be executable* or special in any way. The server finds out that this file needs to be interpreted by PHP because you used the ".php" extension, which the server is configured to pass on to PHP. Think of this as a normal HTML file which happens to have a set of special tags available to you that do a lot of interesting things.

If you tried this example and it did not output anything, it prompted for download, or you see the whole file as text, chances are that the server you are on does not have PHP enabled, or is not configured properly. Ask your administrator to enable it for you using the Installation chapter of the manual. If you are developing locally, also read the installation chapter to make sure everything is configured properly. Make sure that you access the file via http with the server providing you the output. If you just call up the file from your file system, then it will not be parsed by PHP. If the problems persist anyway, do not hesitate to use one of the many » PHP support options.

The point of the example is to show the special PHP tag format. In this example we used *<?php* to indicate the start of a PHP tag. Then we put the PHP statement and left PHP mode by adding the closing tag, *?>*. You may jump in and out of PHP mode in an HTML file like this anywhere you want. For more details, read the manual section on the basic PHP syntax.

Note

A Note on Line Feeds

Line feeds have little meaning in HTML, however it is still a good idea to make your HTML look nice and clean by putting line feeds in. A linefeed that follows immediately after a closing *?>* will be removed by PHP. This can be extremely useful when you are

putting in many blocks of PHP or include files containing PHP that aren't supposed to output anything. At the same time it can be a bit confusing. You can put a space after the closing ?> to force a space and a line feed to be output, or you can put an explicit line feed in the last echo/print from within your PHP block.

Note

A Note on Text Editors

There are many text editors and Integrated Development Environments (IDEs) that you can use to create, edit and manage PHP files. A partial list of these tools is maintained at » PHP Editors List. If you wish to recommend an editor, please visit the above page and ask the page maintainer to add the editor to the list. Having an editor with syntax highlighting can be helpful.

Note

A Note on Word Processors

Word processors such as StarOffice Writer, Microsoft Word and Abiword are not optimal for editing PHP files. If you wish to use one for this test script, you must ensure that you save the file as *plain text* or PHP will not be able to read and execute the script.

Note

A Note on Windows Notepad

If you are writing your PHP scripts using Windows Notepad, you will need to ensure that your files are saved with the .php extension. (Notepad adds a .txt extension to files automatically unless you take one of the following steps to prevent it.) When you save the file and are prompted to provide a name for the file, place the filename in quotes (i.e. " *hello.php* "). Alternatively, you can click on the 'Text Documents' drop-down menu in the 'Save' dialog box and change the setting to "All Files". You can then enter your filename without quotes.

Now that you have successfully created a working PHP script, it is time to create the most famous PHP script! Make a call to the phpinfo() function and you will see a lot of useful information about your system and setup such as available predefined variables, loaded PHP modules, and configuration settings. Take some time and review this important information.

Example #3 - Get system information from PHP

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

Something Useful

Let us do something more useful now. We are going to check what sort of browser the visitor is using. For that, we check the user agent string the browser sends as part of the HTTP request. This information is stored in a variable. Variables always start with a dollar-sign in PHP. The variable we are interested in right now is *$_SERVER['HTTP_USER_AGENT']*.

Note

$_SERVER is a special reserved PHP variable that contains all web server information. It is known as a superglobal. See the related manual page on superglobals for more information. These special variables were introduced in PHP » 4.1.0. Before this time, we used the older *$HTTP_*_VARS* arrays instead, such as *$HTTP_SERVER_VARS*. Although deprecated, these older variables still exist. (See also the note on old code.)

To display this variable, you can simply do:

Example #4 - Printing a variable (Array element)

```
<?php
echo $_SERVER['HTTP_USER_AGENT'];
?>
```

A sample output of this script may be:

```
Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
```

There are many types of variables available in PHP. In the above example we printed an Array element. Arrays can be very useful.

$_SERVER is just one variable that PHP automatically makes available to you. A list can be seen in the Reserved Variables section of the manual or you can get a complete list of them by looking at the output of the phpinfo() function used in the example in the previous section.

You can put multiple PHP statements inside a PHP tag and create little blocks of code that do more than just a single echo. For example, if you want to check for Internet Explorer you can do this:

Example #5 - Example using control structures and functions

```
<?php
if (strpos($_SERVER['HTTP_USER_AGENT'], 'MSIE') !== FALSE) {
   echo 'You are using Internet Explorer.<br />';
}
?>
```

A sample output of this script may be:

```
You are using Internet Explorer.<br />
```

Here we introduce a couple of new concepts. We have an if statement. If you are familiar with the basic syntax used by the C language, this should look logical to you. Otherwise, you should probably pick up an introductory PHP book and read the first couple of chapters, or read the Language Reference part of the manual.

The second concept we introduced was the strpos() function call. strpos() is a function built into PHP which searches a string for another string. In this case we are looking for *'MSIE'* (so-called needle) inside *$_SERVER['HTTP_USER_AGENT']* (so-called haystack). If the needle is found inside the haystack, the function returns the position of the needle relative to the start of the haystack. Otherwise, it returns **FALSE**. If it does not return **FALSE**, the if expression evaluates to **TRUE** and the code within its {braces} is executed. Otherwise, the code is not run. Feel free to create similar examples, with if, else, and other functions such as strtoupper() and strlen(). Each related manual page contains examples too. If you are unsure how to use functions, you will want to read both the manual page on how to read a function definition and the section about PHP functions.

We can take this a step further and show how you can jump in and out of PHP mode even in the middle of a PHP block:

Example #6 - Mixing both HTML and PHP modes

```
<?php
if (strpos($_SERVER['HTTP_USER_AGENT'], 'MSIE') !== FALSE) {
?>
<h3>strpos() must have returned non-false</h3>
<p>You are using Internet Explorer</p>
<?php
} else {
?>
<h3>strpos() must have returned false</h3>
<p>You are not using Internet Explorer</p>
<?php
}
```

```
?>
```

A sample output of this script may be:

```
<h3>strpos() must have returned non-false</h3>
<p>You are using Internet Explorer</p>
```

Instead of using a PHP echo statement to output something, we jumped out of PHP mode and just sent straight HTML. The important and powerful point to note here is that the logical flow of the script remains intact. Only one of the HTML blocks will end up getting sent to the viewer depending on the result of strpos(). In other words, it depends on whether the string *MSIE* was found or not.

Dealing with Forms

One of the most powerful features of PHP is the way it handles HTML forms. The basic concept that is important to understand is that any form element will automatically be available to your PHP scripts. Please read the manual section on Variables from external sources for more information and examples on using forms with PHP. Here is an example HTML form:

Example #7 - A simple HTML form

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

There is nothing special about this form. It is a straight HTML form with no special tags of any kind. When the user fills in this form and hits the submit button, the *action.php* page is called. In this file you would write something like this:

Example #8 - Printing data from our form

```
Hi <?php echo htmlspecialchars($_POST['name']); ?>.
You are <?php echo (int)$_POST['age']; ?> years old.
```

A sample output of this script may be:

```
Hi Joe. You are 22 years old.
```

Apart from the htmlspecialchars() and *(int)* parts, it should be obvious what this does. htmlspecialchars() makes sure any characters that are special in html are properly encoded so people can't inject HTML tags or Javascript into your page. For the age field, since we know it is a number, we can just convert it to an integer which will automatically get rid of any stray characters. You can also have PHP do this for you automatically by using the filter extension. The *$_POST['name']* and *$_POST['age']* variables are automatically set for you by PHP. Earlier we used the *$_SERVER* superglobal; above we just introduced the $_POST superglobal which contains all POST data. Notice how the *method* of our form is POST. If we used the method *GET* then our form information would live in the $_GET superglobal instead. You may also use the $_REQUEST superglobal, if you do not care about the source of your request data. It contains the merged information of GET, POST and COOKIE data. Also see the import_request_variables() function.

You can also deal with XForms input in PHP, although you will find yourself comfortable with the well supported HTML forms for quite some time. While working with XForms is not for beginners, you might be interested in them. We also have a short introduction to handling data received from XForms in our features section.

Using old code with new versions of PHP

Now that PHP has grown to be a popular scripting language, there are a lot of public repositories and libraries containing code you can reuse. The PHP developers have largely tried to preserve backwards compatibility, so a script written for an older version will run (ideally) without changes in a newer version of PHP. In practice, some changes will usually be needed.

Two of the most important recent changes that affect old code are:

- The deprecation of the old *$HTTP_*_VARS* arrays (which need to be indicated as global when used inside a function or method). The following superglobal arrays were introduced in PHP » 4.1.0. They are: *$_GET*, *$_POST*, *$_COOKIE*, *$_SERVER*, *$_FILES*, *$_ENV*, *$_REQUEST*, and *$_SESSION*. The older *$HTTP_*_VARS* arrays, such as *$HTTP_POST_VARS*, also exist. As of PHP 5.0.0, the long PHP predefined variable arrays may be disabled with the register_long_arrays directive.
- External variables are no longer registered in the global scope by default. In other words, as of PHP » 4.2.0 the PHP directive register_globals is *off* by default in *php.ini*. The preferred method of accessing these values is via the superglobal arrays mentioned above. Older scripts, books, and tutorials may rely on this directive being on. If it were on, for example, one could use *$id* from the URL *http://www.example.com/foo.php?id=42*. Whether on or off, *$_GET['id']* is available.

For more details on these changes, see the section on predefined variables and links therein.

What's next?

With your new knowledge you should be able to understand most of the manual and also

the various example scripts available in the example archives. You can also find other examples on the php.net websites in the links section: » http://www.php.net/links.php.

To view various slide presentations that show more of what PHP can do, see the PHP Conference Material Site: » http://talks.php.net/