**Software Development – PHP Basics**

**Training Manual - Trainee Guide**

**Table of Contents**

[Introduction: 5](#_Toc300230598)

[What do I need? 7](#_Toc300230599)

[Comments in PHP 11](#_Toc300230600)

[Variables in PHP 12](#_Toc300230601)

[PHP String Functions 13](#_Toc300230602)

[PHP Operators 15](#_Toc300230603)

[PHP Forms and User Input 17](#_Toc300230604)

[PHP Form Handling 17](#_Toc300230605)

[Form Validation 19](#_Toc300230606)

[PHP $\_GET Function 20](#_Toc300230607)

[PHP $\_POST Function 21](#_Toc300230608)

[Conditional Statements 22](#_Toc300230609)

[The if Statement 22](#_Toc300230610)

[The if...else Statement 22](#_Toc300230611)

[The if...elseif....else Statement 23](#_Toc300230612)

[PHP Switch Statement 24](#_Toc300230613)

[PHP Arrays 25](#_Toc300230614)

[What is an Array? 25](#_Toc300230615)

[Numeric Arrays 25](#_Toc300230616)

[Associative Arrays 26](#_Toc300230617)

[Multidimensional Arrays 26](#_Toc300230618)

[PHP Loops 27](#_Toc300230619)

[The while Loop 27](#_Toc300230620)

[The do...while Statement 28](#_Toc300230621)

[The For Loops 28](#_Toc300230622)

[The foreach Loop 29](#_Toc300230623)

[PHP Functions 31](#_Toc300230624)

[PHP Functions 31](#_Toc300230625)

[Create a PHP Function 31](#_Toc300230626)

[PHP Functions - Adding parameters 31](#_Toc300230627)

[PHP Functions - Return values 32](#_Toc300230628)

[PHP Date() Function 34](#_Toc300230629)

[PHP Include File 35](#_Toc300230630)

[Server Side Includes (SSI) 35](#_Toc300230631)

[PHP include() Function 35](#_Toc300230632)

[PHP require() Function 36](#_Toc300230633)

[PHP File Handling 39](#_Toc300230634)

[Opening a File 39](#_Toc300230635)

[Closing a File 39](#_Toc300230636)

[Check End-of-file 40](#_Toc300230637)

[Reading a File Character by Character 40](#_Toc300230638)

[PHP File Upload 41](#_Toc300230639)

[Saving the Uploaded File 42](#_Toc300230640)

[PHP Cookies 44](#_Toc300230641)

[What is a Cookie? 44](#_Toc300230642)

[How to Create a Cookie? 44](#_Toc300230643)

[How to Retrieve a Cookie Value? 44](#_Toc300230644)

[How to Delete a Cookie? 45](#_Toc300230645)

[What if a Browser Does NOT Support Cookies? 45](#_Toc300230646)

[PHP Sessions 46](#_Toc300230647)

[PHP Session Variables 46](#_Toc300230648)

[Starting a PHP Session 46](#_Toc300230649)

[Storing a Session Variable 46](#_Toc300230650)

[Destroying a Session 47](#_Toc300230651)

[PHP Sending E-mails: 48](#_Toc300230652)

[The PHP mail() Function 48](#_Toc300230653)

[PHP Simple E-Mail 48](#_Toc300230654)

[PHP Mail Form 48](#_Toc300230655)

[PHP Error Handling 50](#_Toc300230656)

[Basic Error Handling: Using the die() function 50](#_Toc300230657)

[Creating a Custom Error Handler 50](#_Toc300230658)

[Error Report levels 51](#_Toc300230659)

[Set Error Handler 51](#_Toc300230660)

[Trigger an Error 52](#_Toc300230661)

[Error Logging 53](#_Toc300230662)

[Send an Error Message by E-Mail 53](#_Toc300230663)

[MySQL PHP Database Tutorial 55](#_Toc300230664)

[Getting Started 55](#_Toc300230665)

[Database design: 55](#_Toc300230666)

[Database Normalization: 56](#_Toc300230667)

[Create Database 57](#_Toc300230668)

[Creating the Database Table 57](#_Toc300230669)

[Executing SQL in PHP 58](#_Toc300230670)

[Insert Data 59](#_Toc300230671)

[View Data from Database 60](#_Toc300230672)

[Manage Data 62](#_Toc300230673)

[URL Rewrite in PHP 65](#_Toc300230674)

[Rules 66](#_Toc300230675)

[Conditions 68](#_Toc300230676)

[For Advanced Users 69](#_Toc300230677)

[Tips, Tricks and Advice 70](#_Toc300230678)

[cURL 72](#_Toc300230679)

[Using SOAP with PHP: 75](#_Toc300230680)

[SOAP Envelope 75](#_Toc300230681)

[SOAP Encoding 76](#_Toc300230682)

[Rules for Encoding Types in XML 76](#_Toc300230683)

[Using SOAP for RPC 79](#_Toc300230684)

[RPC and SOAP Body 79](#_Toc300230685)

[RPC and SOAP Header 80](#_Toc300230686)

[Conclusion 83](#_Toc300230687)

# Introduction:

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:

<!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 ?>](http://www.php.net/manual/en/language.basic-syntax.phpmode.php) 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.

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](http://www.php.net/manual/en/install.php) 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](http://www.php.net/manual/en/features.commandline.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](http://gtk.php.net/).

PHP can be [used](http://www.php.net/manual/en/install.php) 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, IIS, and many others. And this includes any web server that can utilize the FastCGI PHP binary, like lighttpd and nginx. PHP works as either a module, or 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 (OOP), or a mixture of them both.

With PHP you are not limited to output HTML. PHP's abilities include 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 auto generate 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](http://www.php.net/manual/en/refs.database.php). Writing a database-enabled web page is incredibly simple using one of the database specific extensions (e.g., for [MySQL](http://www.php.net/manual/en/book.mysqli.php)), or using an abstraction layer like [PDO](http://www.php.net/manual/en/book.pdo.php), or connect to any database supporting the Open Database Connection standard via the [ODBC](http://www.php.net/manual/en/book.uodbc.php) extension. Other databases may utilize [cURL](http://www.php.net/manual/en/book.curl.php) or [sockets](http://www.php.net/manual/en/book.sockets.php), like CouchDB.

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.

## 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, these 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.

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](http://httpd.apache.org/), and of course [» PHP](http://www.php.net/downloads.php). You will most likely want to install a database as well, such as [» MySQL](http://dev.mysql.com/doc/).

You can either install these individually or choose a simpler way. PHP manual has installation instructions for PHP (assuming you already have some web server set up). It is easy to setup a web server with PHP support on any operating system, including MacOSX, Linux and Windows.

On Ubuntu, you can install everything in a single command.  
Just open your terminal window and type:  
  
    sudo apt-get install lamp-server^  
  
(the ^ symbol at the end of this command is mandatory)

Install **WampServer** on your local machine.

Wamp Server provide us local server environment to test PHP pages.

Copy and install Wamp server from transfer.

Installation step are provided at: http://teamtutorials.com/web-development-tutorials/setting-up-a-wamp-server

**Wamp provides phpmyAdmin**

phpMyAdmin can manage a whole MySQL server (needs a super-user) as well as a single database.

**phpmyAdmin** can do:

* browse and drop databases, tables, views, fields and indexes
* create, copy, drop, rename and alter databases, tables, fields and indexes
* maintenance server, databases and tables, with proposals on server configuration
* execute, edit and bookmark any SQL-statement, even batch-queries
* load text files into tables
* create[1](http://72.26.231.66/phpmyadmin/Documentation.html#footnote_1) and read dumps of tables
* export[1](http://72.26.231.66/phpmyadmin/Documentation.html#footnote_1) data to various formats: CSV, XML, PDF, ISO/IEC 26300 - OpenDocument Text and Spreadsheet, Word, Excel and LATEX formats
* administer multiple servers
* manage MySQL users and privileges
* check referential integrity in MyISAM tables
* using Query-by-example (QBE), create complex queries automatically connecting required tables
* create PDF graphics of your Database layout
* search globally in a database or a subset of it
* transform stored data into any format using a set of predefined functions, like displaying BLOB-data as image or download-link
* support InnoDB tables and foreign keys [(see FAQ 3.6)](http://72.26.231.66/phpmyadmin/Documentation.html#faq3_6)
* support mysqli, the improved MySQL extension [(see FAQ 1.17)](http://72.26.231.66/phpmyadmin/Documentation.html#faq1_17)
* communicate in [54 different languages](http://72.26.231.66/phpmyadmin/translators.html)

We can use any text editor but preferable editors are dreamweaver 8, netBeans.

Both installation files are present on Transfers.

**Apache** is the web server, which processes file requests and sends them back. Apache is generally recognised as the most stable and secure server money can buy. Not that money is an issue, as it’s also totally free, like all the tools we’re using.

**mySQL** is the industry standard database server, which will allow interactivity on your websites. Like Apache, mySQL is free, stable and secure. 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 #1 Our first PHP script: hello.php**

<html>  
 <head>  
  <title>PHP Test</title>  
 </head>  
 <body>  
 <?php echo '<p>This is my first PHP Program.</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()](http://in2.php.net/manual/en/function.echo.php) 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](http://in2.php.net/manual/en/install.php) chapter of the PHP 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.

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.

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](http://in2.php.net/manual/en/language.variables.predefined.php), loaded PHP modules, and [configuration](http://in2.php.net/manual/en/configuration.php) settings. Take some time and review this important information.

**Example #2 Get system information from PHP**

<?php phpinfo(); ?>

|  |
| --- |
| **Important Notes:**   * If, when you test the file in a browser, you see the HTML code rather than your greeting, it may be because you launched the page by dragging it to your browser. The URL should start with 'http://' NOT 'file://' * document\_root variable is  located in your web server configuration file * Each code line in PHP must end with a semicolon. The semicolon is a separator and is used to distinguish one set of instructions from another. |

## Comments in PHP

In PHP, we use // to make a single-line comment or /\* and \*/ to make a large comment block.

<html>  
<body>  
  
<?php  
 //This is a comment  
  
 /\*  
 This is  
 a comment  
 block  
 \*/  
?>  
  
</body>  
</html>

## Variables in PHP

A variable is used to store information.

Variables are used for storing values, like text strings, numbers or arrays.

When a variable is declared, it can be used over and over again in your script.

All variables in PHP start with a $ sign symbol.

The correct way of declaring a variable in PHP:

$var\_name = value;

Let's try creating a variable containing a string, and a variable containing a number:

<?php  
$txt="Hello World!";  
$x=16;  
?>

**PHP is a Loosely Typed Language**

In PHP, a variable does not need to be declared before adding a value to it.

In the example above, you see that you do not have to tell PHP which data type the variable is.

PHP automatically converts the variable to the correct data type, depending on its value.

In a strongly typed programming language, you have to declare (define) the type and name of the variable before using it.

In PHP, the variable is declared automatically when you use it.

**Naming Rules for Variables**

* A variable name must start with a letter or an underscore "\_"
* A variable name can only contain alpha-numeric characters and underscores (a-z, A-Z, 0-9, and \_ )
* A variable name should not contain spaces. If a variable name is more than one word, it should be separated with an underscore ($my\_string), or with capitalization ($myString) Camel case notation.

**String Variables in PHP**

String variables are used for values that contain characters.

In this chapter we are going to look at the most common functions and operators used to manipulate strings in PHP.

After we create a string we can manipulate it. A string can be used directly in a function or it can be stored in a variable.

Below, the PHP script assigns the text "Hello World" to a string variable called $txt:

<?php  
$txt="Hello World";  
echo $txt;  
?>

The output of the code above will be: Hello World

**The Concatenation Operator**

There is only one string operator in PHP.

The concatenation operator (.)  is used to put two string values together.

To concatenate two string variables together, use the concatenation operator:

<?php  
$txt1="Hello World!";  
$txt2="What a nice day!";  
echo $txt1 . " " . $txt2;  
?>

The output of the code above will be: Hello World! What a nice day!

If we look at the code above you see that we used the concatenation operator two times. This is because we had to insert a third string (a space character), to separate the two strings.

**Assignment 1:**

Create a php page to display user profile by using all types of php variables.

## PHP String Functions

PHP: indicates the earliest version of PHP that supports the function.

| **Function** | **Description** | **PHP** |
| --- | --- | --- |
| [addcslashes()](http://www.w3schools.com/php/func_string_addcslashes.asp) | Returns a string with backslashes in front of the specified characters | 4 |
| [addslashes()](http://www.w3schools.com/php/func_string_addslashes.asp) | Returns a string with backslashes in front of predefined characters | 3 |
| [bin2hex()](http://www.w3schools.com/php/func_string_bin2hex.asp) | Converts a string of ASCII characters to hexadecimal values | 3 |
| [chop()](http://www.w3schools.com/php/func_string_chop.asp) | Alias of rtrim() | 3 |
| [chr()](http://www.w3schools.com/php/func_string_chr.asp) | Returns a character from a specified ASCII value | 3 |
| [chunk\_split()](http://www.w3schools.com/php/func_string_chunk_split.asp) | Splits a string into a series of smaller parts | 3 |
| [convert\_cyr\_string()](http://www.w3schools.com/php/func_string_convert_cyr_string.asp) | Converts a string from one Cyrillic character-set to another | 3 |
| [convert\_uudecode()](http://www.w3schools.com/php/func_string_convert_uudecode.asp) | Decodes a uuencoded string | 5 |
| [convert\_uuencode()](http://www.w3schools.com/php/func_string_convert_uuencode.asp) | Encodes a string using the uuencode algorithm | 5 |
| [count\_chars()](http://www.w3schools.com/php/func_string_count_chars.asp) | Returns how many times an ASCII character occurs within a string and returns the information | 4 |
| [crc32()](http://www.w3schools.com/php/func_string_crc32.asp) | Calculates a 32-bit CRC for a string | 4 |
| [crypt()](http://www.w3schools.com/php/func_string_crypt.asp) | One-way string encryption (hashing) | 3 |
| [echo()](http://www.w3schools.com/php/func_string_echo.asp) | Outputs strings | 3 |
| [explode()](http://www.w3schools.com/php/func_string_explode.asp) | Breaks a string into an array | 3 |
| [fprintf()](http://www.w3schools.com/php/func_string_fprintf.asp) | Writes a formatted string to a specified output stream | 5 |
| [get\_html\_translation\_table()](http://www.w3schools.com/php/func_string_get_html_translation_table.asp) | Returns the translation table used by htmlspecialchars() and htmlentities() | 4 |
| [hebrev()](http://www.w3schools.com/php/func_string_hebrev.asp) | Converts Hebrew text to visual text | 3 |
| [hebrevc()](http://www.w3schools.com/php/func_string_hebrevc.asp) | Converts Hebrew text to visual text and new lines (\n) into <br /> | 3 |
| [html\_entity\_decode()](http://www.w3schools.com/php/func_string_html_entity_decode.asp) | Converts HTML entities to characters | 4 |
| [htmlentities()](http://www.w3schools.com/php/func_string_htmlentities.asp) | Converts characters to HTML entities | 3 |
| [htmlspecialchars\_decode()](http://www.w3schools.com/php/func_string_htmlspecialchars_decode.asp) | Converts some predefined HTML entities to characters | 5 |
| [htmlspecialchars()](http://www.w3schools.com/php/func_string_htmlspecialchars.asp) | Converts some predefined characters to HTML entities | 3 |
| [implode()](http://www.w3schools.com/php/func_string_implode.asp) | Returns a string from the elements of an array | 3 |
| [join()](http://www.w3schools.com/php/func_string_join.asp) | Alias of implode() | 3 |
| [levenshtein()](http://www.w3schools.com/php/func_string_levenshtein.asp) | Returns the Levenshtein distance between two strings | 3 |
| [localeconv()](http://www.w3schools.com/php/func_string_localeconv.asp) | Returns locale numeric and monetary formatting information | 4 |
| [ltrim()](http://www.w3schools.com/php/func_string_ltrim.asp) | Strips whitespace from the left side of a string | 3 |
| [md5()](http://www.w3schools.com/php/func_string_md5.asp) | Calculates the MD5 hash of a string | 3 |
| [md5\_file()](http://www.w3schools.com/php/func_string_md5_file.asp) | Calculates the MD5 hash of a file | 4 |
| [metaphone()](http://www.w3schools.com/php/func_string_metaphone.asp) | Calculates the metaphone key of a string | 4 |
| [money\_format()](http://www.w3schools.com/php/func_string_money_format.asp) | Returns a string formatted as a currency string | 4 |
| [nl\_langinfo()](http://www.w3schools.com/php/func_string_nl_langinfo.asp) | Returns specific local information | 4 |
| [nl2br()](http://www.w3schools.com/php/func_string_nl2br.asp) | Inserts HTML line breaks in front of each newline in a string | 3 |
| [number\_format()](http://www.w3schools.com/php/func_string_number_format.asp) | Formats a number with grouped thousands | 3 |
| [ord()](http://www.w3schools.com/php/func_string_ord.asp) | Returns the ASCII value of the first character of a string | 3 |
| [parse\_str()](http://www.w3schools.com/php/func_string_parse_str.asp) | Parses a query string into variables | 3 |
| [print()](http://www.w3schools.com/php/func_string_print.asp) | Outputs a string | 3 |
| [printf()](http://www.w3schools.com/php/func_string_printf.asp) | Outputs a formatted string | 3 |
| [quoted\_printable\_decode()](http://www.w3schools.com/php/func_string_quoted_printable_decode.asp) | Decodes a quoted-printable string | 3 |
| [quotemeta()](http://www.w3schools.com/php/func_string_quotemeta.asp) | Quotes meta characters | 3 |
| [rtrim()](http://www.w3schools.com/php/func_string_rtrim.asp) | Strips whitespace from the right side of a string | 3 |
| [setlocale()](http://www.w3schools.com/php/func_string_setlocale.asp) | Sets locale information | 3 |
| [sha1()](http://www.w3schools.com/php/func_string_sha1.asp) | Calculates the SHA-1 hash of a string | 4 |
| [sha1\_file()](http://www.w3schools.com/php/func_string_sha1_file.asp) | Calculates the SHA-1 hash of a file | 4 |
| [similar\_text()](http://www.w3schools.com/php/func_string_similar_text.asp) | Calculates the similarity between two strings | 3 |
| [soundex()](http://www.w3schools.com/php/func_string_soundex.asp) | Calculates the soundex key of a string | 3 |
| [sprintf()](http://www.w3schools.com/php/func_string_sprintf.asp) | Writes a formatted string to a variable | 3 |
| [sscanf()](http://www.w3schools.com/php/func_string_sscanf.asp) | Parses input from a string according to a format | 4 |
| [str\_ireplace()](http://www.w3schools.com/php/func_string_str_ireplace.asp) | Replaces some characters in a string (case-insensitive) | 5 |
| [str\_pad()](http://www.w3schools.com/php/func_string_str_pad.asp) | Pads a string to a new length | 4 |
| [str\_repeat()](http://www.w3schools.com/php/func_string_str_repeat.asp) | Repeats a string a specified number of times | 4 |
| [str\_replace()](http://www.w3schools.com/php/func_string_str_replace.asp) | Replaces some characters in a string (case-sensitive) | 3 |
| [str\_rot13()](http://www.w3schools.com/php/func_string_str_rot13.asp) | Performs the ROT13 encoding on a string | 4 |
| [str\_shuffle()](http://www.w3schools.com/php/func_string_str_shuffle.asp) | Randomly shuffles all characters in a string | 4 |
| [str\_split()](http://www.w3schools.com/php/func_string_str_split.asp) | Splits a string into an array | 5 |
| [str\_word\_count()](http://www.w3schools.com/php/func_string_str_word_count.asp) | Count the number of words in a string | 4 |
| [strcasecmp()](http://www.w3schools.com/php/func_string_strcasecmp.asp) | Compares two strings (case-insensitive) | 3 |
| [strchr()](http://www.w3schools.com/php/func_string_strchr.asp) | Finds the first occurrence of a string inside another string (alias of strstr()) | 3 |
| [strcmp()](http://www.w3schools.com/php/func_string_strcmp.asp) | Compares two strings (case-sensitive) | 3 |
| [strcoll()](http://www.w3schools.com/php/func_string_strcoll.asp) | Locale based string comparison | 4 |
| [strcspn()](http://www.w3schools.com/php/func_string_strcspn.asp) | Returns the number of characters found in a string before any part of some specified characters are found | 3 |
| [strip\_tags()](http://www.w3schools.com/php/func_string_strip_tags.asp) | Strips HTML and PHP tags from a string | 3 |
| [stripcslashes()](http://www.w3schools.com/php/func_string_stripcslashes.asp) | Unquotes a string quoted with addcslashes() | 4 |
| [stripslashes()](http://www.w3schools.com/php/func_string_stripslashes.asp) | Unquotes a string quoted with addslashes() | 3 |
| [stripos()](http://www.w3schools.com/php/func_string_stripos.asp) | Returns the position of the first occurrence of a string inside another string (case-insensitive) | 5 |
| [stristr()](http://www.w3schools.com/php/func_string_stristr.asp) | Finds the first occurrence of a string inside another string (case-insensitive) | 3 |
| [strlen()](http://www.w3schools.com/php/func_string_strlen.asp) | Returns the length of a string | 3 |
| [strnatcasecmp()](http://www.w3schools.com/php/func_string_strnatcasecmp.asp) | Compares two strings using a "natural order" algorithm (case-insensitive) | 4 |
| [strnatcmp()](http://www.w3schools.com/php/func_string_strnatcmp.asp) | Compares two strings using a "natural order" algorithm (case-sensitive) | 4 |
| [strncasecmp()](http://www.w3schools.com/php/func_string_strncasecmp.asp) | String comparison of the first n characters (case-insensitive) | 4 |
| [strncmp()](http://www.w3schools.com/php/func_string_strncmp.asp) | String comparison of the first n characters (case-sensitive) | 4 |
| [strpbrk()](http://www.w3schools.com/php/func_string_strpbrk.asp) | Searches a string for any of a set of characters | 5 |
| [strpos()](http://www.w3schools.com/php/func_string_strpos.asp) | Returns the position of the first occurrence of a string inside another string (case-sensitive) | 3 |
| [strrchr()](http://www.w3schools.com/php/func_string_strrchr.asp) | Finds the last occurrence of a string inside another string | 3 |
| [strrev()](http://www.w3schools.com/php/func_string_strrev.asp) | Reverses a string | 3 |
| [strripos()](http://www.w3schools.com/php/func_string_strripos.asp) | Finds the position of the last occurrence of a string inside another string (case-insensitive) | 5 |
| [strrpos()](http://www.w3schools.com/php/func_string_strrpos.asp) | Finds the position of the last occurrence of a string inside another string (case-sensitive) | 3 |
| [strspn()](http://www.w3schools.com/php/func_string_strspn.asp) | Returns the number of characters found in a string that contains only characters from a specified charlist | 3 |
| [strstr()](http://www.w3schools.com/php/func_string_strstr.asp) | Finds the first occurrence of a string inside another string (case-sensitive) | 3 |
| [strtok()](http://www.w3schools.com/php/func_string_strtok.asp) | Splits a string into smaller strings | 3 |
| [strtolower()](http://www.w3schools.com/php/func_string_strtolower.asp) | Converts a string to lowercase letters | 3 |
| [strtoupper()](http://www.w3schools.com/php/func_string_strtoupper.asp) | Converts a string to uppercase letters | 3 |
| [strtr()](http://www.w3schools.com/php/func_string_strtr.asp) | Translates certain characters in a string | 3 |
| [substr()](http://www.w3schools.com/php/func_string_substr.asp) | Returns a part of a string | 3 |
| [substr\_compare()](http://www.w3schools.com/php/func_string_substr_compare.asp) | Compares two strings from a specified start position (binary safe and optionally case-sensitive) | 5 |
| [substr\_count()](http://www.w3schools.com/php/func_string_substr_count.asp) | Counts the number of times a substring occurs in a string | 4 |
| [substr\_replace()](http://www.w3schools.com/php/func_string_substr_replace.asp) | Replaces a part of a string with another string | 4 |
| [trim()](http://www.w3schools.com/php/func_string_trim.asp) | Strips whitespace from both sides of a string | 3 |
| [ucfirst()](http://www.w3schools.com/php/func_string_ucfirst.asp) | Converts the first character of a string to uppercase | 3 |
| [ucwords()](http://www.w3schools.com/php/func_string_ucwords.asp) | Converts the first character of each word in a string to uppercase | 3 |
| [vfprintf()](http://www.w3schools.com/php/func_string_vfprintf.asp) | Writes a formatted string to a specified output stream | 5 |
| [vprintf()](http://www.w3schools.com/php/func_string_vprintf.asp) | Outputs a formatted string | 4 |
| [vsprintf()](http://www.w3schools.com/php/func_string_vsprintf.asp) | Writes a formatted string to a variable | 4 |
| [wordwrap()](http://www.w3schools.com/php/func_string_wordwrap.asp) | Wraps a string to a given number of characters | 4 |

## PHP Operators

This section lists the different operators used in PHP.

**Arithmetic Operators**

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Description** | **Example** | **Result** |
| + | Addition | x=2, x+2 | 4 |
| - | Subtraction | x=2, 5-x | 3 |
| \* | Multiplication | x=4, x\*5 | 20 |
| / | Division | 15/5, 5/2 | 3, 2.5 |
| % | Modulus (division remainder) | 5%2, 10%8, 10%2 | 1, 2, 0 |
| ++ | Increment | x=5, x++ | x=6 |
| -- | Decrement | x=5, x-- | x=4 |

**Assignment Operators**

|  |  |  |
| --- | --- | --- |
| **Operator** | **Example** | **Is The Same As** |
| = | x=y | x=y |
| += | x+=y | x=x+y |
| -= | x-=y | x=x-y |
| \*= | x\*=y | x=x\*y |
| /= | x/=y | x=x/y |
| .= | x.=y | x=x.y |
| %= | x%=y | x=x%y |

**Comparison Operators**

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| == | is equal to | 5==8 returns false |
| != | is not equal | 5!=8 returns true |
| <> | is not equal | 5<>8 returns true |
| > | is greater than | 5>8 returns false |
| < | is less than | 5<8 returns true |
| >= | is greater than or equal to | 5>=8 returns false |
| <= | is less than or equal to | 5<=8 returns true |

**Logical Operators**

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| && | and | x=6 y=3  (x < 10 && y > 1) returns true |
| || | or | x=6 y=3  (x==5 || y==5) returns false |
| ! | not | x=6 y=3  !(x==y) returns true |

# PHP Forms and User Input

The PHP $\_GET and $\_POST variables are used to retrieve information from forms, like user input.

## PHP Form Handling

The most important thing to notice when dealing with HTML forms and PHP is that any form element in an HTML page will automatically be available to your PHP scripts.

The set of common methods for HTTP/1.1 is defined below

1. **GET**

The GET method means retrieves whatever information (in the form of an entity) is identified by the Request-URI. If the Request-URI refers to a data-producing process, it is the produced data which shall be returned as the entity in the response and not the source text of the process, unless that text happens to be the output of the process.

The semantics of the GET method change to a "conditional GET" if the request message includes an If-Modified-Since, If-Unmodified-Since, If-Match, If-None-Match, or If-Range header field. A conditional GET method requests that the entity be transferred only under the circumstances described by the conditional header field(s). The conditional GET method is intended to reduce unnecessary network usage by allowing cached entities to be refreshed without requiring multiple requests or transferring data already held by the client.

The semantics of the GET method change to a "partial GET" if the request message includes a Range header field. A partial GET requests that only part of the entity be transferred. The partial GET method is intended to reduce unnecessary network usage by allowing partially-retrieved entities to be completed without transferring data already held by the client.

The response to a GET request is cacheable if and only if it meets the requirements for HTTP caching.

1. **HEAD**

The HEAD method is identical to GET except that the server MUST NOT return a message-body in the response. The metainformation contained in the HTTP headers in response to a HEAD request SHOULD be identical to the information sent in response to a GET request. This method can be used for obtaining metainformation about the entity implied by the request without transferring the entity-body itself. This method is often used for testing hypertext links for validity, accessibility, and recent modification.

The response to a HEAD request MAY be cacheable in the sense that the information contained in the response MAY be used to update a previously cached entity from that resource. If the new field values indicate that the cached entity differs from the current entity (as would be indicated by a change in Content-Length, Content-MD5, ETag or Last-Modified), then the cache MUST treat the cache entry as stale.

1. **POST**

The POST method is used to request that the origin server accept the entity enclosed in the request as a new subordinate of the resource identified by the Request-URI in the Request-Line. POST is designed to allow a uniform method to cover the following functions:

- Annotation of existing resources;

- Posting a message to a bulletin board, newsgroup, mailing list,

or similar group of articles;

- Providing a block of data, such as the result of submitting a

form, to a data-handling process;

- Extending a database through an append operation.

The actual function performed by the POST method is determined by the server and is usually dependent on the Request-URI. The posted entity is subordinate to that URI in the same way that a file is subordinate to a directory containing it, a news article is subordinate to a newsgroup to which it is posted, or a record is subordinate to a database.

The action performed by the POST method might not result in a resource that can be identified by a URI. In this case, either 200 (OK) or 204 (No Content) is the appropriate response status, depending on whether or not the response includes an entity that describes the result.

If a resource has been created on the origin server, the response SHOULD be 201 (Created) and contain an entity which describes the status of the request and refers to the new resource, and a Location header.

Responses to this method are not cacheable, unless the response includes appropriate Cache-Control or Expires header fields. However, the 303 (See Other) response can be used to direct the user agent to retrieve a cacheable resource.

1. **PUT**

The PUT method requests that the enclosed entity be stored under the supplied Request-URI. If the Request-URI refers to an already existing resource, the enclosed entity SHOULD be considered as a modified version of the one residing on the origin server. If the Request-URI does not point to an existing resource, and that URI is capable of being defined as a new resource by the requesting user agent, the origin server can create the resource with that URI.

If a new resource is created, the origin server MUST inform the user agent via the 201 (Created) response. If an existing resource is modified, either the 200 (OK) or 204 (No Content) response codes SHOULD be sent to indicate successful completion of the request. If the resource could not be created or modified with the Request-URI, an appropriate error response SHOULD be given that reflects the nature of the problem. The recipient of the entity MUST NOT ignore any Content-\* (e.g. Content-Range) headers that it does not understand or implement and MUST return a 501 (Not Implemented) response in such cases.

If the request passes through a cache and the Request-URI identifies one or more currently cached entities, those entries SHOULD be treated as stale. Responses to this method are not cacheable.

1. **DELETE**

The DELETE method requests that the origin server delete the resource identified by the Request-URI. This method MAY be overridden by human intervention (or other means) on the origin server. The client cannot be guaranteed that the operation has been carried out, even if the status code returned from the origin server indicates that the action has been completed successfully. However, the server SHOULD NOT indicate success unless, at the time the response is given, it intends to delete the resource or move it to an inaccessible location.

A successful response SHOULD be 200 (OK) if the response includes an entity describing the status, 202 (Accepted) if the action has not yet been enacted, or 204 (No Content) if the action has been enacted but the response does not include an entity.

If the request passes through a cache and the Request-URI identifies one or more currently cached entities, those entries SHOULD be treated as stale. Responses to this method are not cacheable.

1. **TRACE**

The TRACE method is used to invoke a remote, application-layer loop- back of the request message. The final recipient of the request SHOULD reflect the message received back to the client as the entity-body of a 200 (OK) response. The final recipient is either the origin server or the first proxy or gateway to receive a Max-Forwards value of zero (0) in the request. A TRACE request MUST NOT include an entity.

TRACE allows the client to see what is being received at the other end of the request chain and use that data for testing or diagnostic information. The value of the Via header field is of particular interest, since it acts as a trace of the request chain. Use of the Max-Forwards header field allows the client to limit the length of the request chain, which is useful for testing a chain of proxies forwarding messages in an infinite loop.

If the request is valid, the response SHOULD contain the entire request message in the entity-body, with a Content-Type of "message/http". Responses to this method MUST NOT be cached.

**Example**

The example below contains an HTML form with two input fields and a submit button:

<html>  
<body>  
<form action="welcome.php" method="post">  
Name: <input type="text" name="fname" />  
Age: <input type="text" name="age" />  
<input type="submit" />  
</form>  
</body>  
</html>

When a user fills out the form above and click on the submit button, the form data is sent to a PHP file, called "welcome.php":

"welcome.php" looks like this

<html>  
<body>  
Welcome <?php echo $\_POST["fname"]; ?>!<br />  
You are <?php echo $\_POST["age"]; ?> years old.  
</body>  
</html>

Output could be something like this:

Welcome John!  
You are 28 years old.

The PHP $\_GET and $\_POST functions will be explained in the next chapters.

## Form Validation

User input should be validated on the browser whenever possible (by client scripts). Browser validation is faster and reduces the server load.

You should consider server validation if the user input will be inserted into a database. A good way to validate a form on the server is to post the form to itself, instead of jumping to a different page. The user will then get the error messages on the same page as the form. This makes it easier to discover the error.

## PHP $\_GET Function

The built-in $\_GET function is used to collect values in a form with method="get".

**The $\_GET Function:**

The built-in $\_GET function is used to collect values from a form sent with method="get".

Information sent from a form with the GET method is visible to everyone (it will be displayed in the browser's address bar) and has limits on the amount of information to send.

**Example:**

<form action="welcome.php" method="get">  
Name: <input type="text" name="fname" />  
Age: <input type="text" name="age" />  
<input type="submit" />  
</form>

When the user clicks the "Submit" button, the URL sent to the server could look something like this:

<http://localhost/welcome.php?fname=Peter&age=37>

The "welcome.php" file can now use the $\_GET function to collect form data (the names of the form fields will automatically be the keys in the $\_GET array):

Welcome <?php echo $\_GET["fname"]; ?>.<br />  
You are <?php echo $\_GET["age"]; ?> years old!

**When to use method="get"?**

When using method="get" in HTML forms, all variable names and values are displayed in the URL.

**Note:** This method should not be used when sending passwords or other sensitive information!

However, because the variables are displayed in the URL, it is possible to bookmark the page. This can be useful in some cases.

**Note:** The get method is not suitable for very large variable values. It should not be used with values exceeding 2000 characters.

## PHP $\_POST Function

The built-in $\_POST function is used to collect values in a form with method="post".

**The $\_POST Function**

The built-in $\_POST function is used to collect values from a form sent with method="post".

Information sent from a form with the POST method is invisible to others and has no limits on the amount of information to send.

**Note:** However, there is an 8 Mb max size for the POST method, by default (can be changed by setting the post\_max\_size in the php.ini file).

**Example**

<form action="welcome.php" method="post">  
Name: <input type="text" name="fname" />  
Age: <input type="text" name="age" />  
<input type="submit" />  
</form>

When the user clicks the "Submit" button, the URL will look like this:

<http://localhost/welcome.php>

The "welcome.php" file can now use the $\_POST function to collect form data (the names of the form fields will automatically be the keys in the $\_POST array):

Welcome <?php echo $\_POST["fname"]; ?>!<br />  
You are <?php echo $\_POST["age"]; ?> years old.

**When to use method="post"?**

Information sent from a form with the POST method is invisible to others and has no limits on the amount of information to send.

However, because the variables are not displayed in the URL, it is not possible to bookmark the page.

**The PHP $\_REQUEST Function**

The PHP built-in $\_REQUEST function contains the contents of both $\_GET, $\_POST, and $\_COOKIE.

The $\_REQUEST function can be used to collect form data sent with both the GET and POST methods.

**Example**

Welcome <?php echo $\_REQUEST["fname"]; ?>!<br />  
You are <?php echo $\_REQUEST["age"]; ?> years old.

**Assignment 2:**

Create a PHP page to provide all calculations for 2 numbers by using arithmetic operators.

Create a PHP page to provide all comparison operators on 2 strings and 2 numbers.

Create a PHP page to use all logical operators.

# Conditional Statements

Conditional statements are used to perform different actions based on different conditions.

Very often when you write code, you want to perform different actions for different decisions.

You can use conditional statements in your code to do this.

In PHP we have the following conditional statements:

* if statement - use this statement to execute some code only if a specified condition is true
* if...else statement - use this statement to execute some code if a condition is true and another code if the condition is false
* if...elseif....else statement - use this statement to select one of several blocks of code to be executed
* switch statement - use this statement to select one of many blocks of code to be executed

## The if Statement

Use the if statement to execute some code only if a specified condition is true.

**Syntax**

*if (condition) code to be executed if condition is true;*

The following example will output "Have a nice weekend!" if the current day is Friday:

<html>  
<body>  
  
<?php  
$d=date("D");  
if ($d=="Fri") echo "Have a nice weekend!";  
?>  
  
</body>  
</html>

**Notice** that there is no ..else.. in this syntax. The code is executed only if **the specified condition is true.**

## The if...else Statement

Use the if....else statement to execute some code if a condition is true and another code if a condition is false.

**Syntax**

*if (condition)*

*code to be executed if condition is true;*

*else*

*code to be executed if condition is false;*

**Example**

The following example will output "Have a nice weekend!" if the current day is Friday, otherwise it will output "Have a nice day!":

<html>  
<body>  
<?php  
$d=date("D");  
if ($d=="Fri")

echo "Have a nice weekend!";

else

echo "Have a nice day!";

?>

</body>

</html>

## The if...elseif....else Statement

Use the if....elseif...else statement to select one of several blocks of code to be executed.

**Syntax**

*if (condition)*

*code to be executed if condition is true;*

*elseif (condition)*

*code to be executed if condition is true;*

*else*

*code to be executed if condition is false;*

**Example**

The following example will output "Have a nice weekend!" if the current day is Friday, and "Have a nice Sunday!" if the current day is Sunday. Otherwise it will output "Have a nice day!":

<html>  
<body>  
<?php  
$d=date("D");

if ($d=="Fri")

echo "Have a nice weekend!";

elseif ($d=="Sun")

echo "Have a nice Sunday!";

else

echo "Have a nice day!";

?>  
</body>  
</html>

## PHP Switch Statement

Conditional statements are used to perform different actions based on different conditions.

Use the switch statement to select one of many blocks of code to be executed.

**Syntax:**

switch (n)  
{  
case label1:  
  code to be executed if n=label1;  
  break;  
case label2:  
  code to be executed if n=label2;  
  break;  
default:  
  code to be executed if n is different from both label1 and label2;  
}

This is how it works: First we have a single expression n (most often a variable), that is evaluated once. The value of the expression is then compared with the values for each case in the structure. If there is a match, the block of code associated with that case is executed. Use break to prevent the code from running into the next case automatically. The default statement is used if no match is found.

**Example:**

<html>  
<body>  
  
<?php  
switch ($x)  
{  
case 1:  
  echo "Number 1";  
  break;  
case 2:  
  echo "Number 2";  
  break;  
case 3:  
  echo "Number 3";  
  break;  
default:  
  echo "No number between 1 and 3";  
}  
?>  
  
</body>  
</html>

# PHP Arrays

An array stores multiple values in one single variable.

## What is an Array?

A variable is a storage area holding a number or text. The problem is, a variable will hold only one value.

An array is a special variable, which can store multiple values in one single variable.

If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

$cars1="Saab";  
$cars2="Volvo";  
$cars3="BMW";

However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?

The best solution here is to use an array!

An array can hold all your variable values under a single name. And you can access the values by referring to the array name.

Each element in the array has its own index so that it can be easily accessed.

In PHP, there are three kind of arrays:

* **Numeric array** - An array with a numeric index
* **Associative array** - An array where each ID key is associated with a value
* **Multidimensional array** - An array containing one or more arrays

## Numeric Arrays

A numeric array stores each array element with a numeric index.

There are two methods to create a numeric array.

1. In the following example the index are automatically assigned (the index starts at 0):

$cars=array("Saab","Volvo","BMW","Toyota");

2. In the following example we assign the index manually:

$cars[0]="Saab";  
$cars[1]="Volvo";  
$cars[2]="BMW";  
$cars[3]="Toyota";

**Example**

In the following example you access the variable values by referring to the array name and index:

<?php  
$cars[0]="Saab";  
$cars[1]="Volvo";  
$cars[2]="BMW";  
$cars[3]="Toyota";   
echo $cars[0] . " and " . $cars[1] . " are Swedish cars.";  
?>

The code above will output:

Saab and Volvo are Swedish cars.

## Associative Arrays

An associative array, each ID key is associated with a value.

When storing data about specific named values, a numerical array is not always the best way to do it.

With associative arrays we can use the values as keys and assign values to them.

**Example 1**

In this example we use an array to assign ages to the different persons:

$ages = array("Peter"=>32, "Quagmire"=>30, "Joe"=>34);

**Example 2**

This example is the same as example 1, but shows a different way of creating the array:

$ages['Peter'] = "32";  
$ages['Quagmire'] = "30";  
$ages['Joe'] = "34";

The ID keys can be used in a script:

<?php  
$ages['Peter'] = "32";  
$ages['Quagmire'] = "30";  
$ages['Joe'] = "34";  
  
echo "Peter is " . $ages['Peter'] . " years old.";  
?>

The code above will output:

Peter is 32 years old.

## Multidimensional Arrays

In a multidimensional array, each element in the main array can also be an array. And each element in the sub-array can be an array, and so on.

**Example**

In this example we create a multidimensional array, with automatically assigned ID keys:

$families = array("Griffin"=>array("Peter", "Lois", "Megan"),

”Quagmire"=>array("Glenn"),

"Brown"=>array("Cleveland", "Loretta", "Junior"));

The array above would look like this if written to the output:

Array  
(

[Griffin] => Array ([0] => Peter [1] => Lois [2] => Megan)

[Quagmire] => Array ([0] => Glenn)

[Brown] => Array([0] => Cleveland [1] => Loretta [2] => Junior)

)

**Example 2**

Let’s try displaying a single value from the array above:

echo ‘Is ‘ . $families['Griffin'][2] . ‘ a part of the Griffin family?’;

The code above will output: Is Megan a part of the Griffin family?

# PHP Loops

Loops execute a block of code a specified number of times, or while a specified condition is true.

Often when you write code, you want the same block of code to run over and over again in a row. Instead of adding several almost equal lines in a script we can use loops to perform a task like this.

In PHP, we have the following looping statements:

* **while** - loops through a block of code while a specified condition is true
* **do...while** - loops through a block of code once, and then repeats the loop as long as a specified condition is true
* **for** - loops through a block of code a specified number of times
* **foreach** - loops through a block of code for each element in an array

## The while Loop

The while loop executes a block of code while a condition is true.

**Syntax**

while (condition)  
  {  
  code to be executed;  
  }

**Example**

The example below defines a loop that starts with i=1. The loop will continue to run as long as i is less than, or equal to 5. i will increase by 1 each time the loop runs:

<html>  
<body>  
  
<?php  
$i=1;  
while($i<=5)  
  {  
  echo "The number is " . $i . "<br />";  
  $i++;  
  }  
?>  
  
</body>  
</html>

**Output:**

The number is 1  
The number is 2  
The number is 3  
The number is 4  
The number is 5

## The do...while Statement

The do...while statement will always execute the block of code once, it will then check the condition, and repeat the loop while the condition is true.

**Syntax**

do  
  {  
  code to be executed;  
  }  
while (condition);

**Example**

The example below defines a loop that starts with i=1. It will then increment i with 1, and write some output. Then the condition is checked, and the loop will continue to run as long as i is less than, or equal to 5:

<html>  
<body>  
  
<?php  
$i=1;  
do  
  {  
  $i++;  
  echo "The number is " . $i . "<br />";  
  }  
while ($i<=5);  
?>  
  
</body>  
</html>

**Output:**

The number is 2  
The number is 3  
The number is 4  
The number is 5…..

## The For Loops

Loops execute a block of code a specified number of times, or while a specified condition is true.

The for loop is used when you know in advance how many times the script should run.

**Syntax**

for (init; condition; increment)  
  {  
  code to be executed;  
  }

**Parameters:**

* **init**: Mostly used to set a counter (but can be any code to be executed once at the beginning of the loop)
* **condition**: Evaluated for each loop iteration. If it evaluates to TRUE, the loop continues. If it evaluates to FALSE, the loop ends.
* **increment**: Mostly used to increment a counter (but can be any code to be executed at the end of the loop)

**Note:** Each of the parameters above can be empty, or have multiple expressions (separated by commas).

**Example**

The example below defines a loop that starts with i=1. The loop will continue to run as long as i is less than, or equal to 5. i will increase by 1 each time the loop runs:

<html>  
<body>  
  
<?php  
for ($i=1; $i<=5; $i++)  
  {  
  echo ‘The number is ‘ . $i . ‘<br />’;  
  }  
?>  
  
</body>  
</html>

**Output:**

The number is 1  
The number is 2  
The number is 3  
The number is 4  
The number is 5

## The foreach Loop

The foreach loop is used to loop through arrays.

**Syntax**

foreach ($*array* as$*value*)  
  {  
  *code to be executed;*  
  }

For every loop iteration, the value of the current array element is assigned to $value (and the array pointer is moved by one) - so on the next loop iteration, you'll be looking at the next array value.

**Example**

The following example demonstrates a loop that will print the values of the given array:

<html>  
<body>  
  
<?php  
$x=array("one","two","three");  
foreach ($x as $value)  
  {  
  echo $value . "<br />";  
  }  
?>  
  
</body>  
</html>

**Output:**

one  
two  
three

# PHP Functions

The real power of PHP comes from its functions.

In PHP, there are more than 700 built-in functions.

## PHP Functions

In this chapter we will show you how to create your own functions.

To keep the script from being executed when the page loads, you can put it into a function.

A function will be executed by a call to the function.

You may call a function from anywhere within a page.

## Create a PHP Function

A function will be executed by a call to the function.

**Syntax**

function functionName()  
{  
code to be executed;  
}

**PHP function guidelines:**

* Give the function a name that reflects what the function does
* The function name can start with a letter or underscore (not a number)

**Example**

A simple function that writes my name when it is called:

<html>  
<body>  
<?php  
function writeName()  
{  
echo "Kai Jim Refsnes";  
}  
echo "My name is ";  
writeName();  
?>  
</body>  
</html>

**Output:**

My name is Kai Jim Refsnes

## PHP Functions - Adding parameters

To add more functionality to a function, we can add parameters. A parameter is just like a variable.

Parameters are specified after the function name, inside the parentheses.

**Example 1**

The following example will write different first names, but equal last name:

<html>  
<body>  
<?php  
function writeName($fname)  
{  
echo $fname . " Refsnes.<br />";  
}  
echo "My name is ";  
writeName("Kai Jim");  
echo "My sister's name is ";  
writeName("Hege");  
echo "My brother's name is ";  
writeName("Stale");  
?>  
</body>  
</html>

**Output:**

My name is Kai Jim Refsnes.  
My sister's name is Hege Refsnes.  
My brother's name is Stale Refsnes.

**Example 2**

The following function has two parameters:

<html>  
<body>  
<?php  
function writeName($fname,$punctuation)  
{  
echo $fname . " Refsnes" . $punctuation . "<br />";  
}  
echo "My name is ";  
writeName("Kai Jim",".");  
echo "My sister's name is ";  
writeName("Hege","!");  
echo "My brother's name is ";  
writeName("Ståle","?");  
?>  
</body>  
</html>

**Output:**

My name is Kai Jim Refsnes.  
My sister's name is Hege Refsnes!  
My brother's name is Ståle Refsnes?

## PHP Functions - Return values

To let a function return a value, use the return statement.

**Example**

<html>  
<body>  
<?php  
function add($x,$y)  
{  
$total=$x+$y;  
return $total;  
}  
echo "1 + 16 = " . add(1,16);  
?>  
</body>  
</html>

**Output:**

1 + 16 = 17

## PHP Date() Function

The PHP date() function is used to format a time and/or date.

**The PHP Date() Function**

The PHP date() function formats a timestamp to a more readable date and time.

TipA timestamp is a sequence of characters, denoting the date and/or time at which a certain event occurred.

**Syntax**

date(format,timestamp)

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| format | Required. Specifies the format of the timestamp |
| timestamp | Optional. Specifies a timestamp. Default is the current date and time |

**PHP Date() - Format the Date**

Here are some characters that can be used:

* d - Represents the day of the month (01 to 31)
* m - Represents a month (01 to 12)
* Y - Represents a year (in four digits)

A list of all the characters that can be used in the format parameter, can be found in our [PHP Date reference](http://www.w3schools.com/php/php_ref_date.asp).

Other characters, like"/", ".", or "-" can also be inserted between the letters to add additional formatting:

<?php  
echo date("Y/m/d") . "<br />";  
echo date("Y.m.d") . "<br />";  
echo date("Y-m-d");  
?>

The output of the code above could be something like this:

2009/05/11  
2009.05.11  
2009-05-11

**PHP Date() - Adding a Timestamp**

The optional timestamp parameter in the date() function specifies a timestamp. If you do not specify a timestamp, the current date and time will be used.

The mktime() function returns the Unix timestamp for a date.

The Unix timestamp contains the number of seconds between the Unix Epoch (January 1 1970 00:00:00 GMT) and the time specified.

**Syntax for mktime()**

mktime(hour,minute,second,month,day,year,is\_dst)

To go one day in the future we simply add one to the day argument of mktime():

<?php  
$tomorrow = mktime(0,0,0,date("m"),date("d")+1,date("Y"));  
echo "Tomorrow is ".date("Y/m/d", $tomorrow);  
?>

The output of the code above could be something like this:

Tomorrow is 2009/05/12

# PHP Include File

## Server Side Includes (SSI)

You can insert the content of one PHP file into another PHP file before the server executes it, with the include() or require() function.

The two functions are identical in every way, except how they handle errors:

* include() generates a warning, but the script will continue execution
* require() generates a fatal error, and the script will stop

These two functions are used to create functions, headers, footers, or elements that will be reused on multiple pages.

Server side includes saves a lot of work. This means that you can create a standard header, footer, or menu file for all your web pages. When the header needs to be updated, you can only update the include file, or when you add a new page to your site, you can simply change the menu file (instead of updating the links on all your web pages).

## PHP include() Function

The include() function takes all the content in a specified file and includes it in the current file.

If an error occurs, the include() function generates a warning, but the script will continue execution.

**Example 1**

Assume that you have a standard header file, called "header.php". To include the header file in a page, use the include() function:

<html>  
<body>  
  
<?php include("header.php"); ?>  
<h1>Welcome to my home page!</h1>  
<p>Some text.</p>  
  
</body>  
</html>

**Example 2**

Assume we have a standard menu file, called "menu.php", that should be used on all pages:

<a href="/default.php">Home</a>  
<a href="/tutorials.php">Tutorials</a>  
<a href="/references.php">References</a>  
<a href="/examples.php">Examples</a>   
<a href="/about.php">About Us</a>   
<a href="/contact.php">Contact Us</a>

All pages in the Web site should include this menu file. Here is how it can be done:

<html>  
<body>  
  
<div class="leftmenu">  
<?php include("menu.php"); ?>  
</div>  
  
<h1>Welcome to my home page.</h1>  
<p>Some text.</p>  
  
</body>  
</html>

If you look at the source code of the page above (in a browser), it will look like this:

<html>  
<body>  
  
<div class="leftmenu">  
<a href="/default.php">Home</a>  
<a href="/tutorials.php">Tutorials</a>  
<a href="/references.php">References</a>  
<a href="/examples.php">Examples</a>   
<a href="/about.php">About Us</a>   
<a href="/contact.php">Contact Us</a>  
</div>  
  
<h1>Welcome to my home page!</h1>  
<p>Some text.</p>  
  
</body>  
</html>

## PHP require() Function

The require() function is identical to include(), except that it handles errors differently.

If an error occurs, the include() function generates a warning, but the script will continue execution. The require() generates a fatal error, and the script will stop.

**Error Example include() Function**

<html>  
<body>  
  
<?php  
include("wrongFile.php");  
echo "Hello World!";  
?>  
  
</body>  
</html>

Error message:

**Warning:** include(wrongFile.php) [function.include]:  
failed to open stream:  
No such file or directory in C:\home\website\test.php on line 5  
  
**Warning:** include() [function.include]:  
Failed opening 'wrongFile.php' for inclusion  
(include\_path='.;C:\php5\pear')  
in C:\home\website\test.php on line 5  
  
Hello World!

Notice that the echo statement is executed! This is because a Warning does not stop the script execution.

**Error Example require() Function**

Now, let's run the same example with the require() function.

<html>  
<body>  
  
<?php  
require("wrongFile.php");  
echo "Hello World!";  
?>  
  
</body>  
</html>

Error message:

**Warning:** require(wrongFile.php) [function.require]:  
failed to open stream:  
No such file or directory in C:\home\website\test.php on line 5  
  
**Fatal error:** require() [function.require]:  
Failed opening required 'wrongFile.php'  
(include\_path='.;C:\php5\pear')  
in C:\home\website\test.php on line 5

The echo statement is not executed, because the script execution stopped after the fatal error.

It is recommended to use the require() function instead of include(), because scripts should not continue after an error.

**Assignment 3**: Create a php page to display a website frame and create separate pages for header, footer, left menu, right menu to traverse on pages like contact us page, index page, about page, FAQ page.

|  |  |  |
| --- | --- | --- |
| header.php | | |
| leftMenu.php | main.php | rightMenu.php |
| footer.php | | |

Middle section of all pages will have their own content. Ex faq.php will have faq content.

# PHP File Handling

## Opening a File

The fopen() function is used to open files in PHP.

The fopen() function is used to open files in PHP.

The first parameter of this function contains the name of the file to be opened and the second parameter specifies in which mode the file should be opened:

<html>  
<body>  
  
<?php  
$file=fopen("welcome.txt","r");  
?>  
  
</body>  
</html>

The file may be opened in one of the following modes:

|  |  |
| --- | --- |
| **Modes** | **Description** |
| r | Read only. Starts at the beginning of the file |
| r+ | Read/Write. Starts at the beginning of the file |
| w | Write only. Opens and clears the contents of file; or creates a new file if it doesn't exist |
| w+ | Read/Write. Opens and clears the contents of file; or creates a new file if it doesn't exist |
| a | Append. Opens and writes to the end of the file or creates a new file if it doesn't exist |
| a+ | Read/Append. Preserves file content by writing to the end of the file |
| x | Write only. Creates a new file. Returns FALSE and an error if file already exists |
| x+ | Read/Write. Creates a new file. Returns FALSE and an error if file already exists |

**Note:** If the fopen() function is unable to open the specified file, it returns 0 (false).

**Example**

The following example generates a message if the fopen() function is unable to open the specified file:

<html>  
<body>  
  
<?php  
$file=fopen("welcome.txt","r") or exit("Unable to open file!");  
?>  
  
</body>  
</html>

## Closing a File

The fclose() function is used to close an open file:

<?php  
$file = fopen("test.txt","r");  
  
//some code to be executed  
  
fclose($file);  
?>

## Check End-of-file

The feof() function checks if the "end-of-file" (EOF) has been reached.  
  
The feof() function is useful for looping through data of unknown length.

Note: You cannot read from files opened in w, a, and x mode!

if (feof($file)) echo "End of file";

Reading a File Line by Line

The fgets() function is used to read a single line from a file.

Note: After a call to this function the file pointer has moved to the next line.

**Example**

The example below reads a file line by line, until the end of file is reached:

<?php  
$file = fopen("welcome.txt", "r") or exit("Unable to open file!");  
//Output a line of the file until the end is reached  
while(!feof($file))  
  {  
  echo fgets($file). "<br />";  
  }  
fclose($file);  
?>

## Reading a File Character by Character

The fgetc() function is used to read a single character from a file.

Note: After a call to this function the file pointer moves to the next character.

**Example**

The example below reads a file character by character, until the end of file is reached:

<?php  
$file=fopen("welcome.txt","r") or exit("Unable to open file!");  
while (!feof($file))  
  {  
  echo fgetc($file);  
  }  
fclose($file);  
?>

## PHP File Upload

With PHP, it is possible to upload files to the server.

**Create an Upload-File Form**

To allow users to upload files from a form can be very useful.

Look at the following HTML form for uploading files:

<html>  
<body>  
  
<form action="upload\_file.php" method="post"  
enctype="multipart/form-data">  
<label for="file">Filename:</label>  
<input type="file" name="file" id="file" />   
<br />  
<input type="submit" name="submit" value="Submit" />  
</form>  
  
</body>  
</html>

Notice the following about the HTML form above:

* The enctype attribute of the <form> tag specifies which content-type to use when submitting the form. "multipart/form-data" is used when a form requires binary data, like the contents of a file, to be uploaded
* The type="file" attribute of the <input> tag specifies that the input should be processed as a file. For example, when viewed in a browser, there will be a browse-button next to the input field

**Note:** Allowing users to upload files is a big security risk. Only permit trusted users to perform file uploads.

**Create The Upload Script**

The "upload\_file.php" file contains the code for uploading a file:

<?php  
if ($\_FILES["file"]["error"] > 0)  
  {  
  echo "Error: " . $\_FILES["file"]["error"] . "<br />";  
  }  
else  
  {  
  echo "Upload: " . $\_FILES["file"]["name"] . "<br />";  
  echo "Type: " . $\_FILES["file"]["type"] . "<br />";  
  echo "Size: " . ($\_FILES["file"]["size"] / 1024) . " Kb<br />";  
  echo "Stored in: " . $\_FILES["file"]["tmp\_name"];  
  }  
?>

By using the global PHP $\_FILES array you can upload files from a client computer to the remote server.

The first parameter is the form's input name and the second index can be either "name", "type", "size", "tmp\_name" or "error". Like this:

* $\_FILES["file"]["name"] - the name of the uploaded file
* $\_FILES["file"]["type"] - the type of the uploaded file
* $\_FILES["file"]["size"] - the size in bytes of the uploaded file
* $\_FILES["file"]["tmp\_name"] - the name of the temporary copy of the file stored on the server
* $\_FILES["file"]["error"] - the error code resulting from the file upload

This is a very simple way of uploading files. For security reasons, you should add restrictions on what the user is allowed to upload.

**Assignment 4**:

1. Write a php script to read a text file and display content of text file on browser.

2. User will enter their opinions on a textarea and click on post button. This action will create a txt file for that post and will write user’s post in that file.

Every file will have name= current timestamp (datetime)

3. display list of opinions sort of time of post and display the posts on browser.

**Restrictions on Upload**

In this script we add some restrictions to the file upload. The user may only upload .gif or .jpeg files and the file size must be under 20 kb:

<?php  
if ((($\_FILES["file"]["type"] == "image/gif")  
|| ($\_FILES["file"]["type"] == "image/jpeg")  
|| ($\_FILES["file"]["type"] == "image/pjpeg"))  
&& ($\_FILES["file"]["size"] < 20000))  
  {  
  if ($\_FILES["file"]["error"] > 0)  
    {  
    echo "Error: " . $\_FILES["file"]["error"] . "<br />";  
    }  
  else  
    {  
    echo "Upload: " . $\_FILES["file"]["name"] . "<br />";  
    echo "Type: " . $\_FILES["file"]["type"] . "<br />";  
    echo "Size: " . ($\_FILES["file"]["size"] / 1024) . " Kb<br />";  
    echo "Stored in: " . $\_FILES["file"]["tmp\_name"];  
    }  
  }  
else  
  {  
  echo "Invalid file";  
  }  
?>

**Note:** For IE to recognize jpg files the type must be pjpeg, for FireFox it must be jpeg.

## Saving the Uploaded File

The examples above create a temporary copy of the uploaded files in the PHP temp folder on the server.

The temporary copied files disappears when the script ends. To store the uploaded file we need to copy it to a different location:

<?php  
if ((($\_FILES["file"]["type"] == "image/gif")  
|| ($\_FILES["file"]["type"] == "image/jpeg")  
|| ($\_FILES["file"]["type"] == "image/pjpeg"))  
&& ($\_FILES["file"]["size"] < 20000))  
  {  
  if ($\_FILES["file"]["error"] > 0)  
    {  
    echo "Return Code: " . $\_FILES["file"]["error"] . "<br />";  
    }  
  else  
    {  
    echo "Upload: " . $\_FILES["file"]["name"] . "<br />";  
    echo "Type: " . $\_FILES["file"]["type"] . "<br />";  
    echo "Size: " . ($\_FILES["file"]["size"] / 1024) . " Kb<br />";  
    echo "Temp file: " . $\_FILES["file"]["tmp\_name"] . "<br />";  
  
    if (file\_exists("upload/" . $\_FILES["file"]["name"]))  
      {  
      echo $\_FILES["file"]["name"] . " already exists. ";  
      }  
    else  
      {  
      move\_uploaded\_file($\_FILES["file"]["tmp\_name"],  
      "upload/" . $\_FILES["file"]["name"]);  
      echo "Stored in: " . "upload/" . $\_FILES["file"]["name"];  
      }  
    }  
  }  
else  
  {  
  echo "Invalid file";  
  }  
?>

The script above checks if the file already exists, if it does not, it copies the file to the specified folder.

**Note:** This example saves the file to a new folder called "upload"

**Assignment 5:** Write a php page to upload only .jpg image on server and name the uploaded file as auto increment number.

# PHP Cookies

A cookie is often used to identify a user.

## What is a Cookie?

A cookie is often used to identify a user. A cookie is a small file that the server embeds on the user's computer. Each time the same computer requests a page with a browser, it will send the cookie too. With PHP, you can both create and retrieve cookie values.

## How to Create a Cookie?

The setcookie() function is used to set a cookie.

**Note:** The setcookie() function must appear BEFORE the <html> tag.

**Syntax**

setcookie(name, value, expire, path, domain);

**Example 1**

In the example below, we will create a cookie named "user" and assign the value "Alex Porter" to it. We also specify that the cookie should expire after one hour:

<?php  
setcookie("user", "Alex Porter", time()+3600);  
?>  
  
<html>  
.....

**Note:** The value of the cookie is automatically URLencoded when sending the cookie, and automatically decoded when received (to prevent URLencoding, use setrawcookie() instead).

**Example 2**

You can also set the expiration time of the cookie in another way. It may be easier than using seconds.

<?php  
$expire=time()+60\*60\*24\*30;  
setcookie("user", "Alex Porter", $expire);  
?>  
  
<html>  
.....

In the example above the expiration time is set to a month (*60 sec \* 60 min \* 24 hours \* 30 days*).

## How to Retrieve a Cookie Value?

The PHP $\_COOKIE variable is used to retrieve a cookie value.   
  
In the example below, we retrieve the value of the cookie named "user" and display it on a page:

<?php  
// Print a cookie  
echo $\_COOKIE["user"];  
  
// A way to view all cookies  
print\_r($\_COOKIE);  
?>

In the following example we use the isset() function to find out if a cookie has been set:

<html>  
<body>  
  
<?php  
if (isset($\_COOKIE["user"]))  
  echo "Welcome " . $\_COOKIE["user"] . "!<br />";  
else  
  echo "Welcome guest!<br />";  
?>  
  
</body>  
</html>

## How to Delete a Cookie?

When deleting a cookie you should assure that the expiration date is in the past.

Delete example:

<?php  
// set the expiration date to one hour ago  
setcookie("user", "", time()-3600);  
?>

## What if a Browser Does NOT Support Cookies?

If your application deals with browsers that do not support cookies, you will have to use other methods to pass information from one page to another in your application. One method is to pass the data through forms (forms and user input are described earlier in this tutorial).

The form below passes the user input to "welcome.php" when the user clicks on the "Submit" button:

<html>  
<body>  
  
<form action="welcome.php" method="post">  
Name: <input type="text" name="name" />  
Age: <input type="text" name="age" />  
<input type="submit" />  
</form>  
  
</body>  
</html>

Retrieve the values in the "welcome.php" file like this:

<html>  
<body>  
  
Welcome <?php echo $\_POST["name"]; ?>.<br />  
You are <?php echo $\_POST["age"]; ?> years old.  
  
</body>  
</html>

# PHP Sessions

A PHP session variable is used to store information about, or change settings for a user session. Session variables hold information about one single user, and are available to all pages in one application.

## PHP Session Variables

When you are working with an application, you open it, do some changes and then you close it. This is much like a Session. The computer knows who you are. It knows when you start the application and when you end. But on the internet there is one problem: the web server does not know who you are and what you do because the HTTP address doesn't maintain state.

A PHP session solves this problem by allowing you to store user information on the server for later use (i.e. username, shopping items, etc). However, session information is temporary and will be deleted after the user has left the website. If you need a permanent storage you may want to store the data in a database.

Sessions work by creating a unique id (UID) for each visitor and store variables based on this UID. The UID is either stored in a cookie or is propagated in the URL.

## Starting a PHP Session

Before you can store user information in your PHP session, you must first start up the session.

**Note:** The session\_start() function must appear BEFORE the <html> tag:

<?php session\_start(); ?>  
  
<html>  
<body>  
  
</body>  
</html>

The code above will register the user's session with the server, allow you to start saving user information, and assign a UID for that user's session.

## Storing a Session Variable

The correct way to store and retrieve session variables is to use the PHP $\_SESSION variable:

<?php  
session\_start();  
// store session data  
$\_SESSION['views']=1;  
?>  
  
<html>  
<body>  
  
<?php  
//retrieve session data  
echo "Pageviews=". $\_SESSION['views'];  
?>  
  
</body>  
</html>

Output:

Pageviews=1

In the example below, we create a simple page-views counter. The isset() function checks if the "views" variable has already been set. If "views" has been set, we can increment our counter. If "views" doesn't exist, we create a "views" variable, and set it to 1:

<?php  
session\_start();  
  
if(isset($\_SESSION['views']))  
$\_SESSION['views']=$\_SESSION['views']+1;  
else  
$\_SESSION['views']=1;  
echo "Views=". $\_SESSION['views'];  
?>

## Destroying a Session

If you wish to delete some session data, you can use the unset() or the session\_destroy() function.

The unset() function is used to free the specified session variable:

<?php  
unset($\_SESSION['views']);  
?>

You can also completely destroy the session by calling the session\_destroy() function:

<?php  
session\_destroy();  
?>

**Note:** session\_destroy() will reset your session and you will lose all your stored session data.

**Assignment 6:** Write a user login page and maintain his previous visit and login details in cookies, provide “remember me” check box to allow user to save his login credentials in cookies.

# PHP Sending E-mails:

PHP allows you to send e-mails directly from a script.

## The PHP mail() Function

The PHP mail() function is used to send emails from inside a script.

**Syntax**

mail(to,subject,message,headers,parameters)

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| to | Required. Specifies the receiver / receivers of the email |
| subject | Required. Specifies the subject of the email. **Note:** This parameter cannot contain any newline characters |
| message | Required. Defines the message to be sent. Each line should be separated with a LF (\n). Lines should not exceed 70 characters |
| headers | Optional. Specifies additional headers, like From, Cc, and Bcc. The additional headers should be separated with a CRLF (\r\n) |
| parameters | Optional. Specifies an additional parameter to the sendmail program |

## PHP Simple E-Mail

The simplest way to send an email with PHP is to send a text email.

In the example below we first declare the variables ($to, $subject, $message, $from, $headers), then we use the variables in the mail() function to send an e-mail:

<?php  
$to = "someone@example.com";  
$subject = "Test mail";  
$message = "Hello! This is a simple email message.";  
$from = "someonelse@example.com";  
$headers = "From:" . $from;  
mail($to,$subject,$message,$headers);  
echo "Mail Sent.";  
?>

## PHP Mail Form

With PHP, you can create a feedback-form on your website. The example below sends a text message to a specified e-mail address:

<html>  
<body>  
  
<?php  
if (isset($\_REQUEST['email']))  
//if "email" is filled out, send email  
  {  
  //send email  
  $email = $\_REQUEST['email'] ;  
  $subject = $\_REQUEST['subject'] ;  
  $message = $\_REQUEST['message'] ;  
  mail("someone@example.com", "$subject",  
  $message, "From:" . $email);  
  echo "Thank you for using our mail form";  
  }  
else  
//if "email" is not filled out, display the form  
  {  
  echo "<form method='post' action='mailform.php'>  
  Email: <input name='email' type='text' /><br />  
  Subject: <input name='subject' type='text' /><br />  
  Message:<br />  
  <textarea name='message' rows='15' cols='40'>  
  </textarea><br />  
  <input type='submit' />  
  </form>";  
  }  
?>  
  
</body>  
</html>

This is how the example above works:

* First, check if the email input field is filled out
* If it is not set (like when the page is first visited); output the HTML form
* If it is set (after the form is filled out); send the email from the form
* When submit is pressed after the form is filled out, the page reloads, sees that the email input is set, and sends the email

**Assignment 7:** Write a php script to send an invitation mail to user by using HTML template and macro place holders.

# PHP Error Handling

The default error handling in PHP is very simple. An error message with filename, line number and a message describing the error is sent to the browser.

When creating scripts and web applications, error handling is an important part. If your code lacks error checking code, your program may look very unprofessional and you may be open to security risks.

This tutorial contains some of the most common error checking methods in PHP.

We will show different error handling methods:

* Simple "die()" statements
* Custom errors and error triggers
* Error reporting

## Basic Error Handling: Using the die() function

The first example shows a simple script that opens a text file:

<?php  
$file=fopen("welcome.txt","r");  
?>

If the file does not exist you might get an error like this:

**Warning**: fopen(welcome.txt) [function.fopen]: failed to open stream:  
No such file or directory in **C:\webfolder\test.php** on line **2**

To avoid that the user gets an error message like the one above, we test if the file exist before we try to access it:

<?php  
if(!file\_exists("welcome.txt"))  
  {  
  die("File not found");  
  }  
else  
  {  
  $file=fopen("welcome.txt","r");  
  }  
?>

Now if the file does not exist you get an error like this:

File not found

The code above is more efficient than the earlier code, because it uses a simple error handling mechanism to stop the script after the error.

However, simply stopping the script is not always the right way to go. Let's take a look at alternative PHP functions for handling errors.

## Creating a Custom Error Handler

Creating a custom error handler is quite simple. We simply create a special function that can be called when an error occurs in PHP.

This function must be able to handle a minimum of two parameters (error level and error message) but can accept up to five parameters (optionally: file, line-number, and the error context):

**Syntax**

error\_function(error\_level,error\_message,  
error\_file,error\_line,error\_context)

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| error\_level | Required. Specifies the error report level for the user-defined error. Must be a value number. See table below for possible error report levels |
| error\_message | Required. Specifies the error message for the user-defined error |
| error\_file | Optional. Specifies the filename in which the error occurred |
| error\_line | Optional. Specifies the line number in which the error occurred |
| error\_context | Optional. Specifies an array containing every variable, and their values, in use when the error occurred |

## Error Report levels

These error report levels are the different types of error the user-defined error handler can be used for:

|  |  |  |
| --- | --- | --- |
| **Value** | **Constant** | **Description** |
| 2 | E\_WARNING | Non-fatal run-time errors. Execution of the script is not halted |
| 8 | E\_NOTICE | Run-time notices. The script found something that might be an error, but could also happen when running a script normally |
| 256 | E\_USER\_ERROR | Fatal user-generated error. This is like an E\_ERROR set by the programmer using the PHP function trigger\_error() |
| 512 | E\_USER\_WARNING | Non-fatal user-generated warning. This is like an E\_WARNING set by the programmer using the PHP function trigger\_error() |
| 1024 | E\_USER\_NOTICE | User-generated notice. This is like an E\_NOTICE set by the programmer using the PHP function trigger\_error() |
| 4096 | E\_RECOVERABLE\_ERROR | Catchable fatal error. This is like an E\_ERROR but can be caught by a user defined handle (see also set\_error\_handler()) |
| 8191 | E\_ALL | All errors and warnings, except level E\_STRICT (E\_STRICT will be part of E\_ALL as of PHP 6.0) |

Now let’s create a function to handle errors:

function customError($errno, $errstr)  
  {  
  echo "<b>Error:</b> [$errno] $errstr<br />";  
  echo "Ending Script";  
  die();  
  }

The code above is a simple error handling function. When it is triggered, it gets the error level and an error message. It then outputs the error level and message and terminates the script.

Now that we have created an error handling function we need to decide when it should be triggered.

## Set Error Handler

The default error handler for PHP is the built in error handler. We are going to make the function above the default error handler for the duration of the script.

It is possible to change the error handler to apply for only some errors, that way the script can handle different errors in different ways. However, in this example we are going to use our custom error handler for all errors:

set\_error\_handler("customError");

Since we want our custom function to handle all errors, the set\_error\_handler() only needed one parameter, a second parameter could be added to specify an error level.

**Example**

Testing the error handler by trying to output variable that does not exist:

<?php  
//error handler function  
function customError($errno, $errstr)  
  {  
  echo "<b>Error:</b> [$errno] $errstr";  
  }  
  
//set error handler  
set\_error\_handler("customError");  
  
//trigger error  
echo($test);  
?>

The output of the code above should be something like this:

**Error:** [8] Undefined variable: test

## Trigger an Error

In a script where users can input data it is useful to trigger errors when an illegal input occurs. In PHP, this is done by the trigger\_error() function.

**Example**

In this example an error occurs if the "test" variable is bigger than "1":

<?php  
$test=2;  
if ($test>1)  
{  
trigger\_error("Value must be 1 or below");  
}  
?>

The output of the code above should be something like this:

**Notice**: Value must be 1 or below  
in **C:\webfolder\test.php** on line **6**

An error can be triggered anywhere you wish in a script, and by adding a second parameter, you can specify what error level is triggered.

Possible error types:

* E\_USER\_ERROR - Fatal user-generated run-time error. Errors that can not be recovered from. Execution of the script is halted
* E\_USER\_WARNING - Non-fatal user-generated run-time warning. Execution of the script is not halted
* E\_USER\_NOTICE - Default. User-generated run-time notice. The script found something that might be an error, but could also happen when running a script normally

**Example**

In this example an E\_USER\_WARNING occurs if the "test" variable is bigger than "1". If an E\_USER\_WARNING occurs we will use our custom error handler and end the script:

<?php  
//error handler function  
function customError($errno, $errstr)  
  {  
  echo "<b>Error:</b> [$errno] $errstr<br />";  
  echo "Ending Script";  
  die();  
  }  
  
//set error handler  
set\_error\_handler("customError",E\_USER\_WARNING);  
  
//trigger error  
$test=2;  
if ($test>1)  
  {  
  trigger\_error("Value must be 1 or below",E\_USER\_WARNING);  
  }  
?>

The output of the code above should be something like this:

**Error:** [512] Value must be 1 or below  
Ending Script

## Error Logging

By default, PHP sends an error log to the servers logging system or a file, depending on how the error\_log configuration is set in the php.ini file. By using the error\_log() function you can send error logs to a specified file or a remote destination.

Sending errors messages to yourself by e-mail can be a good way of getting notified of specific errors.

## Send an Error Message by E-Mail

In the example below we will send an e-mail with an error message and end the script, if a specific error occurs:

<?php  
//error handler function  
function customError($errno, $errstr)  
  {  
  echo "<b>Error:</b> [$errno] $errstr<br />";  
  echo "Webmaster has been notified";  
  error\_log("Error: [$errno] $errstr",1,  
  "someone@example.com","From: webmaster@example.com");  
  }  
  
//set error handler  
set\_error\_handler("customError",E\_USER\_WARNING);  
  
//trigger error  
$test=2;  
if ($test>1)  
  {  
  trigger\_error("Value must be 1 or below",E\_USER\_WARNING);  
  }  
?>

The output of the code above should be something like this:

**Error:** [512] Value must be 1 or below  
Webmaster has been notified

And the mail received from the code above looks like this:

Error: [512] Value must be 1 or below

This should not be used with all errors. Regular errors should be logged on the server using the default PHP logging system.

# MySQL PHP Database Tutorial

## Getting Started

Getting started with a database means first figuring out what you want to do. What is the nature of your data? How you want to use the data? What are the relationships between the different data elements? These questions and others are used for database design.

Our project in this tutorial is to build a database which manages links to web sites. I want to be able to add, edit and remove links in the database as well as display links from the database. I want the links to be categorized so I can display groups of links, instead of all links at once.

I want to record the following data for each link in the database:

* Name of site
* URL of site
* Description of site
* Link Category

Each of these will become one column in a table in the database. If you are not familiar with databases, a table can be thought of somewhat as a spreadsheet in Excel. The columns in a spreadsheet relate to the columns in a database. Each row in the spreadsheet would be considered a record (or row) in the database. There can also be multiple spreadsheets in a single Excel file, as there can be multiple tables in one database.

This application could use two tables, one table to store all of the categories and another table to store all the links, with a link back to the categories table. Because this is a beginner tutorial, I keep it simple by using just one table to store the info.

## Database design:

The design process consists of the following steps

1. **Determine the purpose of your database** - This helps prepare you for the remaining steps.
2. **Find and organize the information required** - Gather all of the types of information you might want to record in the database, such as product name and order number.
3. **Divide the information into tables** - Divide your information items into major entities or subjects, such as Products or Orders. Each subject then becomes a table.
4. **Turn information items into columns** - Decide what information you want to store in each table. Each item becomes a field, and is displayed as a column in the table. For example, an Employees table might include fields such as Last Name and Hire Date.
5. **Specify primary keys** - Choose each table’s primary key. The primary key is a column that is used to uniquely identify each row. An example might be Product ID or Order ID.
6. **Set up the table relationships** - Look at each table and decide how the data in one table is related to the data in other tables. Add fields to tables or create new tables to clarify the relationships, as necessary.
7. **Refine your design** - Analyze your design for errors. Create the tables and add a few records of sample data. See if you can get the results you want from your tables. Make adjustments to the design, as needed.
8. **Apply the normalization rules** - Apply the data normalization rules to see if your tables are structured correctly. Make adjustments to the tables

In a majority of cases, a person who is doing the design of a database is a person with expertise in the area of database design, rather than expertise in the domain from which the data to be stored is drawn e.g. financial information, biological information etc. Therefore the data to be stored in the database must be determined in cooperation with a person who does have expertise in that domain, and who is aware of what data must be stored within the system.

This process is one which is generally considered part of [requirements analysis](http://en.wikipedia.org/wiki/Requirements_analysis), and requires skill on the part of the database designer to elicit the needed information from those with the domain knowledge. This is because those with the necessary domain knowledge frequently cannot express clearly what their system requirements for the database are as they are unaccustomed to thinking in terms of the discrete data elements which must be stored. Data to be stored can be determined by Requirement Specification.

## Database Normalization:

In the field of [relational database](http://en.wikipedia.org/wiki/Relational_database) design, normalization is a systematic way of ensuring that a database structure is suitable for general-purpose querying and free of certain undesirable characteristics—insertion, update, and deletion anomalies—that could lead to a loss of [data integrity](http://en.wikipedia.org/wiki/Data_integrity).

A standard piece of database design guidance is that the designer should create a fully normalized design; selective [denormalization](http://en.wikipedia.org/wiki/Denormalization) can subsequently be performed, but only for [performance](http://en.wikipedia.org/wiki/Computer_performance) reasons. However, some modeling disciplines, such as the [dimensional modeling](http://en.wikipedia.org/wiki/Dimensional_modeling) approach to [data warehouse](http://en.wikipedia.org/wiki/Data_warehouse) design, explicitly recommend non-normalized designs, i.e. designs that in large part do not adhere to 3NF. Normalization consists of normal forms that are 1NF,2NF,3NF,BOYCE-CODD NF,5NF

Data integrity is data that has a complete or whole structure. All characteristics of the data including [business rules](http://en.wikipedia.org/wiki/Business_rule), rules for how pieces of data relate, dates, definitions and lineage must be correct for data to be complete.

Per the discipline of data architecture, when functions are performed on the data the functions must ensure integrity. Examples of functions are transforming the data, storing the history, storing the definitions (Metadata) and storing the lineage of the data as it moves from one place to another. The most important aspect of data integrity per the data architecture discipline is to expose the data, the functions and the data's characteristics.

Data that has integrity is identically maintained during any operation (such as transfer, storage or retrieval). Put simply in business terms, data integrity is the assurance that data is consistent, certified and can be reconciled.

In terms of a [database](http://en.wikipedia.org/wiki/Database) data integrity refers to the process of ensuring that a database remains an accurate reflection of the universe of discourse it is modelling or representing. In other words there is a close correspondence between the facts stored in the database and the real world it models.

## Create Database

Now that we know what data we want and what we want to with the data, we need to create the database to store the data. The first thing to do is create an empty database to hold our table. For this tutorial, I name the database **linksdb**. In MySQL you use the following command from the command line to create the new database.

mysqladmin -p create linksdb

The "-p" flag is used so it will prompt you for your password to MySQL. Most setups require a username and password for security reasons. Set up the username and password using the mysqladmin tool.

Test connecting to the database using the mysql client. On the command line:

mysql -p linksdb

You should get a mysql> prompt if it connects to the database. Else, it returns an error saying Unknown database 'linksdb'

## Creating the Database Table

Next, we need to create the table in the database. Tables are created using SQL statements, and can be created using the MySQL client tool or PHP. The nice thing about using a PHP script to create the table is you can save the script to use later. If something goes wrong, you can re-create the tables. Or, you can use it to refer to the database schema.

The SQL command to create a table is:

CREATE TABLE tablename (   
column1 column1type,   
column2 column2type,   
etc.... )

The table we want to create is:

Table Name: links

Columns: id (integer - primary link key)

sitename (50 characters)

siteurl (75 characters)

description (text field - lots of text)

category (50 characters)

The SQL code to create the categories table is:

CREATE TABLE links (   
id INT NOT NULL AUTO\_INCREMENT,   
sitename VARCHAR(50),   
siteurl VARCHAR(75),   
description TEXT,   
category VARCHAR(50),   
PRIMARY KEY(id) );

The **id** column is the primary key for this table. In order to be a primary key, the column can not be null (NOT NULL). I also set it to automatically increment the number (AUTO\_INCREMENT) so when each record is added, the id will increase by one. The last line specifies that the id column will be the primary key for this table. A primary key is a unique number for that specific record or row of data.

The **sitename**, **siteurl** and **category** column each are specified as a VARCHAR or variable character field each with their respective maximum length. A VARCHAR field holds characters from 0 up to its maximum specified length. Most databases VARCHAR limit is 255 characters. If you have a field which may require more characters you should use a TEXT field, which is what is used for the **description** field. A text field can hold lots of data, usually dependent on the database.

The most common data types are INT, VARCHAR, TEXT and DATETIME.

## Executing SQL in PHP

The code to execute a SQL statement in PHP has 3 steps:

1. Connect to database
2. Create statement
3. Execute statement

**1**.The command to connect to the database is:

$cid = mysql\_connect($host,$usr,$pwd);

Where $host, $usr, and $pwd are previously assigned. Host ($host) refers to the machine running MySQL. Username ($usr) and password ($pwd) connect to that MySQL machine. This command returns a connection id, which is used to identify this connection in later queries.

**2**. Creating the SQL statement simply consists of assigning the SQL statement such as the one above to a string. I usually have it run over multiple lines, concatenating the string together as I go. This makes it easier to read.

**3**. The command to actually send the database the SQL command is:

$result = mysql\_db\_query($db,"$SQL",$cid);

Where $db is the database to query, $SQL is the SQL statement, and $cid is the connection id created above. This returns a 1 if executed correctly, and undefined or false if an error occurred.

Putting all of this together, and adding a little error detection and displaying of the errors gives us the following script, which you can download and load on to your web server running PHP. Note: Opening the file through the web server will execute the script and create the database tables.

## Insert Data

Next, we need to get some data into the database. To do so, we use an HTML form to collect the data, and then insert the data into the database using PHP. Both these functions can be done on the same page.

The form can submit to the same page it is on. This page checks if the POST method is used. If so, insert into the database. I find it easier to have the form and database insert operation on the same page. Makes one less file to keep track of.

Here is the HTML for the form:

<form name="fa" action="insert\_link.php" method="post">  
<b>Category: </b> <input type="text" name="category" size=40>   
<b>Site Name:</b> <input type="text" name="sitename" size=40>  
<b>Site URL: </b> <input type="text" name="siteurl" value="http://" size=40>  
<b>Description: </b> <textarea name="description" rows=5 cols=40></textarea>  
<p><input type="submit" value="Add Link"></p>  
</form>

The form looks like the one below. This tutorial does **not** use an actual database, so this form is not "hooked up" to anything. It is to show you what we are working with. Also note that the above HTML snippet has the HTML formatting table elements removed because the the snippet is for display purposes only. The actual, underlying HTML used to display the form below uses an HTML table.



This form gathers the data we want to insert into the database and submits back to the same page.

Now we need the PHP code to process this information. The first block of code sets up the connection to the database.

<?php   
  $usr = "---username---";  
  $pwd = "---password---";  
  $db = "linksdb";  
  $host = "localhost";  
  
  $cid = mysql\_connect($host,$usr,$pwd);  
  
  if (!$cid) { echo("ERROR: " . mysql\_error() . "\n"); }  
?>

We need to create our SQL statement for the insert. The SQL format for an INSERT is:

INSERT INTO -tablename-   
(column\_name1, column\_name2, ...) VALUES   
(data1, data2, ... )

Where the first column name matches to the first data in each of the sets. Remember, PHP variables are automatically initialized with the form field names submitted to the page. So the PHP code to submit the query and check for errors is:

<?php   
  
if ($\_SERVER['REQUEST\_METHOD'] == "POST") {   
  
// the following 4 lines are needed if your server has register\_globals set to Off  
$category = $\_POST['category'];  
$sitename = $\_POST['sitename'];  
$siteurl = $\_POST['siteurl'];  
$description = $\_POST['description'];  
  
$SQL = " INSERT INTO links ";  
$SQL = $SQL . " (category, sitename, siteurl, description) VALUES ";  
$SQL = $SQL . " ('$category', '$sitename','$siteurl','$description') ";  
$result = mysql\_db\_query($db,"$SQL",$cid);  
  
if (!$result) {   
    echo("ERROR: " . mysql\_error() . "\n$SQL\n"); }  
  
echo ("New Link Added\n");  
  
}   
mysql\_close($cid);   
?>

## View Data from Database

Now that we have inserted data into the database, we want to retrieve the data and display it as links on the page. The SQL statement that gets data from a database is SELECT. The format is:

SELECT (columns) FROM (table)  
WHERE (exclusive criteria)

For our database, we want to select all columns. For ease, we use a '\*' instead of listing out each column. We also want to select only a specific category of links. Let's say we want the category "Local Docs"   
  
Our SELECT SQL statement would be:

SELECT \* FROM links   
WHERE category = 'Local Docs'

Single quotes specify a string value in the WHERE clause. If we were using a column which was a number value, no quotes would be needed. The WHERE part is optional. If you want to select everything from the database, you can leave off the WHERE portion. Review throughout this SQL Tutorial for more examples of what you can do with SELECT statements and WHERE clauses.

The PHP code to display links from the database starts with the usual code to initialize the database connection.

<?php  
  $usr = "--username--";  
  $pwd = "--password--";  
  $db = "linksdb";  
  $host = "localhost";  
  
  $cid = mysql\_connect($host,$usr,$pwd);  
  if (!$cid) { echo("ERROR: " . mysql\_error() . "\n"); }  
?>

The next part of the script sets up and executes the SQL statement. This will look familiar from the previous pages of this database tutorial.

I set the category as a variable at the beginning, so the code can be copied and pasted for other category selects, and only one change is needed.

<?php  
  $category = "Local Docs";  
  $SQL = " SELECT \* FROM links ";  
  $SQL = $SQL . " WHERE category = '$category' ";  
  $retid = mysql\_db\_query($db, $SQL, $cid);  
  if (!$retid) { echo( mysql\_error()); }

The SELECT statement returns rows of data from the database. We need to loop through that data and display the information we want, which will be our links.

The command to grab a row is:

$row = mysql\_fetch\_array($retid);

This sets $row as an array holding one record from the database, with the column names as the "keys" for the array.

To retrieve the siteurl value from that array you would use:

$siteurl = $row["siteurl"];

When the mysql\_fetch\_array command is called next, it moves to the next data record returned by the SELECT. If there are no more rows of data the command returns false. To loop through all rows of data, we can use a while statement with the mysql\_fetch\_array in it.

Here is the code snippet to loop through the data and display it to the screen:

while ($row = mysql\_fetch\_array($retid)) {   
$siteurl = $row["siteurl"];   
$sitename = $row["sitename"];   
  
echo ("<dd><a href='$siteurl'>$sitename</a></dd>\n");   
}   
echo ("</dt></p>");   
  
?>

Remember that PHP writes out HTML to the page which the browser then renders.

## Manage Data

Now that we have a database populated with data, we need to be able to work with that data. We have collected numerous links, but the links will inevitably get old and need to be updated. In this section of the tutorial, we update and delete data from the database.

The two SQL statements that handle these actions are helpfully named UPDATE and DELETE.

The structure of the UPDATE statment is:

UPDATE (table) SET  
(column1) = (value),  
(column2) = (value),  
...  
WHERE (column) = (value)

The last line (ends with = value) does not have a comma after it.

It is very important to use the WHERE clause to specify which data record you want to update. If the where clause is left out, it will update all records with the specified values. This is where the primary key id becomes useful. The unique primary key is used to specify which record you want to update.

The structure of the DELETE statement is:

DELETE FROM (table)   
WHERE (column) = (value)

This will delete the complete record(s). Multiple rows can be deleted at once depending on the WHERE clause. Again, the WHERE clause is crucial. If it is left off, it will delete all records from the database.

Now, we look at how to edit or delete a link. First, use a SELECT statment to display the available links. The code is the same as before:

<?php  
  $category = "Local Docs";  
  $SQL = " SELECT \* FROM links ";  
  $SQL = $SQL . " WHERE category = '$category' ";  
  $retid = mysql\_db\_query($db, $SQL, $cid);  
  
  if (!$retid) { echo( mysql\_error()); }  
?>

One difference from earlier in this tutorial is displaying the links. We do not need the links "live." Instead we want to be able to edit them by clicking either "edit" or "delete." We will need the primary key id of the link we wish to edit or delete. The primary key can be passed in via the URL and the get method by adding on ?id=(number) at the end of the URL.

The delete code is shown below, I will leave the edit page as an exercise for you to do. The edit page is only slightly more complex because you need another form to display the data to edit. This can be processed the same way as the insert script. What I usually do is copy over the insert form and modify that. A working form gives me a good base for new form, in this case, the edit page.

The PHP code snippet to display the links on screen:

echo ("<p><table><tr><td colspan=3><b>$category</b></td></tr>\n");  
while ($row = mysql\_fetch\_array($retid)) {  
$sitename = $row["sitename"];  
$id = $row["id"];  
  
echo ("<tr>");  
echo ("<td>$sitename</td>\n");  
echo ("<td><a href=\"manageedit.php?id=$id\">Edit</a></td>");  
echo ("<td><a href=\"manage.php?id=$id&task=del\">Delete</a></td>");  
echo ("</tr>");  
}   
echo ("</table>");

When the "Edit" link is clicked it will go to the "manageedit.php" page. It will also pass along the id of the link to edit. This page will display the data in a form. When that form is submitted it will update the information.

When the "Delete" link is clicked it will submit to the same page (manage.php) and pass along two variables. The first variable is the id of the record to delete, the second variable is "task" which is set to "del". This is done so we can catch that variable and know we are performing a delete action when we load the page.

The following code, which is placed in the top of the script, shows how:

if ($task=="del") {  
  
$SQL = " DELETE FROM links ";  
$SQL = $SQL . " WHERE id = $id ";  
mysql\_db\_query($db, $SQL, $cid);  
  
}

**Assignment 8:**

Write a php project for:

1. User registration page.
2. User login page.
3. Index page with header, footer and left menu.
4. Once user get logged in, header will get updated for logout link and hide register link.
5. Display copyright note with current year in footer.
6. Allow user to upload his profile image after successful registration.
7. Allow logged in user to edit/update his profile.
8. Display users list in a separate page, provide a link to add user in friend list and can see profile page of any user from user list.

# URL Rewrite in PHP

Well, let’s get started. You’ll need:

* everyone’s favourite Apache Webserver installed (v1.2 or later)
* optionally, your favourite CGI scripts configured for Apache. Yes, I’ve said optionally, since what we’re going to do will happen right inside Apache and not PHP, or Perl, etc.
* since (nearly) everything in Apache is controlled through its configuration files (httpd.conf, .htaccess, etc.), being familiar with these files might help you. You’ll also need to have write access to this file, and access to restart the Apache. I’d strongly recommend you do everything on a private testserver first, rather than on your own, or your company’s, production server!

Most of you will have read and/or heard about mod\_rewrite — yes, it’s an Apache module, and it’s even installed by default! Go and check your modules directory (note that under \*nix operating systems there’s a chance that your Apache was compiled with missing mod\_rewrite, in which case, consult your sysadmin).

We’re going use this tiny module to achieve everything mentioned above. To use this module, first we have to enable it, since it’s initially disabled in the configuration file. Open the httpd.conf file and uncomment the following lines (remove the trailing #s):

#LoadModule rewrite\_module modules/mod\_rewrite.so

#AddModule mod\_rewrite.c

The first line tells Apache to load the mod\_rewrite module, while the second one enables the use of it. After you restart Apache, mod\_rewrite should be enabled, but not yet running.

**What is the mod\_rewrite Solution, Exactly?**

But what does it exactly do? Hey! Here comes the whole point of this article!

mod\_rewrite catches URLs that meet specific conditions, and rewrites them as it was told to.

For example, you can have a non-existing

http://www.mysite.com/anything

URL that is rewritten to:

http://www.mysite.com/deep/stuff/very\_complicated\_url?text=

having\_lots\_of\_extra\_characters

//

Did you expect something more? Be patient…

<IfModule mod\_rewrite.c>

RewriteEngine on

RewriteRule ^/shortcut$ /complicated/and/way/too/long/url/here

</IfModule>

Of course this, too, should go into the httpd.conf file again, (you can even put it into a virtualhost context).

After you restart Apache (you’ll get used to it soon!) you can type this into your browser:

http://localhost/shortcut

If there’s a directory structure /complicated/and/way/too/long/url/here existing in your document root, you’re going to be “redirected” there, where you’ll see the contents of this directory (eg, the directory listing, index.html, whatever there is).

To understand mod\_rewrite better, it’s important to know that this is not true redirection. “Classic” redirection is done with the Location: header of the HTTP protocol, and tells the browser itself to go to another URL. There are numerous ways to do this, for example, in PHP you could write:

<?php

// this PHP file is located at http://localhost/shortcut/index.php

header

("Location: /complicated/and/way/too/long/url/here");

?>

This code shows the same page by sending a HTTP header back to the browser. That header tells the browser to move to another URL location instantly. But, what mod\_rewrite does is totally different: it ‘tricks’ the browser, and serves the page as if it were really there – that’s why this is an URL rewriter and not a simple redirector (you can even verify the HTTP headers sent and received to understand the difference).

But it’s not just shortening paths that makes mod\_rewrite the “Swiss Army Knife of URL manipulation”…

## Rules

You’ve just seen how to specify a really simple RewriteRule. Now let’s take a closer look…

RewriteRule Pattern Substitution [Flag(s)]

RewriteRule is a simple instruction that tells mod\_rewrite what to do. The magic is that you can use regular expressions in the Pattern and references in the Substitution strings. What do you think of the following rule?

RewriteRule /products/([0-9]+) /siteengine/products.php?id=$1

Now you can use the following syntax in your URLs:

http://localhost/products/123

After restarting Apache, you’ll find this is translated as:

http://localhost/siteengine/products.php?id=123

If you use only ‘fancy’ URLs in your scripts, there will be no way for your visitor to find out where your script resides (/siteengine in the example), what its name is (products.php), or what the name of the parameter to pass (productid) is! Do you like it? We’ve just completed two of our tasks, look!

* **Search-engine compatibility:** there are no fancy characters in the URL, so the engines will explore your whole site
* **Security:** ScriptKiddie(tm)-modified URLs will cause no error, as they’re verified with the regular expression first to be a number – URLs with no proper syntax can’t even reach the script itself.

Of course, you can create more complex RewriteRules. For example, here’s a set of rules I’m using on a site:

 RewriteRule ^/products$ /content.php

 RewriteRule ^/products/([0-9]+)$ /content.php?id=$1

 RewriteRule ^/products/([0-9]+),([ad]\*),([0-9]{0,3}),([0-9]\*),([0-9]\*$)

   /marso/content.php?id=$1&sort=$2&order=$3&start=$4

Thanks to these rules I can use the followings links in the application:

* Show an opening page that contains product categories:http://somesite.hu/products
* Product listing, categoryid is 123, page 1 (as default), default order:http://somesite.hu/products/123 http://somesite.hu/products/123,,,,
* Product listing, categoryid is 123, page 2, descending order by third field (d for descending, 3 for third field):http://somesite.hu/products/123,d,3,2

This is also an example of the use of multiple RewriteRules. When there’s a RegExp match, the proper substitution occurs, mod\_rewrite stops running and Apache serves the page with the substituted URL. Should there be no match (after processing all the rules), a usual 404 page comes up. And of course you can also define one or more rules (eg. ^.\*$ as last pattern) to specify which script(s) to run depending on the mistaken URL.

The third, optional part of RewriteRule is:

RewriteRule Pattern Substitution Flag(s)

With flags, you can send specific headers to the browser when the URL matches the pattern, such as:

* ‘forbidden‘ or ‘f‘ for 403 forbidden,
* ‘gone‘ or ‘g‘ for 410 gone,
* you may also force redirection, or force a MIME-type.

You can even use the:

* ‘nocase‘ or ‘NC‘ flag to make the pattern case-insensitive
* ‘next‘/N‘ to loop back to the first rule (‘next round‘ — though this may result in an endless loop, be careful with it!)
* ‘skip=N‘/'S=N‘ to skip the following N rules

…and so on.

I hope you feel like I felt while playing around with this module for the first time!

## Conditions

But that’s not all! Though RewriteRule gives you an opportunity to have professional URL rewriting, you can make it more customized using conditions.

The format of the conditions is simple:

RewriteCond Something\_to\_test Condition

Any RewriteCond condition affects the behaviour of the following RewriteRule, which is a little confusing, as RewriteCond won’t be evaluated until the following RewriteRule pattern matches the current URL.

It works like this: mod\_rewrite takes all the RewriteRules and starts matching the current URL against each RewriteRule pattern. If there’s a RewriteRule pattern that matches the URL, mod\_rewrite checks if there are existing conditions for this RewriteRule, and if the first one returns true. If it does, the proper substitution will occur, but if not, mod\_rewrite looks for remaining conditions. When there are no more conditions, the subsequent RewriteRule is checked.

This way you can customize URL rewriting using conditions based on practically everything that’s known during a HTTP transfer in Apache — and a lot more! Basically you can use all of these variables in the Something\_to\_test string:

* HTTP header variables:  
  HTTP\_USER\_AGENT, HTTP\_REFERER, HTTP\_COOKIE, HTTP\_FORWARDED, HTTP\_HOST, HTTP\_PROXY\_CONNECTION, HTTP\_ACCEPT
* Connection & request variables:REMOTE\_ADDR, REMOTE\_HOST, REMOTE\_USER, REMOTE\_IDENT, REQUEST\_METHOD, SCRIPT\_FILENAME, PATH\_INFO, QUERY\_STRING, AUTH\_TYPE
* Server internal variables:DOCUMENT\_ROOT, SERVER\_ADMIN, SERVER\_NAME, SERVER\_ADDR, SERVER\_PORT, SERVER\_PROTOCOL, SERVER\_SOFTWARE
* System variables:TIME\_YEAR, TIME\_MON, TIME\_DAY, TIME\_HOUR, TIME\_MIN, TIME\_SEC, TIME\_WDAY, TIME
* mod\_rewrite special values:API\_VERSION, THE\_REQUEST, REQUEST\_URI, REQUEST\_FILENAME, IS\_SUBREQ

The condition can be a simple string or a standard regular expression, with additions like:

* <, >, = simple comparison operators
* -f if Something\_to\_test is a file
* -d if Something\_to\_test is a directory

As you can see, these are more than enough to specify a condition like this one (taken from the mod\_rewrite manual):

RewriteCond %{HTTP\_USER\_AGENT} ^Mozilla.\*

RewriteRule ^/$ /homepage.max.html [L]

RewriteCond %{HTTP\_USER\_AGENT} ^Lynx.\*

RewriteRule ^/$ /homepage.min.html [L]

RewriteRule ^/$ /homepage.std.html [L]

When a browser requests the index page, 3 things can happen:

* browser with a Mozilla engine the browser will be served homepage.max.html
* using Lynx (character-based browser) the homepage.min.html will open
* if the browser’s name doesn’t contain ‘Mozilla’ nor ‘Lynx’, the standard homepage.std.html file will be sent

You can even disable users from accessing images from outside your server:

RewriteCond %{HTTP\_REFERER} !^$

RewriteCond %{HTTP\_REFERER} !^http://localhost/.\*$ [OR,NC]

RewriteCond %{HTTP\_REFERER} !^http://mysite.com/.\*$ [OR,NC]

RewriteCond %{HTTP\_REFERER} !^http://www.mysite.com/.\*$ [OR,NC]

RewriteRule .\*\.(gif|GIF|jpg|JPG)$ http://mysite/images/bad.gif [L,R]

But of course, there are endless possibilities, including IP- or time-dependant conditions, etc.

## For Advanced Users

I mentioned user-friendliness in the introduction, and haven’t dealt with it. First, let’s imagine we’re having a huge download site that has the downloadable software separated into categories, each with a unique id (which is used in the SQL SELECTs). We use links like open.php?categoryid=23487678 to display the contents of a category.

To ensure that our URLs were easily memorized (eg. http://www.downloadsite.com/Nettools/Messengers) we could use:

RewriteRule ^/NetTools$ /test.php?target=3

RewriteRule ^/NetTools/Messengers$ /test.php?target=34

assuming the ID is 3 for the NetTools category and 34 for Messengers subcategory.

But our site is huge, as I’ve mentioned – who wants to hunt down all the IDs from the database, and then edit the config file by hand? No-one! Instead, we can use the mapping feature of mod\_rewrite. Map allows us to provide a replacement-table – stored in a single text file — within a hash file (for fast lookups), or even served through an external program!

For better performance I’d generate a single text file using PHP, which contains the following:

NetTools 3

NetTools/Messengers 34

.

.

.

and so on.

The httpd.conf file would contain:

RewriteMap categories txt:/path/to/file/categoryids.txt

RewriteRule ^(.\*)$ open.php?categoryid=${categories:$1|0}

These lines tell mod\_rewrite to read the categoryids.txt file upon Apache startup, and provide the ID for the URL for open.php. The |0 means that categoryid will be 0 if there’s no matching key in the textfile.

You can also choose to serve the IDs on-the-fly via a script or other executable code. The program is started by Apache on server startup, and runs until shutdown. The program must have buffered I/O disabled, read from the stdin, and write results to stdout — it’s that simple!

With RewriteMap you can do a lot more, including:

* load balancing through servers (using rnd:),
* creation of a Webcluster that has an homogenous URL layout,
* redirection to mirror sites without modifying your Web application,
* denial of user access based on a hostlist,

and so on.

## Tips, Tricks and Advice

1. Before using mod\_rewrite in a production server, I’d recommend setting up a testserver (or playground, whatever you prefer to call it).

 During development, you must avoid using ‘old-fashioned’ URLs in your application.

 There might still be need to verify data passed through the URL (passing non-existing — too large or small – IDs, for example, might be risky).

 Writing ‘intelligent’ RewriteRules saved me coding time and helped me write simpler code. I’m using error\_reporting(E\_ALL); everywhere (and I recommend it!), but I find it boring to do the following for the ten thousandths time:

if (isset($\_GET['id']) && (validNumber($\_GET['id']))

if (isset($\_GET['todo']) && ($\_GET['todo']=='deleteitem'))

The following trick helped me to get rid of the extra isset() expression by providing all the needed parameters each time in the RewriteRules:

RewriteRule ^/products/[0-9]+$ products.php?id=$1&todo=

I know, I know it’s not the answer to the meaning of life — but it’s hard to show how nice and clear a solution this might provide in such a short example.

**Finally…**

That’s all for our ‘brief’ overview of mod\_rewrite. After you’ve mastered the basics, you’ll find you can easily create your own rules. If you like the idea of URL rewriting, may want to play with mod\_rewrite – some ideas follow (note that the underlying PHP code is not important in this case):

http://www.mysite.com/1/2/3/content.html => 1\_2\_3\_content.html http://www.mysite.com/1/2/3/content.html => content.php ? category=1

http://www.mysite.com/1/2/3/ => content.php ? category=1 & subcat1 = 2 & subcat2 = 3

http://www.mysite.com/1/2/3/details => content.php ? category=1 & subcat1 = 2 & subcat2 = 3

http://www.mysite.com/bookshop/browse/bytitle => library.php ? target=listbooks & order = title

http://www.mysite.com/bookshop/browse/byauthor => library.php ? target=listbooks & order = author

http://www.mysite.com/bookshop/product/123 => library.php ? target=showproduct & itemid=123

http://www.mysite.com/bookshop/helpdesk/2 => library.php ? target=showhelp & page=2

http://www.mysite.com/bookshop/registration => library.php ? target=reg

**Assignment 9:**

Update profile page from assignment 8 so that profile page will display with user name.

Ex: /amol

Will display profile page of user having name=amol.

# cURL

curl is a tool to transfer data from or to a server, using one of the supported protocols (DICT, FILE, FTP, FTPS, GOPHER, HTTP, HTTPS, IMAP, IMAPS, LDAP, LDAPS, POP3, POP3S, RTMP, RTSP, SCP, SFTP, SMTP, SMTPS, TELNET and TFTP). The command is designed to work without user interaction.

curl offers a busload of useful tricks like proxy support, user authentication, FTP upload, HTTP post, SSL connections, cookies, file transfer resume and more. As you will see below, the number of features will make your head spin!

curl is powered by libcurl for all transfer-related features.

cURL is a library which allows you to connect and communicate to many different types of servers with many different types of protocols. Using cURL you can:

* Implement payment gateways’ payment notification scripts.
* Download and upload files from remote servers.
* Login to other websites and access members only sections.

PHP cURL library is definitely the odd man out. Unlike other PHP libraries where a whole plethora of functions is made available, PHP cURL wraps up a major parts of its functionality in just four functions.

A typical PHP cURL usage follows the following sequence of steps.

**curl\_init** – Initializes the session and returns a cURL handle which can be passed to other cURL functions.

**curl\_opt** – This is the main work horse of cURL library. This function is called multiple times and specifies what we want the cURL library to do.

**curl\_exec** – Executes a cURL session.

**curl\_close** – Closes the current cURL session.

**Ex1.** Below are some examples which should make the working of cURL clearer.

<?php

/\*\*

\* Initialize the cURL session

\*/

$ch = curl\_init();

/\*\*

\* Set the URL of the page or file to download.

\*/

curl\_setopt($ch, CURLOPT\_URL, ‘http://news.google.com/news?hl=en&topic=t&output=rss’);

/\*\*

\* Ask cURL to return the contents in a variable instead of simply echoing them to the browser.

\*/

curl\_setopt($ch, CURLOPT\_RETURNTRANSFER, 1);

/\*\*

\* Execute the cURL session

\*/

$contents = curl\_exec ($ch);

/\*\*

\* Close cURL session

\*/

curl\_close ($ch);

?>

As you can see, **curl\_setopt** is the pivot around which the main cURL functionality revolves. cURL functioning is controlled by way of passing predefined options and values to this function.

The above code uses two such options.

* **CURLOPT\_URL**: Use it to specify the URL which you want to process. This could be the URL of the file you want to download or it could be the URL of the script to which you want to post some data.
* **CURLOPT\_RETURNTRANSFER**: Setting this option to 1 will cause the curl\_exec function to return the contents instead of echoing them to the browser.

**Ex2. Download file or web page using PHP cURL and save it to file**

The below PHP code is a slight variation of the above code. It not only downloads the contents of the specified URL but also saves it to a file.

<?php

/\*\*

\* Initialize the cURL session

\*/

$ch = curl\_init();

/\*\*

\* Set the URL of the page or file to download.

\*/

curl\_setopt($ch, CURLOPT\_URL, ‘http://news.google.com/news?hl=en&topic=t&output=rss’);

/\*\*

\* Create a new file

\*/

$fp = fopen(‘rss.xml’, ‘w’);

/\*\*

\* Ask cURL to write the contents to a file

\*/

curl\_setopt($ch, CURLOPT\_FILE, $fp);

/\*\*

\* Execute the cURL session

\*/

curl\_exec ($ch);

/\*\*

\* Close cURL session and file

\*/

curl\_close ($ch);

fclose($fp);

?>

Here we have used another of the cURL options, CURLOPT\_FILE. Obtain a file handler by creating a new file or opening an existing one and then pass this file handler to the curl\_set\_opt function.

cURL will now write the contents to a file as it downloads a web page or file.

**Assignment 10:**

Write a php page to check login credentials from a separate page= checkAuth.php.

checkAuth.php will response a formatted xml with user stastus like

1 = valid user

2 = user not found

3 = user has blocked by admin.

# Using SOAP with PHP:

SOAP is a lightweight protocol for exchange of information in a decentralized, distributed environment. It is an XML based protocol that consists of three parts: an envelope that defines a framework for describing what is in a message and how to process it, a set of encoding rules for expressing instances of application-defined data types, and a convention for representing remote procedure calls and responses. SOAP can potentially be used in combination with a variety of other protocols; however, the only bindings defined in this document describe how to use SOAP in combination with HTTP and HTTP Extension Framework.

SOAP consists of three parts:

* The **SOAP envelope** construct defines an overall framework for expressing **what** is in a message; **who** should deal with it, and **whether** it is optional or mandatory.
* The **SOAP encoding** rules defines a serialization mechanism that can be used to exchange instances of application-defined data types.
* The **SOAP RPC representation** defines a convention that can be used to represent remote procedure calls and responses.

Although these parts are described together as part of SOAP, they are functionally orthogonal. In particular, the envelope and the encoding rules are defined in different namespaces in order to promote simplicity through modularity.

In addition to the SOAP envelope, the SOAP encoding rules and the SOAP RPC conventions, this specification defines two protocol bindings that describe how a SOAP message can be carried in HTTP [[5]](http://www.w3.org/TR/soap11/#RFC2616) messages either with or without the HTTP Extension Framework

## SOAP Envelope

A SOAP message is an XML document that consists of a mandatory SOAP envelope, an optional SOAP header, and a mandatory SOAP body. This XML document is referred to as a SOAP message for the rest of this specification. The namespace identifier for the elements and attributes defined in this section is "<http://schemas.xmlsoap.org/soap/envelope/>". A SOAP message contains the following:

* The Envelope is the top element of the XML document representing the message.
* The Header is a generic mechanism for adding features to a SOAP message in a decentralized manner without prior agreement between the communicating parties. SOAP defines a few attributes that can be used to indicate who should deal with a feature and whether it is optional or mandatory.
* The Body is a container for mandatory information intended for the ultimate recipient of the. SOAP defines one element for the body, which is the Fault element used for reporting errors.

The grammar rules are as follows:

1. Envelope
   * The element name is "Envelope".
   * The element MUST be present in a SOAP message
   * The element MAY contain namespace declarations as well as additional attributes. If present, such additional attributes MUST be namespace-qualified. Similarly, the element MAY contain additional sub elements. If present these elements MUST be namespace-qualified and MUST follow the SOAP Body element.
2. Header
   * The element name is "Header".
   * The element MAY be present in a SOAP message. If present, the element MUST be the first immediate child element of a SOAP Envelope element.
   * The element MAY contain a set of header entries each being an immediate child element of the SOAP Header element. All immediate child elements of the SOAP Header element MUST be namespace-qualified.
3. Body
   * The element name is "Body".
   * The element MUST be present in a SOAP message and MUST be an immediate child element of a SOAP Envelope element. It MUST directly follow the SOAP Header element if present. Otherwise it MUST be the first immediate child element of the SOAP Envelope element.
   * The element MAY contain a set of body entries each being an immediate child element of the SOAP Body element. Immediate child elements of the SOAP Body element MAY be namespace-qualified. SOAP defines the SOAP Fault element, which is used to indicate error messages.

## SOAP Encoding

The SOAP encoding style is based on a simple type system that is a generalization of the common features found in type systems in programming languages, databases and semi-structured data. A type either is a simple (scalar) type or is a compound type constructed as a composite of several parts, each with a type. This is described in more detail below. This section defines rules for serialization of a graph of typed objects. It operates on two levels. First, given a schema in any notation consistent with the type system described, a schema for an XML grammar may be constructed. Second, given a type-system schema and a particular graph of values conforming to that schema, an XML instance may be constructed. In reverse, given an XML instance produced in accordance with these rules, and given also the original schema, a copy of the original value graph may be constructed.

The namespace identifier for the elements and attributes defined in this section is "<http://schemas.xmlsoap.org/soap/encoding/>". The encoding samples shown assume all namespace declarations are at a higher element level.

Use of the data model and encoding style described in this section is encouraged but not required; other data models and encodings can be used in conjunction with.

## Rules for Encoding Types in XML

XML allows very flexible encoding of data. SOAP defines a narrower set of rules for encoding. This section defines the encoding rules at a high level, and the next section describes the encoding rules for specific types when they require more detail. The encodings described in this section can be used in conjunction with the mapping of RPC calls and responses.

To describe encoding, the following terminology is used:

1. A "value" is a string, the name of a measurement (number, date, enumeration, etc.) or a composite of several such primitive values. All values are of specific types.
2. A "simple value" is one without named parts. Examples of simple values are particular strings, integers, enumerated values etc.
3. A "compound value" is an aggregate of relations to other values. Examples of Compound Values are particular purchase orders, stock reports, street addresses, etc.
4. Within a compound value, each related value is potentially distinguished by a role name, ordinal or both. This is called its "accessor." Examples of compound values include particular Purchase Orders, Stock Reports etc. Arrays are also compound values. It is possible to have compound values with several accessors each named the same, as for example, RDF does.
5. An "array" is a compound value in which ordinal position serves as the only distinction among member values.
6. A "struct" is a compound value in which accessor name is the only distinction among member values, and no accessor has the same name as any other.
7. A "simple type" is a class of simple values. Examples of simple types are the classes called "string," "integer," enumeration classes, etc.
8. A "compound type" is a class of compound values. An example of a compound type is the class of purchase order values sharing the same accessors (shipTo, totalCost, etc.) though with potentially different values (and perhaps further constrained by limits on certain values).
9. Within a compound type, if an accessor has a name that is distinct within that type but is not distinct with respect to other types, that is, the name plus the type together are needed to make a unique identification; the name is called "locally scoped." If however the name is based in part on a Uniform Resource Identifier, directly or indirectly, such that the name alone is sufficient to uniquely identify the accessor irrespective of the type within which it appears, the name is called "universally scoped."
10. Given the information in the schema relative to which a graph of values is serialized, it is possible to determine that some values can only be related by a single instance of an accessor. For others, it is not possible to make this determination. If only one accessor can reference it, a value is considered "single-reference". If referenced by more than one, actually or potentially, it is "multi-reference." Note that it is possible for a certain value to be considered "single-reference" relative to one schema and "multi-reference" relative to another.
11. Syntactically, an element may be "independent" or "embedded." An independent element is any element appearing at the top level of a serialization. All others are embedded elements.

Although it is possible to use the xsi:type attribute such that a graph of values is self-describing both in its structure and the types of its values, the serialization rules permit that the types of values MAY be determinate only by reference to a schema. Such schemas MAY be in the notation described by "XML Schema Part 1: Structures" and "XML Schema Part 2: Datatypes" or MAY be in any other notation. Note also that, while the serialization rules apply to compound types other than arrays and structs, many schemas will contain only struct and array types.

The rules for serialization are as follows:

1. All values are represented as element content. A multi-reference value MUST be represented as the content of an independent element. A single-reference value SHOULD not be (but MAY be).
2. For each element containing a value, the type of the value MUST be represented by at least one of the following conditions: (a) the containing element instance contains an xsi:type attribute, (b) the containing element instance is itself contained within an element containing a (possibly defaulted) SOAP-ENC:arrayType attribute or (c) or the name of the element bears a definite relation to the type, that type then determinable from a schema.
3. A simple value is represented as character data, that is, without any subelements. Every simple value must have a type that is either listed in the XML Schemas Specification, or whose source type is listed therein.
4. A Compound Value is encoded as a sequence of elements, each accessor represented by an embedded element whose name corresponds to the name of the accessor. Accessors whose names are local to their containing types have unqualified element names; all others have qualified names.
5. A multi-reference simple or compound value is encoded as an independent element containing a local, unqualified attribute named "id" and of type "ID" per the XML Specification. Each accessor to this value is an empty element having a local, unqualified attribute named "href" and of type "uri-reference" per the XML Schema Specification, with a "href" attribute value of a URI fragment identifier referencing the corresponding independent element.
6. Strings and byte arrays are represented as multi-reference simple types, but special rules allow them to be represented efficiently for common cases. An accessor to a string or byte-array value MAY have an attribute named "id" and of type "ID" per the XML Specification. If so, all other accessors to the same value are encoded as empty elements having a local, unqualified attribute named "href" and of type "uri-reference" per the XML Schema Specification, with a "href" attribute value of a URI fragment identifier referencing the single element containing the value.
7. It is permissible to encode several references to a value as though these were references to several distinct values, but only when from context it is known that the meaning of the XML instance is unaltered.
8. Arrays are compound values. SOAP arrays are defined as having a type of "SOAP-ENC:Array" or a type derived there from.  
     
   SOAP arrays have one or more dimensions (rank) whose members are distinguished by ordinal position. An array value is represented as a series of elements reflecting the array, with members appearing in ascending ordinal sequence. For multi-dimensional arrays the dimension on the right side varies most rapidly. Each member element is named as an independent element.  
     
   SOAP arrays can be single-reference or multi-reference values, and consequently may be represented as the content of either an embedded or independent element.  
     
   SOAP arrays MUST contain a "SOAP-ENC:arrayType" attribute whose value specifies the type of the contained elements as well as the dimension(s) of the array. The value of the "SOAP-ENC:arrayType" attribute is defined as follows:

arrayTypeValue = atype asize  
atype          = QName \*( rank )  
rank           = "[" \*( "," ) "]"  
asize          = "[" #length "]"  
length         = 1\*DIGIT

The "atype" construct is the type name of the contained elements expressed as a QName as would appear in the "type" attribute of an XML Schema element declaration and acts as a type constraint (meaning that all values of contained elements are asserted to conform to the indicated type; that is, the type cited in SOAP-ENC:arrayType must be the type or a supertype of every array member). In the case of arrays of arrays or "jagged arrays", the type component is encoded as the "innermost" type name followed by a rank construct for each level of nested arrays starting from 1. Multi-dimensional arrays are encoded using a comma for each dimension starting from 1.  
  
The "asize" construct contains a comma separated list of zero, one, or more integers indicating the lengths of each dimension of the array. A value of zero integers indicates that no particular quantity is asserted but that the size may be determined by inspection of the actual members.  
  
For example, an array with 5 members of type array of integers would have an arrayTypeValue value of "int[][5]" of which the atype value is "int[]" and the asize value is "[5]". Likewise, an array with 3 members of type two-dimensional arrays of integers would have an arrayTypeValue value of "int[,][3]" of which the atype value is "int[,]" and the asize value is "[3]".  
  
A SOAP array member MAY contain a "SOAP-ENC:offset" attribute indicating the offset position of that item in the enclosing array. This can be used to indicate the offset position of a partially represented array. Likewise, an array member MAY contain a "SOAP-ENC:position" attribute indicating the position of that item in the enclosing array. This can be used to describe members of sparse arrays. The value of the "SOAP-ENC:offset" and the "SOAP-ENC:position" attribute is defined as follows:

arrayPoint = "[" #length "]"  
with offsets and positions based at 0.

1. A NULL value or a default value MAY be represented by omission of the accessor element. A NULL value MAY also be indicated by an accessor element containing the attribute xsi:null with value '1' or possibly other application-dependent attributes and values.

## Using SOAP for RPC

One of the design goals of SOAP is to encapsulate and exchange RPC calls using the extensibility and flexibility of XML. This section defines a uniform representation of remote procedure calls and responses.

Although it is anticipated that this representation is likely to be used in combination with the encoding style defined in section 5 other representations are possible. The SOAP encodingStyle attribute can be used to indicate the encoding style of the method call and or the response using the representation described in this section.

Using SOAP for RPC is orthogonal to the SOAP protocol binding. In the case of using HTTP as the protocol binding, an RPC call maps naturally to an HTTP request and an RPC response maps to an HTTP response. However, using SOAP for RPC is not limited to the HTTP protocol binding.

To make a method call, the following information is needed:

* The URI of the target object
* A method name
* An optional method signature
* The parameters to the method
* Optional header data

SOAP relies on the protocol binding to provide a mechanism for carrying the URI. For example, for HTTP the request URI indicates the resource that the invocation is being made against. Other than it be a valid URI, SOAP places no restriction on the form of an address.

## RPC and SOAP Body

RPC method calls and responses are both carried in the SOAP Body element using the following representation:

* A method invocation is modelled as a struct.
* The method invocation is viewed as a single struct containing an accessor for each [in] or [in/out] parameter. The struct is both named and typed identically to the method name.
* Each [in] or [in/out] parameter is viewed as an accessor, with a name corresponding to the name of the parameter and type corresponding to the type of the parameter. These appear in the same order as in the method signature.
* A method response is modelled as a struct.
* The method response is viewed as a single struct containing an accessor for the return value and each [out] or [in/out] parameter. The first accessor is the return value followed by the parameters in the same order as in the method signature.
* Each parameter accessor has a name corresponding to the name of the parameter and type corresponding to the type of the parameter. The name of the return value accessor is not significant. Likewise, the name of the struct is not significant. However, a convention is to name it after the method name with the string "Response" appended.
* A method fault is encoded using the SOAP Fault element. If a protocol binding adds additional rules for fault expression, those also MUST be followed.

As noted above, method and response structs can be encoded according to the rules in section 5, or other encodings can be specified using the encodingStyle attribute.

Applications MAY process requests with missing parameters but also MAY return a fault.

Because a result indicates success and a fault indicates failure, it is an error for the method response to contain both a result and a fault.

## RPC and SOAP Header

Additional information relevant to the encoding of a method request but not part of the formal method signature MAY be expressed in the RPC encoding. If so, it MUST be expressed as a subelement of the SOAP Header element.

An example of the use of the header element is the passing of a transaction ID along with a message. Since the transaction ID is not part of the signature and is typically held in an infrastructure component rather than application code, there is no direct way to pass the necessary information with the call. By adding an entry to the headers and giving it a fixed name, the transaction manager on the receiving side can extract the transaction ID and use it without affecting the coding of remote procedure calls.

To show you an example of how PHP SOAP Extension can be used in a SOAP client application, here is my first SOAP PHP program, soapClient.php  
  
**Example : soapClient.php**

<?php   
$client = new SoapClient("http://[insert real path here]/productOrder.wsdl");  
print\_r($client->serverFunction('UserName', password', $Integer1, $string1));  
?>

This client function will communicate with productOrder.wsdl file on the server. WSDL (Web Services Definition Language) is an XML based standard designed to describe protocol bindings and message formats of Web services. WSDL is often pronounced as "Whiz-Dull.  
  
Being XML-based, this allows clients to automatically discover everything about the functionality of the web service. Human-readable documentation is technically not required for a SOAP service that uses a WSDL document, though it is still highly recommended. Let's take a look at the structure of a WSDL document and how we can use it to figure out what is available to us in a SOAP-based web service. Out of all three specifications that we're going to look at in relationship to SOAP, WSDL is the most ethereal. Both supporters and detractors often call writing WSDL documents a black art.  
  
The only argument against using it is that the client has to load the relevant WSDL document from the server before the RPC can be made, and this can take a significant amount of time in a Web environment. In order to speed things up, PHP's ext/soap uses a WSDL caching feature that can be controlled through setting the soap.wsdl\_cache\_enabled, soap.wsdl\_cache\_dir and soap.wsdl\_cache\_ttl configuration directives, either in your php.ini or by using ini\_set()(see SOAP Server Example ). By default, WSDL caching is turned on and caches WSDL files for one day.  
Beginning with a root definitions element, WSDL documents follow this basic structure:

<definitions>  
<types>  
…  
</types>  
<message>  
…  
</message>  
<portType>  
…  
</portType>  
<binding>  
…  
</binding>  
</definitions>

As you can see, in addition to the definitions element, there are four main sections to a WSDL document: types, message, portType, and binding.  
  
Here is a copy of the WSDL document for the demonstration Web service used in in the SOAP Server.

**Example : productOrder.wsdl**

<?xml version ='1.0' encoding ='UTF-8' ?>  
<definitions name=SOAPDefn  
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='getOrderRequest'>  
<part name='userID' type='xsd:string'/> //passing variable 1  
<part name='paswrd' type='xsd:string'/> //passing variable 2  
<part name='orderNo' type='xsd:integer'/> //passing variable 3  
<part name='itemName' type='xsd:string'/> //passing variable 4  
  
</message>  
<message name='getOrderResponse'>  
<part name='Result' type='xsd:integer'/>  
</message>  
  
<portType name='orderPortType'>  
<operation name='saveOrderDetails'>  
<input message='tns:getOrderRequest'/>  
<output message='tns:getOrderResponse'/>  
</operation>  
</portType>  
  
<binding name='orderBinding' type='tns:orderPortType'>  
<soap:binding style='rpc'  
transport='http://schemas.xmlsoap.org/soap/http'/>  
<operation name='saveOrderDetails'>  
<soap:operation soapAction='urn:xmethods-delayed-quotes#saveOrderDetails'/>  
<input>  
<soap:body use='encoded' namespace='urn:xmethods-delayed-quotes'  
encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'/>  
</input>  
<output>  
<soap:body use='encoded' namespace='urn:xmethods-delayed-quotes'  
encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'/>  
</output>  
</operation>  
</binding>  
  
<service name='orderService'>  
<port name='StockQuotePort' binding='tns:orderBinding'>  
<soap:address location= ‘http://[insert real path here]/ server.php'/>  
</port>  
</service>  
</definitions>

Note: The WSDL caching feature is on by default. During the development of your WSDL file it should be turned off.  
  
Now it's time to create our server.  
  
**Example : server.php**

<?php  
require\_once "SOAPFunctionsManager.obj";  
ini\_set("soap.wsdl\_cache\_enabled", "0"); //disabling WSDL cache  
$server = new SoapServer("productOrder.wsdl");  
$server->setClass("SOAPFunctionsManager");  
$server->handle();  
?>

$server->setClass method connect the SoapServer object with the SOAPFunctionsManager class, and used all methods in the SOAPFunctionsManager class with the SOAP Server .  
  
**Example: SOAPFunctionsManager**

class SOAPFunctionsManager

{

function saveOrderDetails($userName, $paswrd, $orderNo, $itemname)

{

if($this->authenticateUser($userName, $paswrd))

{

$order\_Index = $this->saveOrder($orderNo, $itemname);

return $order\_Index;

} else {

return 0;

}

}

function saveOrder($orderNo, $itemname)

{

execute MY SQL Query here..

return last Insert\_ID()

}

function authenticateUser($userName, $paswrd)

{

if($userName == '\_USER\_NAME\_' && $paswrd == '\_PASSWORD\_)

return 1;

else

return 0;

}

}

Conclusion   
  
In this article, we were introduced to SOAP, the most complex of the web service protocols so far. SOAP relies heavily on other standards like WSDL and XSD. Well XML-RPC is a great way to transmit simple data as shown in the above sample. However, SOAP allows Objects to be used. These days most people are working in object oriented languages and know the power of using Objects. SOAP allows you to harness that power in your web applications.  
  
The SOAP extension is fully documented at http://www.php.net/manual/en/ref.soap.php.  
If you read the SOAP Extension reference page, you will see that SOAP Extension support SOAP client applications with a class called SoapClient, which offers the following functions:  
  
• SoapClient->\_\_construct() - constructs a new SoapClient object  
• SoapClient->\_\_soapCall() - Calls a SOAP function  
• SoapClient->\_\_getFunctions() - Returns list of SOAP functions  
• SoapClient->\_\_getLastRequestHeaders() - Returns last SOAP request headers  
• SoapClient->\_\_getLastRequest() - Returns last SOAP request  
• SoapClient->\_\_getLastResponseHeaders() - Returns last SOAP response headers  
• SoapClient->\_\_getLastResponse() - Returns last SOAP response

**Assignment 11:**

Write a SOAP application to provide help tips for asked topic.

Build a database with formatted help points or tips on related questions.

A SOAP call will serve related help content on asked questions or topic.