PHP Tutorial

PHP is a powerful server-side scripting language for creating dynamic and interactive websites.

PHP is the widely-used, free, and efficient alternative to competitors such as Microsoft's ASP. PHP is perfectly suited for Web development and can be embedded directly into the HTML code.

The PHP syntax is very similar to Perl and C. PHP is often used together with Apache (web server) on various operating systems. It also supports ISAPI and can be used with Microsoft's IIS on Windows.

Introduction to PHP

A PHP file may contain text, HTML tags and scripts. Scripts in a PHP file are executed on the server.

What You Should Already Know

Before you continue you should have a basic understanding of the following:

- HTML / XHTML
- Some scripting knowledge

If you want to study these subjects first, find the tutorials on our Home page.

What is PHP?

- PHP stands for **P**HP: **H**ypertext **P**reprocessor
- PHP is a server-side scripting language, like ASP
- PHP scripts are executed on the server
- PHP supports many databases (MySQL, Informix, Oracle, Sybase, Solid, PostgreSQL, Generic ODBC, etc.)
- PHP is an open source software (OSS)
- PHP is free to download and use

What is a PHP File?

- PHP files may contain text, HTML tags and scripts
- PHP files are returned to the browser as plain HTML
- PHP files have a file extension of ".php", ".php3", or ".phtml"

What is MySQL?

- MySQL is a database server
- MySQL is ideal for both small and large applications
- MySQL supports standard SQL
- MySQL compiles on a number of platforms
- MySQL is free to download and use

PHP + MySQL

- PHP combined with MySQL are cross-platform (means that you can develop in Windows and serve on a Unix platform)

Why PHP?

- PHP runs on different platforms (Windows, Linux, Unix, etc.)
- PHP is compatible with almost all servers used today (Apache, IIS, etc.)
- PHP is FREE to download from the official PHP resource: www.php.net
- PHP is easy to learn and runs efficiently on the server side

Where to Start?

- Install an Apache server on a Windows or Linux machine
- Install PHP on a Windows or Linux machine
- Install MySQL on a Windows or Linux machine

PHP Installation

What do You Need?

This tutorial will not explain how to install PHP, MySQL, or Apache Server.

If your server supports PHP - you don't need to do anything! You do not need to compile anything or install any extra tools - just create some .php files in your web directory - and the server will parse them for you. Most web hosts offer PHP support.

However, if your server does not support PHP, you must install PHP. Below is a link to a good tutorial from PHP.net on how to install PHP5:

http://www.php.net/manual/en/install.php

Download PHP

Download PHP for free here: http://www.php.net/downloads.php

Download MySQL Database

Download MySQL for free here: http://www.mysql.com/downloads/index.html

Download Apache Server

Download Apache for free here: http://httpd.apache.org/download.cgi

PHP Syntax

You cannot view the PHP source code by selecting "View source" in the browser - you will only see the output from the PHP file, which is plain HTML. This is because the scripts are executed on the server before the result is sent back to the browser.

Basic PHP Syntax

A PHP scripting block always starts with **<?php** and ends with **?>**. A PHP scripting block can be placed anywhere in the document.

On servers with shorthand support enabled you can start a scripting block with <? and end with ?>.

However, for maximum compatibility, we recommend that you use the standard form (<?php) rather than the shorthand form.

```
<?php
?>
```

A PHP file normally contains HTML tags, just like an HTML file, and some PHP scripting code.

Below, we have an example of a simple PHP script which sends the text "Hello World" to the browser:

```
<html>
<body>
<?php
echo "Hello World";
?>
</body>
</html>
```

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.

There are two basic statements to output text with PHP: **echo** and **print**. In the example above we have used the echo statement to output the text "Hello World".

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>
```

PHP Variables

Variables are used for storing values, such as numbers, strings or function results, so that they can be used many times in a script.

Variables in PHP

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

When a variable is set it can be used over and over again in your script

All variables in PHP start with a $ sign symbol.

The correct way of setting a variable in PHP:

```
$var_name = value;
```

New PHP programmers often forget the $ sign at the beginning of the variable. In that case it will not work.

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

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

PHP is a Loosely Typed Language

In PHP a variable does not need to be declared before being set.

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 how they are set.

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.

Variable Naming Rules

- A variable name must start with a letter or an underscore "_"
- A variable name can only contain alpha-numeric characters and underscores (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 underscore ($my_string), or with capitalization ($myString)

PHP String

A string variable is used to store and manipulate a piece of text.

Strings in PHP

String variables are used for a value that contains character strings.

In this tutorial we are going to look at some of 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 string "Hello World" to a string variable called $txt:

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

The output of the code above will be:

```
Hello World
```

Now, lets try to use some different functions and operators to manipulate our string.

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 variables together, use the dot (.) operator:

```
<?php
$txt1="Hello World";
$txt2="1234";
echo $txt1 . " " . $txt2;
?>
```

The output of the code above will be:

```
Hello World 1234
```

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.

Between the two string variables we added a string with a single character, an empty space, to separate the two variables.

Using the strlen() function

The strlen() function is used to find the length of a string.

Let's find the length of our string "Hello world!":

```
<?php
echo strlen("Hello world!");
?>
```

The output of the code above will be:

```
12
```

The length of a string is often used in loops or other functions, when it is important to know when the string ends. (i.e. in a loop, we would want to stop the loop after the last character in the string)

Using the strpos() function

The strpos() function is used to search for a string or character within a string.

If a match is found in the string, this function will return the position of the first match. If no match is found, it will return FALSE.

Let's see if we can find the string "world" in our string:

```
<?php
echo strpos("Hello world!","world");
?>
```

The output of the code above will be:

```
6
```

As you see the position of the string "world" in our string is position 6. The reason that it is 6, and not 7, is that the first position in the string is 0, and not 1.

Complete PHP String Reference

For a complete reference of all string functions, go to our complete PHP String Reference.

The reference contains a brief description and examples of use for each function

PHP String Functions

PHP String Introduction

The string functions allow you to manipulate strings.

Installation

The string functions are part of the PHP core. There is no installation needed to use these functions.

PHP String Functions

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

Function	Description	PHP
addcslashes()	Returns a string with backslashes in front of the specified characters	4
addslashes()	Returns a string with backslashes in front of predefined characters	3
bin2hex()	Converts a string of ASCII characters to hexadecimal values	3
chop()	Alias of rtrim()	3
chr()	Returns a character from a specified ASCII value	3
chunk_split()	Splits a string into a series of smaller parts	3
convert_cyr_string()	Converts a string from one Cyrillic character-set to another	3
convert_uudecode()	Decodes a uuencoded string	5
convert_uuencode()	Encodes a string using the uuencode algorithm	5
count_chars()	Returns how many times an ASCII character occurs within a string and returns the information	4
crc32()	Calculates a 32-bit CRC for a string	4
crypt()	One-way string encryption (hashing)	3
echo()	Outputs strings	3
explode()	Breaks a string into an array	3
fprintf()	Writes a formatted string to a specified output stream	5
get_html_translation_table()	Returns the translation table used by htmlspecialchars() and htmlentities()	4
hebrev()	Converts Hebrew text to visual text	3
hebrevc()	Converts Hebrew text to visual text and new lines (\n) into <br />	3
html_entity_decode()	Converts HTML entities to characters	4
htmlentities()	Converts characters to HTML entities	3
htmlspecialchars_decode()	Converts some predefined HTML entities to characters	5
htmlspecialchars()	Converts some predefined characters to HTML entities	3
implode()	Returns a string from the elements of an array	3
join()	Alias of implode()	3
levenshtein()	Returns the Levenshtein distance between two strings	3
localeconv()	Returns locale numeric and monetary formatting information	4
ltrim()	Strips whitespace from the left side of a string	3
md5()	Calculates the MD5 hash of a string	3
md5_file()	Calculates the MD5 hash of a file	4
metaphone()	Calculates the metaphone key of a string	4
money_format()	Returns a string formatted as a currency string	4
nl_langinfo()	Returns specific local information	4
nl2br()	Inserts HTML line breaks in front of each newline in a string	3
number_format()	Formats a number with grouped thousands	3
ord()	Returns the ASCII value of the first character of a string	3
parse_str()	Parses a query string into variables	3
print()	Outputs a string	3
printf()	Outputs a formatted string	3
quoted_printable_decode()	Decodes a quoted-printable string	3
quotemeta()	Quotes meta characters	3
rtrim()	Strips whitespace from the right side of a string	3
setlocale()	Sets locale information	3
sha1()	Calculates the SHA-1 hash of a string	4
sha1_file()	Calculates the SHA-1 hash of a file	4
similar_text()	Calculates the similarity between two strings	3
soundex()	Calculates the soundex key of a string	3
sprintf()	Writes a formatted string to a variable	3
sscanf()	Parses input from a string according to a format	4
str_ireplace()	Replaces some characters in a string (case-insensitive)	5

str_pad()	Pads a string to a new length	4
str_repeat()	Repeats a string a specified number of times	4
str_replace()	Replaces some characters in a string (case-sensitive)	3
str_rot13()	Performs the ROT13 encoding on a string	4
str_shuffle()	Randomly shuffles all characters in a string	4
str_split()	Splits a string into an array	5
str_word_count()	Count the number of words in a string	4
strcasecmp()	Compares two strings (case-insensitive)	3
strchr()	Finds the first occurrence of a string inside another string (alias of strstr())	3
strcmp()	Compares two strings (case-sensitive)	3
strcoll()	Locale based string comparison	4
strcspn()	Returns the number of characters found in a string before any part of some specified characters are found	3
strip_tags()	Strips HTML and PHP tags from a string	3
stripcslashes()	Unquotes a string quoted with addcslashes()	4
stripslashes()	Unquotes a string quoted with addslashes()	3
stripos()	Returns the position of the first occurrence of a string inside another string (case-insensitive)	5
stristr()	Finds the first occurrence of a string inside another string (case-insensitive)	3
strlen()	Returns the length of a string	3
strnatcasecmp()	Compares two strings using a "natural order" algorithm (case-insensitive)	4
strnatcmp()	Compares two strings using a "natural order" algorithm (case-sensitive)	4
strncasecmp()	String comparison of the first n characters (case-insensitive)	4
strncmp()	String comparison of the first n characters (case-sensitive)	4
strpbrk()	Searches a string for any of a set of characters	5
strpos()	Returns the position of the first occurrence of a string inside another string (case-sensitive)	3
strrchr()	Finds the last occurrence of a string inside another string	3
strrev()	Reverses a string	3
strripos()	Finds the position of the last occurrence of a string inside another string (case-insensitive)	5
strrpos()	Finds the position of the last occurrence of a string inside another string (case-sensitive)	3
strspn()	Returns the number of characters found in a string that contains only characters from a specified charlist	3
strstr()	Finds the first occurrence of a string inside another string (case-sensitive)	3
strtok()	Splits a string into smaller strings	3
strtolower()	Converts a string to lowercase letters	3
strtoupper()	Converts a string to uppercase letters	3
strtr()	Translates certain characters in a string	3
substr()	Returns a part of a string	3
substr_compare()	Compares two strings from a specified start position (binary safe and optionally case-sensitive)	5
substr_count()	Counts the number of times a substring occurs in a string	4
substr_replace()	Replaces a part of a string with another string	4
trim()	Strips whitespace from both sides of a string	3
ucfirst()	Converts the first character of a string to uppercase	3
ucwords()	Converts the first character of each word in a string to uppercase	3
vfprintf()	Writes a formatted string to a specified output stream	5
vprintf()	Outputs a formatted string	4

vsprintf()	Writes a formatted string to a variable	4
wordwrap()	Wraps a string to a given number of characters	4

PHP String Constants

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

Constant	**Description**	**PHP**
CRYPT_SALT_LENGTH	Contains the length of the default encryption method for the system. For standard DES encryption, the length is 2	
CRYPT_STD_DES	Set to 1 if the standard DES-based encryption with a 2 character salt is supported, 0 otherwise	
CRYPT_EXT_DES	Set to 1 if the extended DES-based encryption with a 9 character salt is supported, 0 otherwise	
CRYPT_MD5	Set to 1 if the MD5 encryption with a 12 character salt starting with $1$ is supported, 0 otherwise	
CRYPT_BLOWFISH	Set to 1 if the Blowfish encryption with a 16 character salt starting with $2$ or $2a$ is supported, 0 otherwise0	
HTML_SPECIALCHARS		
HTML_ENTITIES		
ENT_COMPAT		
ENT_QUOTES		
ENT_NOQUOTES		
CHAR_MAX		
LC_CTYPE		
LC_NUMERIC		
LC_TIME		
LC_COLLATE		
LC_MONETARY		
LC_ALL		
LC_MESSAGES		
STR_PAD_LEFT		
STR_PAD_RIGHT		
STR_PAD_BOTH		

PHP Operators

Operators are used to operate on values.

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	3

		5/2	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

Comparison Operators

Operator	Description	Example
==	is equal to	5==8 returns false
!=	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 If...Else Statements

The if, elseif and else statements in PHP are used to perform different actions based on different conditions.

Conditional Statements

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.

- **if...else statement** - use this statement if you want to execute a set of code when a condition is true and another if the condition is not true
- **elseif statement** - is used with the if...else statement to execute a set of code if **one** of several condition are true

---

The If...Else Statement

If you want to execute some code if a condition is true and another code if a condition is false, use the if....else statement.

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>
```

If more than one line should be executed if a condition is true/false, the lines should be enclosed within curly braces:

```
<html>
<body>
<?php
$d=date("D");
if ($d=="Fri")
  {
  echo "Hello!<br />";
  echo "Have a nice weekend!";
  echo "See you on Monday!";
  }
?>
</body>
</html>
```

The ElseIf Statement

If you want to execute some code if one of several conditions are true use the elseif statement

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

The Switch statement in PHP is used to perform one of several different actions based on one of several different conditions.

The Switch Statement

If you want to select one of many blocks of code to be executed, use the Switch statement.

The switch statement is used to avoid long blocks of if..elseif..else code.

Syntax

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

Example

This is how it works:

- A single expression (most often a variable) is evaluated once

- The value of the expression is compared with the values for each case in the structure
- If there is a match, the code associated with that case is executed
- After a code is executed, **break** is used to stop the code from running into the next case
- The default statement is used if none of the cases are true

```
<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 can store one or more values in a single variable name.

What is an array?

When working with PHP, sooner or later, you might want to create many similar variables.

Instead of having many similar variables, you can store the data as elements in an array.

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

There are three different kind of arrays:

- **Numeric array** - An array with a numeric ID key
- **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 element with a numeric ID key.

There are different ways to create a numeric array.

Example 1

In this example the ID key is automatically assigned:

```
$names = array("Peter","Quagmire","Joe");
```

Example 2

In this example we assign the ID key manually:

```
$names[0] = "Peter";
$names[1] = "Quagmire";
$names[2] = "Joe";
```

The ID keys can be used in a script:

```
<?php
$names[0] = "Peter";
$names[1] = "Quagmire";
$names[2] = "Joe";
echo $names[1] . " and " . $names[2] .
" are ". $names[0] . "'s neighbors";
?>
```

The code above will output:

```
Quagmire and Joe are Peter's neighbors
```

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

Lets 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 Looping

Looping statements in PHP are used to execute the same block of code a specified number of times.

Looping

Very often when you write code, you want the same block of code to run a number of times. You can use looping statements in your code to perform this.

In PHP we have the following looping statements:

- **while** - loops through a block of code if and as long as a specified condition is true
- **do...while** - loops through a block of code once, and then repeats the loop as long as a special 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 Statement

The while statement will execute a block of code **if and as long as** a condition is true.

Syntax

```
while (condition)
code to be executed;
```

Example

The following example demonstrates a loop that will continue to run as long as the variable 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>
```

The do...while Statement

The do...while statement will execute a block of code **at least once** - it then will repeat the loop **as long as** a condition is true.

Syntax

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

Example

The following example will increment the value of i at least once, and it will continue incrementing the variable i as long as it has a value of less than 5:

```
<html>
<body>
<?php
$i=0;
do
  {
  $i++;
  echo "The number is " . $i . "<br />";
  }
while ($i<5);
?>
</body>
</html>
```

The for Statement

The for statement is used when you know how many times you want to execute a statement or a list of statements.

Syntax

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

Note: The for statement has three parameters. The first parameter initializes variables, the second parameter holds the condition, and the third parameter contains the increments required to implement the loop. If more than one variable is included in the initialization or the increment parameter, they should be separated by commas. The condition must evaluate to true or false.

Example

The following example prints the text "Hello World!" five times:

```
<html>
<body>
<?php
for ($i=1; $i<=5; $i++)
{
  echo "Hello World!<br />";
}
```

```
?>
</body>
</html>
```

The foreach Statement

The foreach statement is used to loop through arrays.

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

Syntax

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

Example

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

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

PHP Functions

The real power of PHP comes from its functions.

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

PHP Functions

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

For a reference and examples of the built-in functions, please visit our PHP Reference.

Create a PHP Function

A function is a block of code that can be executed whenever we need it.

Creating PHP functions:

- All functions start with the word "function()"
- Name the function - It should be possible to understand what the function does by its name. The name can start with a letter or underscore (not a number)
- Add a "{" - The function code starts after the opening curly brace
- Insert the function code
- Add a "}" - The function is finished by a closing curly brace

Example

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

```
<html>
<body>
<?php
function writeMyName()
  {
  echo "Kai Jim Refsnes";
  }
writeMyName();
?>
</body>
</html>
```

Use a PHP Function

Now we will use the function in a PHP script:

```
<html>
<body>
<?php
function writeMyName()
  {
  echo "Kai Jim Refsnes";
  }
echo "Hello world!<br />";
echo "My name is ";
writeMyName();
echo ".<br />That's right, ";
writeMyName();
echo " is my name.";
?>
</body>
</html>
```

The output of the code above will be:

```
Hello world!
My name is Kai Jim Refsnes.
That's right, Kai Jim Refsnes is my name.
```

PHP Functions - Adding parameters

Our first function (writeMyName()) is a very simple function. It only writes a static string.

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

You may have noticed the parentheses after the function name, like: writeMyName(). The parameters are specified inside the parentheses.

Example 1

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

```
<html>
<body>
<?php
function writeMyName($fname)
  {
  echo $fname . " Refsnes.<br />";
  }
echo "My name is ";
writeMyName("Kai Jim");
echo "My name is ";
writeMyName("Hege");
echo "My name is ";
writeMyName("Stale");
?>
</body>
</html>
```

The output of the code above will be:

```
My name is Kai Jim Refsnes.
My name is Hege Refsnes.
My name is Stale Refsnes.
```

Example 2

The following function has two parameters:

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

The output of the code above will be:

```
My name is Kai Jim Refsnes.
My name is Hege Refsnes!
My name is Ståle Refsnes...
```

PHP Functions - Return values

Functions can also be used to return values.

Example

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

The output of the code above will be:

```
1 + 16 = 17
```

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.

Form example:

```
<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>
```

The example HTML page above contains two input fields and a submit button. When the user fills in this form and click on the submit button, the form data is sent to the "welcome.php" file.

The "welcome.php" file looks like this:

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

A sample output of the above script may be:

```
Welcome John.
You are 28 years old.
```

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

Form Validation

User input should be validated whenever possible. Client side validation is faster, and will reduce server load.

However, any site that gets enough traffic to worry about server resources, may also need to worry about site security. You should always use server side validation if the form accesses 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

The $_GET variable is used to collect values from a form with method="get".

The $_GET Variable

The $_GET variable is an array of variable names and values sent by the HTTP GET method.

The $_GET variable is used to collect values from a form 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 it has limits on the amount of information to send (max. 100 characters).

Example

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

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

```
http://www.w3schools.com/welcome.php?name=Peter&age=37
```

The "welcome.php" file can now use the $_GET variable to catch the form data (notice that the names of the form fields will automatically be the ID keys in the $_GET array):

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

Why use $_GET?

Note: When using the $_GET variable all variable names and values are displayed in the URL. So 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 HTTP GET method is not suitable on large variable values; the value cannot exceed 100 characters.

The $_REQUEST Variable

The PHP $_REQUEST variable contains the contents of both $_GET, $_POST, and $_COOKIE.

The PHP $_REQUEST variable can be used to get the result from form data sent with both the GET and POST methods.

Example

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

PHP $_POST

The $_POST variable is used to collect values from a form with method="post".

The $_POST Variable

The $_POST variable is an array of variable names and values sent by the HTTP POST method.

The $_POST variable is used to collect values from a form 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.

Example

```
<form action="welcome.php" method="post">
Enter your name: <input type="text" name="name" />
Enter your age: <input type="text" name="age" />
<input type="submit" />
</form>
```

When the user clicks the "Submit" button, the URL will not contain any form data, and will look something like this:

```
http://www.w3schools.com/welcome.php
```

The "welcome.php" file can now use the $_POST variable to catch the form data (notice that the names of the form fields will automatically be the ID keys in the $_POST array):

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

Why use $_POST?

- Variables sent with HTTP POST are not shown in the URL
- Variables have no length limit

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

The $_REQUEST Variable

The PHP $_REQUEST variable contains the contents of both $_GET, $_POST, and $_COOKIE.

The PHP $_REQUEST variable can be used to get the result from form data sent with both the GET and POST methods.

Example

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

PHP Date()

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

The PHP Date() Function

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

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 (as a timestamp)

PHP Date - What is a Timestamp?

A timestamp is the number of seconds since January 1, 1970 at 00:00:00 GMT. This is also known as the Unix Timestamp.

PHP Date - Format the Date

The first parameter in the date() function specifies how to format the date/time. It uses letters to represent date and time formats. Here are some of the letters that can be used:

- d - The day of the month (01-31)
- m - The current month, as a number (01-12)
- Y - The current year in four digits

An overview of all the letters that can be used in the format parameter, can be found in our PHP Date reference.

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

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

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

```
2006/07/11
2006.07.11
2006-07-11
```

PHP Date - Adding a Timestamp

The second parameter in the date() function specifies a timestamp. This parameter is optional. If you do not supply a timestamp, the current time will be used.

In our next example we will use the mktime() function to create a timestamp for tomorrow.

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

Syntax

```
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 2006/07/12
```

PHP Date - Reference

For more information about all the PHP date functions, please visit our PHP Date Reference.

PHP Include File

Server Side Includes (SSI) are used to create functions, headers, footers, or elements that will be reused on multiple pages.

Server Side Includes

You can insert the content of a file into a 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. The include() function generates a warning (but the script will continue execution) while the require() function generates a fatal error (and the script execution will stop after the error).

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

This can save the developer a considerable amount of time. This means that you can create a standard header or menu file that you want all your web pages to include. When the header needs to be updated, you can only update this one 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 web pages).

The include() Function

The include() function takes all the text in a specified file and copies it into the file that uses the include function.

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, like this:

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

Example 2

Now, let's assume we have a standard menu file that should be used on all pages (include files usually have a ".php" extension). Look at the "menu.php" file below:

```
<html>
<body>
<a href="http://www.w3schools.com/default.php">Home</a> |
<a href="http://www.w3schools.com/about.php">About Us</a> |
<a href="http://www.w3schools.com/contact.php">Contact Us</a>
```

The three files, "default.php", "about.php", and "contact.php" should all include the "menu.php" file. Here is the code in "default.php":

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

If you look at the source code of the "default.php" in a browser, it will look something like this:

```
<html>
<body>
```

```
<a href="default.php">Home</a> |
<a href="about.php">About Us</a> |
<a href="contact.php">Contact Us</a>
<h1>Welcome to my home page</h1>
<p>Some text</p>
</body>
</html>
```

And, of course, we would have to do the same thing for "about.php" and "contact.php". By using include files, you simply have to update the text in the "menu.php" file if you decide to rename or change the order of the links or add another web page to the site.

The require() Function

The require() function is identical to include(), they only handle errors differently.

The include() function generates a warning (but the script will continue execution) while the require() function generates a fatal error (and the script execution will stop after the error).

If you include a file with the include() function and an error occurs, you might get an error message like the one below.

PHP code:

```
<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 still executed! This is because a Warning does not stop the script execution.

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

PHP code:

```
<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 was 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 executing if files are missing or misnamed.

PHP File Handling

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

Opening a File

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 Filesystem Reference

PHP Filesystem Functions

PHP Filesystem Introduction

The filesystem functions allow you to access and manipulate the filesystem.

Installation

The filesystem functions are part of the PHP core. There is no installation needed to use these functions.

Runtime Configuration

The behavior of the filesystem functions is affected by settings in php.ini.

Filesystem configuration options:

Name	**Default**	**Description**	**Changeable**
allow_url_fopen	"1"	Allows fopen()-type functions to work with URLs (available since PHP 4.0.4)	PHP_INI_SYSTEM
user_agent	NULL	Defines the user agent for PHP to send (available since PHP 4.3)	PHP_INI_ALL
default_socket_timeout	"60"	Sets the default timeout, in seconds, for socket based streams (available since PHP 4.3)	PHP_INI_ALL
from	""	Defines the anonymous FTP password (your email address)	PHP_INI_ALL
auto_detect_line_endings	"0"	When set to "1", PHP will examine the	PHP_INI_ALL

		data read by fgets() and file() to see if it is using Unix, MS-Dos or Mac line-ending characters (available since PHP 4.3)	

Unix / Windows Compatibility

When specifying a path on Unix platforms, the forward slash (/) is used as directory separator. However, on Windows platforms, both forward slash (/) and backslash (\) can be used.

PHP Filesystem Functions

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

Function	Description	PHP
basename()	Returns the filename component of a path	3
chgrp()	Changes the file group	3
chmod()	Changes the file mode	3
chown()	Changes the file owner	3
clearstatcache()	Clears the file status cache	3
copy()	Copies a file	3
delete()	See unlink() or unset()	
dirname()	Returns the directory name component of a path	3
disk_free_space()	Returns the free space of a directory	4
disk_total_space()	Returns the total size of a directory	4
diskfreespace()	Alias of disk_free_space()	3
fclose()	Closes an open file	3
feof()	Tests for end-of-file on an open file	3
fflush()	Flushes buffered output to an open file	4
fgetc()	Returns a character from an open file	3
fgetcsv()	Parses a line from an open file, checking for CSV fields	3
fgets()	Returns a line from an open file	3
fgetss()	Returns a line, with HTML and PHP tags removed, from an open file	3
file()	Reads a file into an array	3
file_exists()	Checks whether or not a file or directory exists	3
file_get_contents()	Reads a file into a string	4
file_put_contents	Writes a string to a file	5
fileatime()	Returns the last access time of a file	3
filectime()	Returns the last change time of a file	3
filegroup()	Returns the group ID of a file	3
fileinode()	Returns the inode number of a file	3
filemtime()	Returns the last modification time of a file	3
fileowner()	Returns the user ID (owner) of a file	3
fileperms()	Returns the permissions of a file	3
filesize()	Returns the file size	3
filetype()	Returns the file type	3
flock()	Locks or releases a file	3
fnmatch()	Matches a filename or string against a specified pattern	4
fopen()	Opens a file or URL	3
fpassthru()	Reads from an open file, until EOF, and writes the result to the output buffer	3
fputcsv()	Formats a line as CSV and writes it to an open file	5
fputs()	Alias of fwrite()	3
fread()	Reads from an open file	3

fscanf()	Parses input from an open file according to a specified format	4
fseek()	Seeks in an open file	3
fstat()	Returns information about an open file	4
ftell()	Returns the current position in an open file	3
ftruncate()	Truncates an open file to a specified length	4
fwrite()	Writes to an open file	3
glob()	Returns an array of filenames / directories matching a specified pattern	4
is_dir()	Checks whether a file is a directory	3
is_executable()	Checks whether a file is executable	3
is_file()	Checks whether a file is a regular file	3
is_link()	Checks whether a file is a link	3
is_readable()	Checks whether a file is readable	3
is_uploaded_file()	Checks whether a file was uploaded via HTTP POST	3
is_writable()	Checks whether a file is writeable	4
is_writeable()	Alias of is_writable()	3
link()	Creates a hard link	3
linkinfo()	Returns information about a hard link	3
lstat()	Returns information about a file or symbolic link	3
mkdir()	Creates a directory	3
move_uploaded_file()	Moves an uploaded file to a new location	4
parse_ini_file()	Parses a configuration file	4
pathinfo()	Returns information about a file path	4
pclose()	Closes a pipe opened by popen()	3
popen()	Opens a pipe	3
readfile()	Reads a file and writes it to the output buffer	3
readlink()	Returns the target of a symbolic link	3
realpath()	Returns the absolute pathname	4
rename()	Renames a file or directory	3
rewind()	Rewinds a file pointer	3
rmdir()	Removes an empty directory	3
set_file_buffer()	Sets the buffer size of an open file	3
stat()	Returns information about a file	3
symlink()	Creates a symbolic link	3
tempnam()	Creates a unique temporary file	3
tmpfile()	Creates a unique temporary file	3
touch()	Sets access and modification time of a file	3
umask()	Changes file permissions for files	3
unlink()	Deletes a file	3

PHP Filesystem Constants

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

Constant	Description	PHP
GLOB_BRACE		
GLOB_ONLYDIR		
GLOB_MARK		
GLOB_NOSORT		
GLOB_NOCHECK		
GLOB_NOESCAPE		
PATHINFO_DIRNAME		
PATHINFO_BASENAME		
PATHINFO_EXTENSION		
FILE_USE_INCLUDE_PATH		
FILE_APPEND		

FILE_IGNORE_NEW_LINES		
FILE_SKIP_EMPTY_LINES		

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.

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/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";
  }
?>
```

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/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"

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

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>
<body>
</body>
</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).

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.

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

Note: For the mail functions to be available, PHP requires an installed and working email system. The program to be used is defined by the configuration settings in the php.ini file. Read more in our PHP Mail reference.

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: $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

PHP Mail Reference

For more information about the PHP mail() function, visit our PHP Mail Reference.

PHP Secure E-mails

There is a weakness in the PHP e-mail script in the previous chapter.

PHP E-mail Injections

First, look at the PHP code from the previous chapter:

```
<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: $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>
```

The problem with the code above is that unauthorized users can insert data into the mail headers via the input form.

What happens if the user adds the following text to the email input field in the form?

```
someone@example.com%0ACc:person2@example.com
%0ABcc:person3@example.com,person3@example.com,
anotherperson4@example.com,person5@example.com
%0ABTo:person6@example.com
```

The mail() function puts the text above into the mail headers as usual, and now the header has an extra Cc:, Bcc:, and To: field. When the user clicks the submit button, the e-mail will be sent to all of the addresses above!

PHP Stopping E-mail Injections

The best way to stop e-mail injections is to validate the input.

The code below is the same as in the previous chapter, but now we have added an input validator that checks the email field in the form:

```
<html>
<body>
<?php
function spamcheck($field)
  {
//eregi() performs a case insensitive regular expression match
  if(eregi("to:",$field) || eregi("cc:",$field))
    {
    return TRUE;
    }
  else
```

```
    {
    return FALSE;
    }
  }
//if "email" is filled out, send email
if (isset($_REQUEST['email']))
  {
  //check if the email address is invalid
  $mailcheck = spamcheck($_REQUEST['email']);
  if ($mailcheck==TRUE)
    {
    echo "Invalid input";
    }
  else
    {
    //send email
    $email = $_REQUEST['email'] ;
    $subject = $_REQUEST['subject'] ;
    $message = $_REQUEST['message'] ;
    mail("someone@example.com", "Subject: $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>
```

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.

PHP Error Handling

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 lets 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:

```
Custom 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
```

Now that we have learned to create our own errors and how to trigger them, lets take a look at error logging.

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.

PHP Exception Handling

Exceptions are used to change the normal flow of a script if a specified error occurs

---

What is an Exception

With PHP 5 came a new object oriented way of dealing with errors.

Exception handling is used to change the normal flow of the code execution if a specified error (exceptional) condition occurs. This condition is called an exception.

This is what normally happens when an exception is triggered:

- The current code state is saved
- The code execution will switch to a predefined (custom) exception handler function
- Depending on the situation, the handler may then resume the execution from the saved code state, terminate the script execution or continue the script from a different location in the code

We will show different error handling methods:

- Basic use of Exceptions
- Creating a custom exception handler
- Multiple exceptions
- Re-throwing an exception
- Setting a top level exception handler

Note: Exceptions should only be used with error conditions, and should not be used to jump to another place in the code at a specified point.

Basic Use of Exceptions

When an exception is thrown, the code following it will not be executed, and PHP will try to find the matching "catch" block.

If an exception is not caught, a fatal error will be issued with an "Uncaught Exception" message.

Lets try to throw an exception without catching it:

```
<?php
//create function with an exception
function checkNum($number)
 {
```

```
 if($number>1)
  {
  throw new Exception("Value must be 1 or below");
  }
 return true;
 }

//trigger exception
checkNum(2);
?>
```

The code above will get an error like this:

```
Fatal error: Uncaught exception 'Exception'
with message 'Value must be 1 or below' in C:\webfolder\test.php:6
Stack trace: #0 C:\webfolder\test.php(12):
checkNum(28) #1 {main} thrown in C:\webfolder\test.php on line 6
```

Try, throw and catch

To avoid the error from the example above, we need to create the proper code to handle an exception.

Proper exception code should include:

1. Try - A function using an exception should be in a "try" block. If the exception does not trigger, the code will continue as normal. However if the exception triggers, an exception is "thrown"
2. Throw - This is how you trigger an exception. Each "throw" must have at least one "catch"
3. Catch - A "catch" block retrieves an exception and creates an object containing the exception information

Lets try to trigger an exception with valid code:

```
<?php
//create function with an exception
function checkNum($number)
 {
 if($number>1)
  {
  throw new Exception("Value must be 1 or below");
  }
 return true;
 }

//trigger exception in a "try" block
try
 {
 checkNum(2);
 //If the exception is thrown, this text will not be shown
 echo 'If you see this, the number is 1 or below';
 }

//catch exception
catch(Exception $e)
 {
 echo 'Message: ' .$e->getMessage();
 }
?>
```

The code above will get an error like this:

```
Message: Value must be 1 or below
```

Example explained:

The code above throws an exception and catches it:

1. The checkNum() function is created. It checks if a number is greater than 1. If it is, an exception is thrown
2. The checkNum() function is called in a "try" block
3. The exception within the checkNum() function is thrown
4. The "catch" block retrives the exception and creates an object ($e) containing the exception information
5. The error message from the exception is echoed by calling $e->getMessage() from the exception object

However, one way to get around the "every throw must have a catch" rule is to set a top level exception handler to errors that slip trough.

Creating a Custom Exception Class

Creating a custom exception handler is quite simple. We simply create a special class with functions that can be called when an exception occurs in PHP. The class must be an extension of the exception class.

The custom exception class inherits the properties from PHP's exception class and you can add custom functions to it.

Lets create an exception class:

```
<?php
class customException extends Exception
 {
 public function errorMessage()
  {
  //error message
  $errorMsg = 'Error on line '.$this->getLine().' in '.$this->getFile()
  .': <b>'.$this->getMessage().'</b> is not a valid E-Mail address';
  return $errorMsg;
  }
 }
$email = "someone@example...com";
try
 {
 //check if
 if(filter_var($email, FILTER_VALIDATE_EMAIL) === FALSE)
  {
  //throw exception if email is not valid
  throw new customException($email);
  }
 }
catch (customException $e)
 {
 //display custom message
 echo $e->errorMessage();
 }
?>
```

The new class is a copy of the old exception class with an addition of the errorMessage() function. Since it is a copy of the old class, and it inherits the properties and methods from the old class, we can use the exception class methods like getLine() and getFile() and getMessage().

Example explained:

The code above throws an exception and catches it with a custom exception class:

1. The customException() class is created as an extension of the old exception class. This way it inherits all methods and properties from the old exception class
2. The errorMessage() function is created. This function returns an error message if an e-mail address is invalid
3. The $email variable is set to a string that is not a valid e-mail address
4. The "try" block is executed and an exception is thrown since the e-mail address is invalid
5. The "catch" block catches the exception and displays the error message

---

Multiple Exceptions

It is possible for a script to use multiple exceptions to check for multiple conditions.

It is possible to use several if..else blocks, a switch, or nest multiple exceptions. These exceptions can use different exception classes and return different error messages:

```
<?php
class customException extends Exception
{
public function errorMessage()
{
//error message
$errorMsg = 'Error on line '.$this->getLine().' in '.$this->getFile()
.': <b>'.$this->getMessage().'</b> is not a valid E-Mail address';
return $errorMsg;
}
}

$email = "someone@example.com";

try
 {
 //check if
 if(filter_var($email, FILTER_VALIDATE_EMAIL) === FALSE)
  {
  //throw exception if email is not valid
  throw new customException($email);
  }
 //check for "example" in mail address
 if(strpos($email, "example") !== FALSE)
  {
  throw new Exception("$email is an example e-mail");
  }
 }

catch (customException $e)
 {
 echo $e->errorMessage();
 }
catch(Exception $e)
 {
 echo $e->getMessage();
 }
```

```
?>
```

Example explained:

The code above tests two conditions and throws an exception if any of the conditions are not met:

1. The customException() class is created as an extension of the old exception class. This way it inherits all methods and properties from the old exception class
2. The errorMessage() function is created. This function returns an error message if an e-mail address is invalid
3. The $email variable is set to a string that is a valid e-mail address, but contains the string "example"
4. The "try" block is executed and an exception is not thrown on the first condition
5. The second condition triggers an exception since the e-mail contains the string "example"
6. The "catch" block catches the exception and displays the correct error message

If there was no customException catch, only the base exception catch, the exception would be handled there

---

Re-throwing Exceptions

Sometimes, when an exception is thrown, you may wish to handle it differently than the standard way. It is possible to throw an exception a second time within a "catch" block.

A script should hide system errors from users. System errors may be important for the coder, but is of no intrest to the user. To make things easier for the user you can re-throw the exception with a user friendly message:

```
<?php
class customException extends Exception
 {
 public function errorMessage()
  {
  //error message
  $errorMsg = $this->getMessage().' is not a valid E-Mail address.';
  return $errorMsg;
  }
 }
$email = "someone@example.com";
try
 {
 try
  {
  //check for "example" in mail address
  if(strpos($email, "example") !== FALSE)
   {
   //throw exception if email is not valid
   throw new Exception($email);
   }
  }
 catch(Exception $e)
  {
  //re-throw exception
  throw new customException($email);
  }
 }
catch (customException $e)
 {
 //display custom message
 echo $e->errorMessage();
 }
?>
```

Example explained:

The code above tests if the email-address contains the string "example" in it, if it does, the exception is re-thrown:

1. The customException() class is created as an extension of the old exception class. This way it inherits all methods and properties from the old exception class
2. The errorMessage() function is created. This function returns an error message if an e-mail address is invalid
3. The $email variable is set to a string that is a valid e-mail address, but contains the string "example"
4. The "try" block contains another "try" block to make it possible to re-throw the exception
5. The exception is triggered since the e-mail contains the string "example"
6. The "catch" block catches the exception and re-throws a "customException"
7. The "customException" is caught and displays an error message

If the exception is not caught in it's current "try" block, it will search for a catch block on "higher levels".

Set a Top Level Exception Handler

The set_exception_handler() function sets a user-defined function to handle all uncaught exceptions.

```
<?php
function myException($exception)
{
echo "<b>Exception:</b> " , $exception->getMessage();
}
set_exception_handler('myException');
throw new Exception('Uncaught Exception occurred');
?>
```

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

```
Exception: Uncaught Exception occurred
```

In the code above there was no "catch" block. Instead, the top level exception handler triggered. This function should be used to catch uncaught exceptions.

Rules for exceptions

- Code may be surrounded in a try block, to help catch potential exceptions
- Each try block or "throw" must have at least one corresponding catch block
- Multiple catch blocks can be used to catch different classes of exceptions
- Exceptions can be thrown (or re-thrown) in a catch block within a try block

A simple rule: If you throw something, you have to catch it.

PHP Filter

PHP filters is used to validate and filter data coming from insecure sources, like user input.

What is a PHP Filter?

A PHP filter is used to validate and filter data coming from insecure sources.

To test, validate and filter user input or custom data is an important part of any web application.

The PHP filter extension is designed to make data filtering easier and quicker.

Why use a Filter?

Almost all web applications depend on external input. Usually this comes from a user or another application (like a web service). By using filters you can be sure your application gets the correct input type.

You should always filter all external data!

Input filtering is one of the most important application security issues.

What is external data?

- Input data from a form
- Cookies
- Web services data
- Server variables
- Database query results

Functions and Filters

To filter a variable, use one of the following filter functions:

- filter_var() - Filters a single variable with a specified filter
- filter_var_array() - Filter several variables with the same or different filters
- filter_input - Get one input variable and filter it
- filter_input_array - Get several input variables and filter them with the same or different filters

In the example below, we validate an integer using the filter_var() function:

```
<?php
$int = 123;
if(!filter_var($int, FILTER_VALIDATE_INT))
 {
 echo("Integer is not valid");
 }
else
 {
 echo("Integer is valid");
 }
?>
```

The code above uses the "FILTER_VALIDATE_INT" filter to filter the variable. Since the integer is valid, the output of the code above will be: "Integer is valid".

If we try with a variable that is not an integer (like "123abc"), the output will be: "Integer is not valid".

For a complete list of functions and filters, visit our PHP Filter Reference.

Validating and Sanitizing

There are two kinds of filters:

Validating filters:

- Are used to validate user input
- Strict format rules (like URL or E-Mail validating)
- Returns the expected type on success or FALSE on failure

Sanitizing filters:

- Are used to allow or disallow specified characters in a string
- No data format rules
- Always return the string

Options and Flags

Options and flags are used to add additional filtering options to the specified filters.

Different filters have different options and flags.

In the example below, we validate an integer using the filter_var() and the "min_range" and "max_range" options:

```
<?php
$var=300;
$int_options = array(
"options"=>array
 (
 "min_range"=>0,
 "max_range"=>256
 )
);
if(!filter_var($var, FILTER_VALIDATE_INT, $int_options))
 {
 echo("Integer is not valid");
 }
else
 {
 echo("Integer is valid");
 }
?>
```

Like the code above, options must be put in an associative array with the name "options". If a flag is used it does not need to be in an array.

Since the integer is "300" it is not in the specified range, and the output of the code above will be: "Integer is not valid".

For a complete list of functions and filters, visit our PHP Filter Reference. Check each filter to see what options and flags are available.

Validate Input

Let's try validating input from a form.

The first thing we need to do is to confirm that the input data we are looking for exists.

Then we filter the input data using the filter_input() function.

In the example below, the input variable "email" is sent to the PHP page:

```
<?php
if(!filter_has_var(INPUT_GET, "email"))
 {
 echo("Input type does not exist");
 }
else
 {
 if (!filter_input(INPUT_GET, "email", FILTER_VALIDATE_EMAIL))
  {
  echo "E-Mail is not valid";
  }
 else
  {
  echo "E-Mail is valid";
  }
 }
?>
```

Example Explained

The example above has an input (email) sent to it using the "GET" method:

1. Check if an "email" input variable of the "GET" type exist
2. If the input variable exists, check if it is a valid e-mail address

---

Sanitize Input

Let's try cleaning up an URL sent from a form.

First we confirm that the input data we are looking for exists.

Then we sanitize the input data using the filter_input() function.

In the example below, the input variable "url" is sent to the PHP page:

```
<?php
if(!filter_has_var(INPUT_POST, "url"))
 {
 echo("Input type does not exist");
 }
else
 {
 $url = filter_input(INPUT_POST,
 "url", FILTER_SANITIZE_URL);
 }
?>
```

Example Explained

The example above has an input (url) sent to it using the "POST" method:

1. Check if the "url" input of the "POST" type exists
2. If the input variable exists, sanitize (take away invalid characters) and store it in the $url variable

If the input variable is a string like this "http://www.W3ååSchøøools.com/", the $url variable after the sanitizing will look like this:

```
http://www.W3Schools.com/
```

Filter Multiple Inputs

A form almost always consist of more than one input field. To avoid calling the filter_var or filter_input functions over and over, we can use the filter_var_array or the filter_input_array functions.

In this example we use the filter_input_array() function to filter three GET variables. The received GET variables is a name, an age and an e-mail address:

```
<?php
$filters = array
 (
 "name" => array
  (
  "filter"=>FILTER_SANITIZE_STRING
  ),
 "age" => array
  (
  "filter"=>FILTER_VALIDATE_INT,
  "options"=>array
   (
   "min_range"=>1,
   "max_range"=>120
   )
  ),
 "email"=> FILTER_VALIDATE_EMAIL,
 );
$result = filter_input_array(INPUT_GET, $filters);
if (!$result["age"])
 {
 echo("Age must be a number between 1 and 120.<br />");
 }
elseif(!$result["email"])
 {
 echo("E-Mail is not valid.<br />");
 }
else
 {
```

```
 echo("User input is valid");
 }
?>
```

Example Explained

The example above has three inputs (name, age and email) sent to it using the "GET" method:

1. Set an array containing the name of input variables and the filters used on the specified input variables
2. Call the filter_input_array() function with the GET input variables and the array we just set
3. Check the "age" and "email" variables in the $result variable for invalid inputs. (If any of the input variables are invalid, that input variable will be FALSE after the filter_input_array() function)

The second parameter of the filter_input_array() function can be an array or a single filter ID.

If the parameter is a single filter ID all values in the input array are filtered by the specified filter.

If the parameter is an array it must follow these rules:

- Must be a associative array containing an input variable as an array key (like the "age" input variable)
- The array value must be a filter ID or an array specifying the filter, flags and options

---

Using Filter Callback

It is possible to call a user defined function and use it as a filter using the FILTER_CALLBACK filter. This way, we have full control of the data filtering.

You can create your own user defined function or use an existing PHP function

The function you wish to use to filter is specified the same way as an option is specified. In an associative array with the name "options"

In the example below, we us a user created function to convert all "_" to whitespaces:

```
<?php
function convertSpace($string)
{
return str_replace("_", " ", $string);
}

$string = "Peter_is_a_great_guy!";

echo filter_var($string, FILTER_CALLBACK,
array("options"=>"convertSpace"));
?>
```

The result from the code above should look like this:

```
Peter is a great guy!
```

Example Explained

The example above converts all "_" to whitespaces:

1. Create a function to replace "_" to whitespaces
2. Call the filter_var() function with the FILTER_CALLBACK filter and an array containing our function

PHP MySQL Introduction

MySQL is the most popular open source database server.

What is MySQL?

MySQL is a database. A database defines a structure for storing information.

In a database, there are tables. Just like HTML tables, database tables contain rows, columns, and cells.

Databases are useful when storing information categorically. A company may have a database with the following tables: "Employees", "Products", "Customers" and "Orders".

Database Tables

A database most often contains one or more tables. Each table has a name (e.g. "Customers" or "Orders"). Each table contains records (rows) with data.

Below is an example of a table called "Persons":

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Svendson	Tove	Borgvn 23	Sandnes
Pettersen	Kari	Storgt 20	Stavanger

The table above contains three records (one for each person) and four columns (LastName, FirstName, Address, and City).

Queries

A query is a question or a request.

With MySQL, we can query a database for specific information and have a recordset returned.

Look at the following query:

```
SELECT LastName FROM Persons
```

The query above selects all the data in the LastName column in the Persons table, and will return a recordset like this:

LastName
Hansen
Svendson
Pettersen

Download MySQL Database

If you don't have a PHP server with a MySQL Database, you can download MySQL for free here: http://www.mysql.com/downloads/index.html

Facts About MySQL Database

One great thing about MySQL is that it can be scaled down to support embedded database applications. Perhaps it is because of this reputation that many people believe that MySQL can only handle small to medium-sized systems.

The truth is that MySQL is the de-facto standard database for web sites that support huge volumes of both data and end users (like Friendster, Yahoo, Google). Look at http://www.mysql.com/customers/ for an overview of companies that use MySQL.

PHP MySQL Connect to a Database

The free MySQL Database is very often used with PHP.

Connecting to a MySQL Database

Before you can access and work with data in a database, you must create a connection to the database.

In PHP, this is done with the mysql_connect() function.

Syntax

```
mysql_connect(servername,username,password);
```

Parameter	Description
servername	Optional. Specifies the server to connect to. Default value is "localhost:3306"
username	Optional. Specifies the username to log in with. Default value is the name of the user that owns the server process
password	Optional. Specifies the password to log in with. Default is ""

Note: There are more available parameters, but the ones listed above are the most important. Visit our full PHP MySQL Reference for more details.

Example

In the following example we store the connection in a variable ($con) for later use in the script. The "die" part will be executed if the connection fails:

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }
// some code
?>
```

Closing a Connection

The connection will be closed as soon as the script ends. To close the connection before, use the mysql_close() function.

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }
// some code
mysql_close($con);
?>
```

PHP MySQL Create Database and Tables

A database holds one or multiple tables.

Create a Database

The CREATE DATABASE statement is used to create a database in MySQL.

Syntax

```
CREATE DATABASE database_name
```

To get PHP to execute the statement above we must use the mysql_query() function. This function is used to send a query or command to a MySQL connection.

Example

In the following example we create a database called "my_db":

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }
if (mysql_query("CREATE DATABASE my_db",$con))
  {
  echo "Database created";
  }
else
  {
  echo "Error creating database: " . mysql_error();
  }
mysql_close($con);
?>
```

Create a Table

The CREATE TABLE statement is used to create a database table in MySQL.

Syntax

```
CREATE TABLE table_name
(
column_name1 data_type,
column_name2 data_type,
column_name3 data_type,
.......
)
```

We must add the CREATE TABLE statement to the mysql_query() function to execute the command.

Example

The following example shows how you can create a table named "person", with three columns. The column names will be "FirstName", "LastName" and "Age":

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }
// Create database
if (mysql_query("CREATE DATABASE my_db",$con))
  {
  echo "Database created";
  }
else
  {
  echo "Error creating database: " . mysql_error();
  }
// Create table in my_db database
mysql_select_db("my_db", $con);
$sql = "CREATE TABLE person
(
FirstName varchar(15),
LastName varchar(15),
Age int
)";
mysql_query($sql,$con);
mysql_close($con);
?>
```

Important: A database must be selected before a table can be created. The database is selected with the mysql_select_db() function.

Note: When you create a database field of type varchar, you must specify the maximum length of the field, e.g. varchar(15).

MySQL Data Types

Below is the different MySQL data types that can be used:

Numeric Data Types	**Description**

int(size)
smallint(size)
tinyint(size)
mediumint(size)
	Hold integers only. The maximum number of digits can be specified in the size parameter

bigint(size)	

decimal(size,d)
double(size,d)
float(size,d)
	Hold numbers with fractions. The maximum number of digits can be specified in the size parameter. The maximum number of digits to the right of the decimal is specified in the d parameter

Textual Data Types	Description
char(size)	Holds a fixed length string (can contain letters, numbers, and special characters). The fixed size is specified in parenthesis
varchar(size)	Holds a variable length string (can contain letters, numbers, and special characters). The maximum size is specified in parenthesis
tinytext	Holds a variable string with a maximum length of 255 characters
text blob	Holds a variable string with a maximum length of 65535 characters
mediumtext mediumblob	Holds a variable string with a maximum length of 16777215 characters
longtext longblob	Holds a variable string with a maximum length of 4294967295 characters

Date Data Types	Description

date(yyyy-mm-dd)
datetime(yyyy-mm-dd hh:mm:ss)
timestamp(yyyymmddhhmmss)
time(hh:mm:ss)
	Holds date and/or time

Misc. Data Types	Description
enum(value1,value2,ect)	ENUM is short for ENUMERATED list. Can store one of up to 65535 values listed within the ( ) brackets. If a value is inserted that is not in the list, a blank value will be inserted
set	SET is similar to ENUM. However, SET can have up to 64 list items and can store more than one choice

---

Primary Keys and Auto Increment Fields

Each table should have a primary key field.

A primary key is used to uniquely identify the rows in a table. Each primary key value must be unique within the table. Furthermore, the primary key field cannot be null because the database engine requires a value to locate the record.

The primary key field is always indexed. There is no exception to this rule! You must index the primary key field so the database engine can quickly locate rows based on the key's value.

The following example sets the personID field as the primary key field. The primary key field is often an ID number, and is often used with the AUTO_INCREMENT setting. AUTO_INCREMENT automatically increases the value of the field by 1 each time a new record is added. To ensure that the primary key field cannot be null, we must add the NOT NULL setting to the field.

Example

```
$sql = "CREATE TABLE person
(
personID int NOT NULL AUTO_INCREMENT,
PRIMARY KEY(personID),
FirstName varchar(15),
LastName varchar(15),
Age int
)";
mysql_query($sql,$con);
```

PHP MySQL Insert Into

The INSERT INTO statement is used to insert new records into a database table.

Insert Data Into a Database Table

The INSERT INTO statement is used to add new records to a database table.

Syntax

```
INSERT INTO table_name
VALUES (value1, value2,....)
```

You can also specify the columns where you want to insert the data:

```
INSERT INTO table_name (column1, column2,...)
VALUES (value1, value2,....)
```

Note: SQL statements are not case sensitive. INSERT INTO is the same as insert into.

To get PHP to execute the statements above we must use the mysql_query() function. This function is used to send a query or command to a MySQL connection.

Example

In the previous chapter we created a table named "Person", with three columns; "Firstname", "Lastname" and "Age". We will use the same table in this example. The following example adds two new records to the "Person" table:

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }
mysql_select_db("my_db", $con);
mysql_query("INSERT INTO person (FirstName, LastName, Age)
VALUES ('Peter', 'Griffin', '35')");
mysql_query("INSERT INTO person (FirstName, LastName, Age)
VALUES ('Glenn', 'Quagmire', '33')");
mysql_close($con);
?>
```

Insert Data From a Form Into a Database

Now we will create an HTML form that can be used to add new records to the "Person" table.

Here is the HTML form:

```
<html>
<body>
```

```
<form action="insert.php" method="post">
Firstname: <input type="text" name="firstname" />
Lastname: <input type="text" name="lastname" />
Age: <input type="text" name="age" />
<input type="submit" />
</form>
</body>
</html>
```

When a user clicks the submit button in the HTML form in the example above, the form data is sent to "insert.php". The "insert.php" file connects to a database, and retrieves the values from the form with the PHP $_POST variables. Then, the mysql_query() function executes the INSERT INTO statement, and a new record will be added to the database table.

Below is the code in the "insert.php" page:

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }
mysql_select_db("my_db", $con);
$sql="INSERT INTO person (FirstName, LastName, Age)
VALUES
('$_POST[firstname]','$_POST[lastname]','$_POST[age]')";
if (!mysql_query($sql,$con))
  {
  die('Error: ' . mysql_error());
  }
echo "1 record added";
mysql_close($con)
?>
```

PHP MySQL Select

The SELECT statement is used to select data from a database.

Select Data From a Database Table

The SELECT statement is used to select data from a database.

Syntax

```
SELECT column_name(s)
FROM table_name
```

Note: SQL statements are not case sensitive. SELECT is the same as select.

To get PHP to execute the statement above we must use the mysql_query() function. This function is used to send a query or command to a MySQL connection.

Example

The following example selects all the data stored in the "Person" table (The * character selects all of the data in the table):

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }
mysql_select_db("my_db", $con);
$result = mysql_query("SELECT * FROM person");
while($row = mysql_fetch_array($result))
  {
  echo $row['FirstName'] . " " . $row['LastName'];
  echo "<br />";
  }
mysql_close($con);
?>
```

The example above stores the data returned by the mysql_query() function in the $result variable. Next, we use the mysql_fetch_array() function to return the first row from the recordset as an array. Each subsequent call to mysql_fetch_array() returns the next row in the recordset. The while loop loops through all the records in the recordset. To print the value of each row, we use the PHP $row variable ($row['FirstName'] and $row['LastName']).

The output of the code above will be:

```
Peter Griffin
Glenn Quagmire
```

Display the Result in an HTML Table

The following example selects the same data as the example above, but will display the data in an HTML table:

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }

mysql_select_db("my_db", $con);

$result = mysql_query("SELECT * FROM person");

echo "<table border='1'>
<tr>
<th>Firstname</th>
<th>Lastname</th>
</tr>";
while($row = mysql_fetch_array($result))
  {
  echo "<tr>";
  echo "<td>" . $row['FirstName'] . "</td>";
  echo "<td>" . $row['LastName'] . "</td>";
  echo "</tr>";
  }
```

```
echo "</table>";
mysql_close($con);
?>
```

The output of the code above will be:

Firstname	Lastname
Glenn	Quagmire
Peter	Griffin

PHP MySQL The Where Clause

To select only data that matches a specified criteria, add a WHERE clause to the SELECT statement.

The WHERE clause

To select only data that matches a specific criteria, add a WHERE clause to the SELECT statement.

Syntax

```
SELECT column FROM table
WHERE column operator value
```

The following operators can be used with the WHERE clause:

Operator	Description
=	Equal
!=	Not equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
BETWEEN	Between an inclusive range
LIKE	Search for a pattern

Note: SQL statements are not case sensitive. WHERE is the same as where.

To get PHP to execute the statement above we must use the mysql_query() function. This function is used to send a query or command to a MySQL connection.

Example

The following example will select all rows from the "Person" table, where FirstName='Peter':

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }

mysql_select_db("my_db", $con);
```

```
$result = mysql_query("SELECT * FROM person
WHERE FirstName='Peter'");

while($row = mysql_fetch_array($result))
  {
  echo $row['FirstName'] . " " . $row['LastName'];
  echo "<br />";
  }

?>
```

The output of the code above will be:

```
Peter Griffin
```

PHP MySQL Order By Keyword

The ORDER BY keyword is used to sort the data in a recordset.

The ORDER BY Keyword

The ORDER BY keyword is used to sort the data in a recordset.

Syntax

```
SELECT column_name(s)
FROM table_name
ORDER BY column_name
```

Note: SQL statements are not case sensitive. ORDER BY is the same as order by.

Example

The following example selects all the data stored in the "Person" table, and sorts the result by the "Age" column:

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }

mysql_select_db("my_db", $con);

$result = mysql_query("SELECT * FROM person ORDER BY age");

while($row = mysql_fetch_array($result))
  {
  echo $row['FirstName'];
  echo " " . $row['LastName'];
  echo " " . $row['Age'];
  echo "<br />";
  }
mysql_close($con);
?>
```

The output of the code above will be:

```
Glenn Quagmire 33
Peter Griffin 35
```

Sort Ascending or Descending

If you use the ORDER BY keyword, the sort-order of the recordset is ascending by default (1 before 9 and "a" before "p").

Use the DESC keyword to specify a descending sort-order (9 before 1 and "p" before "a"):

```
SELECT column_name(s)
FROM table_name
ORDER BY column_name DESC
```

Order by Two Columns

It is possible to order by more than one column. When ordering by more than one column, the second column is only used if the values in the first column are identical:

```
SELECT column_name(s)
FROM table_name
ORDER BY column_name1, column_name2
```

PHP MySQL Update

The UPDATE statement is used to modify data in a database table.

Update Data In a Database

The UPDATE statement is used to modify data in a database table.

Syntax

```
UPDATE table_name
SET column_name = new_value
WHERE column_name = some_value
```

Note: SQL statements are not case sensitive. UPDATE is the same as update.

To get PHP to execute the statement above we must use the mysql_query() function. This function is used to send a query or command to a MySQL connection.

Example

Earlier in the tutorial we created a table named "Person". Here is how it looks:

FirstName	LastName	Age
Peter	Griffin	35
Glenn	Quagmire	33

The following example updates some data in the "Person" table:

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }
mysql_select_db("my_db", $con);

mysql_query("UPDATE Person SET Age = '36'
WHERE FirstName = 'Peter' AND LastName = 'Griffin'");
mysql_close($con);
?>
```

After the update, the "Person" table will look like this:

FirstName	LastName	Age
Peter	Griffin	36
Glenn	Quagmire	33

PHP MySQL Delete From

The DELETE FROM statement is used to delete rows from a database table.

Delete Data In a Database

The DELETE FROM statement is used to delete records from a database table.

Syntax

<

```
DELETE FROM table_name
WHERE column_name = some_value
```

Note: SQL statements are not case sensitive. DELETE FROM is the same as delete from.

To get PHP to execute the statement above we must use the mysql_query() function. This function is used to send a query or command to a MySQL connection.

Example

Earlier in the tutorial we created a table named "Person". Here is how it looks:

FirstName	LastName	Age
Peter	Griffin	35
Glenn	Quagmire	33

The following example deletes all the records in the "Person" table where LastName='Griffin':

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }
mysql_select_db("my_db", $con);
```

```
mysql_query("DELETE FROM Person WHERE LastName='Griffin'");
mysql_close($con);
?>
```

After the deletion, the table will look like this:

FirstName	LastName	Age
Glenn	Quagmire	33

PHP Database ODBC

ODBC is an Application Programming Interface (API) that allows you to connect to a data source (e.g. an MS Access database).

Create an ODBC Connection

With an ODBC connection, you can connect to any database, on any computer in your network, as long as an ODBC connection is available.

Here is how to create an ODBC connection to a MS Access Database:

1. Open the **Administrative Tools** icon in your Control Panel.
2. Double-click on the **Data Sources (ODBC)** icon inside.
3. Choose the **System DSN** tab.
4. Click on **Add** in the System DSN tab.
5. Select the **Microsoft Access Driver**. Click **Finish.**
6. In the next screen, click **Select** to locate the database.
7. Give the database a **Data Source Name (DSN)**.
8. Click **OK**.

Note that this configuration has to be done on the computer where your web site is located. If you are running Internet Information Server (IIS) on your own computer, the instructions above will work, but if your web site is located on a remote server, you have to have physical access to that server, or ask your web host to to set up a DSN for you to use.

Connecting to an ODBC

The odbc_connect() function is used to connect to an ODBC data source. The function takes four parameters: the data source name, username, password, and an optional cursor type.

The odbc_exec() function is used to execute an SQL statement.

Example

The following example creates a connection to a DSN called northwind, with no username and no password. It then creates an SQL and executes it:

```
$conn=odbc_connect('northwind','','');
$sql="SELECT * FROM customers";
$rs=odbc_exec($conn,$sql);
```

Retrieving Records

The odbc_fetch_row() function is used to return records from the result-set. This function returns true if it is able to return rows, otherwise false.

The function takes two parameters: the ODBC result identifier and an optional row number:

```
odbc_fetch_row($rs)
```

Retrieving Fields from a Record

The odbc_result() function is used to read fields from a record. This function takes two parameters: the ODBC result identifier and a field number or name.

The code line below returns the value of the first field from the record:

```
$compname=odbc_result($rs,1);
```

The code line below returns the value of a field called "CompanyName":

```
$compname=odbc_result($rs,"CompanyName");
```

Closing an ODBC Connection

The odbc_close() function is used to close an ODBC connection.

```
odbc_close($conn);
```

An ODBC Example

The following example shows how to first create a database connection, then a result-set, and then display the data in an HTML table.

```
<html>
<body>
<?php
$conn=odbc_connect('northwind','','');
if (!$conn)
  {exit("Connection Failed: " . $conn);}
$sql="SELECT * FROM customers";
$rs=odbc_exec($conn,$sql);
if (!$rs)
  {exit("Error in SQL");}
echo "<table><tr>";
echo "<th>Companyname</th>";
echo "<th>Contactname</th></tr>";
while (odbc_fetch_row($rs))
{
  $compname=odbc_result($rs,"CompanyName");
  $conname=odbc_result($rs,"ContactName");
  echo "<tr><td>$compname</td>";
  echo "<td>$conname</td></tr>";
}
```

```
odbc_close($conn);
echo "</table>";
?>
</body>
</html>
```

PHP XML Expat Parser

The built-in Expat parser makes it possible to process XML documents in PHP.

What is XML?

XML is used to describe data and to focus on what data is. An XML file describes the structure of the data.

In XML, no tags are predefined. You must define your own tags.

If you want to learn more about XML, please visit our XML tutorial.

What is Expat?

To read and update - create and manipulate - an XML document, you will need an XML parser.

There are two basic types of XML parsers:

- Tree-based parser: This parser transforms an XML document into a tree structure. It analyzes the whole document, and provides access to the tree elements. e.g. the Document Object Model (DOM)
- Event-based parser: Views an XML document as a series of events. When a specific event occurs, it calls a function to handle it

The Expat parser is an event-based parser.

Event-based parsers focus on the content of the XML documents, not their structure. Because of this, event-based parsers can access data faster than tree-based parsers.

Look at the following XML fraction:

```
<from>Jani</from>
```

An event-based parser reports the XML above as a series of three events:

- Start element: from
- Start CDATA section, value: Jani
- Close element: from

The XML example above contains well-formed XML. However, the example is not valid XML, because there is no Document Type Definition (DTD) associated with it.

However, this makes no difference when using the Expat parser. Expat is a non-validating parser, and ignores any DTDs.

As an event-based, non-validating XML parser, Expat is fast and small, and a perfect match for PHP web applications.

Note: XML documents must be well-formed or Expat will generate an error.

Installation

The XML Expat parser functions are part of the PHP core. There is no installation needed to use these functions.

An XML File

The XML file below will be used in our example:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

Initializing the XML Parser

We want to initialize the XML parser in PHP, define some handlers for different XML events, and then parse the XML file.

Example

```
<?php
//Initialize the XML parser
$parser=xml_parser_create();
//Function to use at the start of an element
function start($parser,$element_name,$element_attrs)
  {
  switch($element_name)
    {
    case "NOTE":
    echo "-- Note --<br />";
    break;
    case "TO":
    echo "To: ";
    break;
    case "FROM":
    echo "From: ";
    break;
    case "HEADING":
    echo "Heading: ";
    break;
    case "BODY":
    echo "Message: ";
    }
  }
//Function to use at the end of an element
function stop($parser,$element_name)
  {
```

```
  echo "<br />";
  }
//Function to use when finding character data
function char($parser,$data)
  {
  echo $data;
  }
//Specify element handler
xml_set_element_handler($parser,"start","stop");
//Specify data handler
xml_set_character_data_handler($parser,"char");
//Open XML file
$fp=fopen("test.xml","r");
//Read data
while ($data=fread($fp,4096))
  {
  xml_parse($parser,$data,feof($fp)) or
  die (sprintf("XML Error: %s at line %d",
  xml_error_string(xml_get_error_code($parser)),
  xml_get_current_line_number($parser)));
  }
//Free the XML parser
xml_parser_free($parser);
?>
```

The output of the code above will be:

```
-- Note --
To: Tove
From: Jani
Heading: Reminder
Message: Don't forget me this weekend!
```

How it works:

1. Initialize the XML parser with the xml_parser_create() function
2. Create functions to use with the different event handlers
3. Add the xml_set_element_handler() function to specify which function will be executed when the parser encounters the opening and closing tags
4. Add the xml_set_character_data_handler() function to specify which function will execute when the parser encounters character data
5. Parse the file "test.xml" with the xml_parse() function
6. In case of an error, add xml_error_string() function to convert an XML error to a textual description
7. Call the xml_parser_free() function to release the memory allocated with the xml_parser_create() function

---

More PHP Expat Parser

For more information about the PHP Expat functions, visit our PHP XML Parser Reference.

PHP XML DOM

The built-in DOM parser makes it possible to process XML documents in PHP.

What is DOM?

The W3C DOM provides a standard set of objects for HTML and XML documents, and a standard interface for accessing and manipulating them.

The W3C DOM is separated into different parts (Core, XML, and HTML) and different levels (DOM Level 1/2/3):

* Core DOM - defines a standard set of objects for any structured document
* XML DOM - defines a standard set of objects for XML documents
* HTML DOM - defines a standard set of objects for HTML documents

If you want to learn more about the XML DOM, please visit our XML DOM tutorial.

XML Parsing

To read and update - create and manipulate - an XML document, you will need an XML parser.

There are two basic types of XML parsers:

- Tree-based parser: This parser transforms an XML document into a tree structure. It analyzes the whole document, and provides access to the tree elements
- Event-based parser: Views an XML document as a series of events. When a specific event occurs, it calls a function to handle it

The DOM parser is an tree-based parser.

Look at the following XML document fraction:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<from>Jani</from>
```

The XML DOM sees the XML above as a tree structure:

- Level 1: XML Document
- Level 2: Root element: <from>
- Level 3: Text element: "Jani"

Installation

The DOM XML parser functions are part of the PHP core. There is no installation needed to use these functions.

An XML File

The XML file below will be used in our example:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
```

```
</note>
```

Load and Output XML

We want to initialize the XML parser, load the xml, and output it:

Example

```
<?php
$xmlDoc = new DOMDocument();
$xmlDoc->load("note.xml");

print $xmlDoc->saveXML();
?>
```

The output of the code above will be:

```
Tove Jani Reminder Don't forget me this weekend!
```

If you select "View source" in the browser window, you will see the following HTML:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

The example above creates a DOMDocument-Object and loads the XML from "note.xml" into it.

Then the saveXML() function to puts the internal XML document into a string, so that we can print it.

Looping through XML

We want to initialize the XML parser, load the XML, and loop through all elements of the <note> element:

Example

```
<?php
$xmlDoc = new DOMDocument();
$xmlDoc->load("note.xml");
$x = $xmlDoc->documentElement;
foreach ($x->childNodes AS $item)
  {
  print $item->nodeName . " = " . $item->nodeValue . "<br />";
  }
?>
```

The output of the code above will be:

```
#text =
to = Tove
#text =
from = Jani
#text =
heading = Reminder
#text =
body = Don't forget me this weekend!
#text =
```

In the example above you see that there are empty text nodes between each element.

When XML generates, it often contains white-spaces between the nodes. The XML DOM parser treats these as ordinary elements, and if you are not aware of them, they sometimes cause problems.

If you want to learn more about the XML DOM, please visit our XML DOM tutorial.

PHP SimpleXML

SimpleXML handles the most common XML tasks and leaves the rest for other extensions.

What is SimpleXML?

SimpleXML is new in PHP 5. It is an easy way of getting an element's attributes and text, if you know the XML document's layout.

Compared to DOM or the Expat parser, SimpleXML just takes a few lines of code to read text data from an element.

SimpleXML converts the XML document into an object, like this:

- Elements - Are converted to single attributes of the SimpleXMLElement object. When there's more than one element on one level, they're placed inside an array
- Attributes - Are accessed using associative arrays, where an index corresponds to the attribute name
- Element Data - Text data from elements are converted to strings. If an element has more than one text node, they will be arranged in the order they are found

SimpleXML is fast and easy to use when performing basic tasks like:

- Reading XML files
- Extracting data from XML strings
- Editing text nodes or attributes

However, when dealing with advanced XML, like namespaces, you are better off using the Expat parser or the XML DOM.

Installation

As of PHP 5.0, the SimpleXML functions are part of the PHP core. There is no installation needed to use these functions.

Using SimpleXML

Below is an XML file:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

We want to output the element names and data from the XML file above.

Here's what to do:

1. Load the XML file
2. Get the name of the first element
3. Create a loop that will trigger on each child node, using the children() function
4. Output the element name and data for each child node

Example

```
<?php
$xml = simplexml_load_file("test.xml");
echo $xml->getName() . "<br />";
foreach($xml->children() as $child)
  {
  echo $child->getName() . ": " . $child . "<br />";
  }
?>
```

The output of the code above will be:

```
note
to: Tove
from: Jani
heading: Reminder
body: Don't forget me this weekend!
```

More PHP SimpleXML

For more information about the PHP SimpleXML functions, visit our PHP SimpleXML Reference.

AJAX Introduction

AJAX = Asynchronous JavaScript And XML

AJAX is an acronym for **A**synchronous **J**avaScript **A**nd **X**ML.

AJAX is not a new programming language, but simply a new technique for creating better, faster, and more interactive web applications.

AJAX uses JavaScript to send and receive data between a web browser and a web server.

The AJAX technique makes web pages more responsive by exchanging data with the web server behind the scenes, instead of reloading an entire web page each time a user makes a change.

AJAX Is Based On Open Standards

AJAX is based on the following open standards:

- **JavaScript**
- **XML**
- **HTML**
- **CSS**

The open standards used in AJAX are well defined, and supported by all major browsers. AJAX applications are browser and platform independent. (Cross-Platform, Cross-Browser technology)

AJAX Is About Better Internet Applications

Web applications have many benefits over desktop applications:

- they can reach a larger audience
- they are easier to install and support
- they are easier to develop

However, Internet applications are not always as "rich" and user-friendly as traditional desktop applications.

With AJAX, Internet applications can be made richer (smaller, faster, and easier to use).

You Can Start Using AJAX Today

There is nothing new to learn.

AJAX is based on open standards. These standards have been used by most developers for several years.

Most existing web applications can be rewritten to use AJAX technology instead of traditional HTML forms.

AJAX Uses XML And HTTP Requests

A traditional web application will submit input (using an HTML form) to a web server. After the web server has processed the data, it will return a completely new web page to the user.

Because the server returns a new web page each time the user submits input, traditional web applications often run slowly and tend to be less user friendly.

With AJAX, web applications can send and retrieve data without reloading the whole web page. This is done by sending HTTP requests to the server (behind the scenes), and by modifying only parts of the web page using JavaScript when the server returns data.

XML is commonly used as the format for receiving server data, although any format, including plain text, can be used.

You will learn more about how this is done in the next chapters of this tutorial.

PHP and AJAX

There is no such thing as an AJAX server.

AJAX is a technology that runs in your browser. It uses asynchronous data transfer (HTTP requests) between the browser and the web server, allowing web pages to request small bits of information from the server instead of whole pages.

AJAX is a web browser technology independent of web server software.

However, in this tutorial we will focus more on actual examples running on a PHP server, and less on **how** AJAX works.

To read more about how AJAX works, visit our AJAX tutorial.

AJAX XMLHttpRequest

The XMLHttpRequest object makes AJAX possible.

The XMLHttpRequest

The XMLHttpRequest object is the key to AJAX.

It has been available ever since Internet Explorer 5.5 was released in July 2000, but not fully discovered before people started to talk about AJAX and Web 2.0 in 2005.

Creating An XMLHttpRequest Object

Different browsers use different methods to create an **XMLHttpRequest** object.

Internet Explorer uses an **ActiveXObject**.

Other browsers uses a built in JavaScript object called **XMLHttpRequest**.

Here is the simplest code you can use to overcome this problem:

```
var XMLHttp=null
if (window.XMLHttpRequest)
  {
  XMLHttp=new XMLHttpRequest()
  }
else if (window.ActiveXObject)
  {
  XMLHttp=new ActiveXObject("Microsoft.XMLHTTP")
  }
```

Example above explained:

1. First create a variable **XMLHttp** to use as your XMLHttpRequest object. Set the value to null.
2. Then test if the object **window.XMLHttpRequest** is available. This object is available in newer versions of Firefox, Mozilla, Opera, and Safari.
3. If it's available, use it to create a new object: **XMLHttp=new XMLHttpRequest()**
4. If it's not available, test if an object **window.ActiveXObject** is available. This object is available in Internet Explorer version 5.5 and later.
5. If it is available, use it to create a new object: **XMLHttp=new ActiveXObject()**

A Better Example?

Some programmers will prefer to use the newest and fastest version of the XMLHttpRequest object.

The example below tries to load Microsoft's latest version "Msxml2.XMLHTTP", available in Internet Explorer 6, before it falls back to "Microsoft.XMLHTTP", available in Internet Explorer 5.5 and later.

```
function GetXmlHttpObject()
{
var xmlHttp=null;
try
 {
 // Firefox, Opera 8.0+, Safari
 xmlHttp=new XMLHttpRequest();
 }
catch (e)
 {
 // Internet Explorer
 try
  {
  xmlHttp=new ActiveXObject("Msxml2.XMLHTTP");
  }
 catch (e)
  {
  xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
 }
return xmlHttp;
}
```

Example above explained:

1. First create a variable **XMLHttp** to use as your XMLHttpRequest object. Set the value to null.
2. Try to create the object the according to web standards (Mozilla, Opera and Safari):**XMLHttp=new XMLHttpRequest()**
3. Try to create the object the Microsoft way, available in Internet Explorer 6 and later:**XMLHttp=new ActiveXObject("Msxml2.XMLHTTP")**
4. If this catches an error, try the older (Internet Explorer 5.5) way: **XMLHttp=new ActiveXObject("Microsoft.XMLHTTP")**

More about the XMLHttpRequest object

If you want to read more about the XMLHttpRequest, visit our AJAX tutorial.

PHP and AJAX Suggest

AJAX Suggest

In the AJAX example below we will demonstrate how a web page can communicate with a web server online as a user enters data into a web form.

Type a Name in the Box Below

First Name:

Suggestions:

This example consists of three pages:

- a simple HTML form
- a JavaScript
- a PHP page

The HTML Form

This is the HTML page. It contains a simple HTML form and a link to a JavaScript:

```
<html>
<head>
<script src="clienthint.js"></script>
</head>
<body>
<form>
First Name:
<input type="text" id="txt1"
onkeyup="showHint(this.value)">
</form>
<p>Suggestions: <span id="txtHint"></span></p>
</body>
</html>
```

Example Explained - The HTML Form

As you can see, the HTML page above contains a simple HTML form with an input field called "txt1".

The form works like this:

1. An event is triggered when the user presses, and releases a key in the input field
2. When the event is triggered, a function called showHint() is executed.
3. Below the form is a <span> called "txtHint". This is used as a placeholder for the return data of the showHint() function.

The JavaScript

The JavaScript code is stored in "clienthint.js" and linked to the HTML document:

```
var xmlHttp
```

```
function showHint(str)
{
if (str.length==0)
  {
  document.getElementById("txtHint").innerHTML=""
  return
  }
xmlHttp=GetXmlHttpObject()
if (xmlHttp==null)
  {
  alert ("Browser does not support HTTP Request")
  return
  }
var url="gethint.php"
url=url+"?q="+str
url=url+"&sid="+Math.random()
xmlHttp.onreadystatechange=stateChanged
xmlHttp.open("GET",url,true)
xmlHttp.send(null)
}

function stateChanged()
{
if (xmlHttp.readyState==4 || xmlHttp.readyState=="complete")
 {
 document.getElementById("txtHint").innerHTML=xmlHttp.responseText
 }
}
function GetXmlHttpObject()
{
var xmlHttp=null;
try
 {
 // Firefox, Opera 8.0+, Safari
 xmlHttp=new XMLHttpRequest();
 }
catch (e)
 {
 // Internet Explorer
 try
  {
  xmlHttp=new ActiveXObject("Msxml2.XMLHTTP");
  }
 catch (e)
  {
  xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
 }
return xmlHttp;
}
```

Example Explained

The showHint() Function

This function executes every time a character is entered in the input field.

If there is some input in the text field (str.length > 0) the function executes the following:

1. Defines the url (filename) to send to the server
2. Adds a parameter (q) to the url with the content of the input field
3. Adds a random number to prevent the server from using a cached file

4. Calls on the GetXmlHttpObject function to create an XMLHTTP object, and tells the object to execute a function called stateChanged when a change is triggered
5. Opens the XMLHTTP object with the given url.
6. Sends an HTTP request to the server

If the input field is empty, the function simply clears the content of the txtHint placeholder.

The stateChanged() Function

This function executes every time the state of the XMLHTTP object changes.

When the state changes to 4 (or to "complete"), the content of the txtHint placeholder is filled with the response text.

The GetXmlHttpObject() Function

AJAX applications can only run in web browsers with complete XML support.

The code above called a function called GetXmlHttpObject().

The purpose of the function is to solve the problem of creating different XMLHTTP objects for different browsers.

This is explained in the previous chapter.

The PHP Page

The server page called by the JavaScript code is a simple PHP file called "gethint.php".

The code in the "gethint.php" checks an array of names and returns the corresponding names to the client:

```
<?php
// Fill up array with names
$a[]="Anna";
$a[]="Brittany";
$a[]="Cinderella";
$a[]="Diana";
$a[]="Eva";
$a[]="Fiona";
$a[]="Gunda";
$a[]="Hege";
$a[]="Inga";
$a[]="Johanna";
$a[]="Kitty";
$a[]="Linda";
$a[]="Nina";
$a[]="Ophelia";
$a[]="Petunia";
$a[]="Amanda";
$a[]="Raquel";
$a[]="Cindy";
$a[]="Doris";
$a[]="Eve";
$a[]="Evita";
$a[]="Sunniva";
$a[]="Tove";
$a[]="Unni";
```

```
$a[]="Violet";
$a[]="Liza";
$a[]="Elizabeth";
$a[]="Ellen";
$a[]="Wenche";
$a[]="Vicky";
//get the q parameter from URL
$q=$_GET["q"];
//lookup all hints from array if length of q>0
if (strlen($q) > 0)
{
$hint="";
for($i=0; $i<count($a); $i++)
  {
  if (strtolower($q)==strtolower(substr($a[$i],0,strlen($q))))
    {
    if ($hint=="")
      {
      $hint=$a[$i];
      }
    else
      {
      $hint=$hint." , ".$a[$i];
      }
    }
  }
}

//Set output to "no suggestion" if no hint were found
//or to the correct values
if ($hint == "")
{
$response="no suggestion";
}
else
{
$response=$hint;
}

//output the response
echo $response;
?>
```

If there is any text sent from the JavaScript (strlen($q) > 0) the following happens:

1. Find a name matching the characters sent from the JavaScript
2. If more than one name is found, include all names in the response string
3. If no matching names were found, set response to "no suggestion"
4. If one or more matching names were found, set response to these names
5. The response is sent to the "txtHint" placeholder

PHP and AJAX XML Example

AJAX can be used for interactive communication with an XML file.

AJAX XML Example

In the AJAX example below we will demonstrate how a web page can fetch information from an XML file using AJAX technology.

Select a CD in the Box Below

Select a CD: Bob Dylan

CD info will be listed here.

This example consists of four pages:

- a simple HTML form
- an XML file
- a JavaScript
- a PHP page

The HTML Form

The example above contains a simple HTML form and a link to a JavaScript:

```
<html>
<head>
<script src="selectcd.js"></script>
</head>
<body>
<form>
Select a CD:
<select name="cds" onchange="showCD(this.value)">
<option value="Bob Dylan">Bob Dylan</option>
<option value="Bee Gees">Bee Gees</option>
<option value="Cat Stevens">Cat Stevens</option>
</select>
</form>
<p>
<div id="txtHint"><b>CD info will be listed here.</b></div>
</p>
</body>
</html>
```

Example Explained

As you can see it is just a simple HTML form with a simple drop down box called "cds".

The paragraph below the form contains a div called "txtHint". The div is used as a placeholder for info retrieved from the web server.

When the user selects data, a function called "showCD" is executed. The execution of the function is triggered by the "onchange" event.

In other words: Each time the user changes the value in the drop down box, the function showCD is called.

The XML File

The XML file is "cd_catalog.xml". This document contains a CD collection.

The JavaScript

This is the JavaScript code stored in the file "selectcd.js":

```
var xmlHttp

function showCD(str)
{
xmlHttp=GetXmlHttpObject()
if (xmlHttp==null)
 {
 alert ("Browser does not support HTTP Request")
 return
 }
var url="getcd.php"
url=url+"?q="+str
url=url+"&sid="+Math.random()
xmlHttp.onreadystatechange=stateChanged
xmlHttp.open("GET",url,true)
xmlHttp.send(null)
}

function stateChanged()
{
 if (xmlHttp.readyState==4 || xmlHttp.readyState=="complete")
 {
 document.getElementById("txtHint").innerHTML=xmlHttp.responseText
 }
}
function GetXmlHttpObject()
{
var xmlHttp=null;
try
 {
 // Firefox, Opera 8.0+, Safari
 xmlHttp=new XMLHttpRequest();
 }
catch (e)
 {
 // Internet Explorer
 try
  {
  xmlHttp=new ActiveXObject("Msxml2.XMLHTTP");
  }
 catch (e)
  {
  xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
 }
return xmlHttp;
}
```

Example Explained

The stateChanged() and GetXmlHttpObject functions are the same as in the last chapter, you can go to the previous page for an explanation of those

The showCD() Function

If an item in the drop down box is selected the function executes the following:

1. Calls on the GetXmlHttpObject function to create an XMLHTTP object

2. Defines the url (filename) to send to the server
3. Adds a parameter (q) to the url with the content of the input field
4. Adds a random number to prevent the server from using a cached file
5. Call stateChanged when a change is triggered
6. Opens the XMLHTTP object with the given url.
7. Sends an HTTP request to the server

The PHP Page

The server paged called by the JavaScript, is a simple PHP file called "getcd.php".

The page is written in PHP using the XML DOM to load the XML document "cd_catalog.xml".

The code runs a query against the XML file and returns the result as HTML:

```
<?php
$q=$_GET["q"];
$xmlDoc = new DOMDocument();
$xmlDoc->load("cd_catalog.xml");
$x=$xmlDoc->getElementsByTagName('ARTIST');
for ($i=0; $i<=$x->length-1; $i++)
{
//Process only element nodes
if ($x->item($i)->nodeType==1)
  {
  if ($x->item($i)->childNodes->item(0)->nodeValue == $q)
    {
    $y=($x->item($i)->parentNode);
    }
  }
}
$cd=($y->childNodes);
for ($i=0;$i<$cd->length;$i++)
{
//Process only element nodes
if ($cd->item($i)->nodeType==1)
  {
  echo($cd->item($i)->nodeName);
  echo(": ");
  echo($cd->item($i)->childNodes->item(0)->nodeValue);
  echo("<br />");
  }
}
?>
```

Example Explained

When the query is sent from the JavaScript to the PHP page the following happens:

1. PHP creates an XML DOM object of the "cd_catalog.xml" file
2. All "artist" elements (nodetypes = 1) are looped through to find a name matching the one sent from the JavaScript.
3. The CD containing the correct artist is found
4. The album information is output and sent to the "txtHint" placeholder

PHP and AJAX MySQL Database Example

AJAX can be used for interactive communication with a database.

AJAX Database Example

In the AJAX example below we will demonstrate how a web page can fetch information from a MySQL database using AJAX technology.

Select a Name in the Box Below

Select a User: Peter Griffin

User info will be listed here.

This example consists of four elements:

- a MySQL database
- a simple HTML form
- a JavaScript
- a PHP page

The Database

The database we will be using in this example looks like this:

id	FirstName	LastName	Age	Hometown	Job
1	Peter	Griffin	41	Quahog	Brewery
2	Lois	Griffin	40	Newport	Piano Teacher
3	Joseph	Swanson	39	Quahog	Police Officer
4	Glenn	Quagmire	41	Quahog	Pilot

The HTML Form

The example above contains a simple HTML form and a link to a JavaScript:

```
<html>
<head>
<script src="selectuser.js"></script>
</head>
<body>
<form>
Select a User:
<select name="users" onchange="showUser(this.value)">
<option value="1">Peter Griffin</option>
<option value="2">Lois Griffin</option>
<option value="3">Glenn Quagmire</option>
<option value="4">Joseph Swanson</option>
</select>
</form>
<p>
<div id="txtHint"><b>User info will be listed here.</b></div>
</p>
</body>
</html>
```

Example Explained - The HTML Form

As you can see it is just a simple HTML form with a drop down box called "users" with names and the "id" from the database as option values.

The paragraph below the form contains a div called "txtHint". The div is used as a placeholder for info retrieved from the web server.

When the user selects data, a function called "showUser()" is executed. The execution of the function is triggered by the "onchange" event.

In other words: Each time the user changes the value in the drop down box, the function showUser() is called.

The JavaScript

This is the JavaScript code stored in the file "selectuser.js":

```
var xmlHttp
function showUser(str)
{
xmlHttp=GetXmlHttpObject()
if (xmlHttp==null)
 {
 alert ("Browser does not support HTTP Request")
 return
 }
var url="getuser.php"
url=url+"?q="+str
url=url+"&sid="+Math.random()
xmlHttp.onreadystatechange=stateChanged
xmlHttp.open("GET",url,true)
xmlHttp.send(null)
}
function stateChanged()
{
if (xmlHttp.readyState==4 || xmlHttp.readyState=="complete")
 {
 document.getElementById("txtHint").innerHTML=xmlHttp.responseText
 }
}
function GetXmlHttpObject()
{
var xmlHttp=null;
try
 {
 // Firefox, Opera 8.0+, Safari
 xmlHttp=new XMLHttpRequest();
 }
catch (e)
 {
 //Internet Explorer
 try
  {
  xmlHttp=new ActiveXObject("Msxml2.XMLHTTP");
  }
 catch (e)
  {
  xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
 }
return xmlHttp;
}
```

Example Explained

The stateChanged() and GetXmlHttpObject functions are the same as in the PHP AJAX Suggest chapter, you can go to there for an explanation of those.

The showUser() Function

If an item in the drop down box is selected the function executes the following:

1. Calls on the GetXmlHttpObject function to create an XMLHTTP object
2. Defines the url (filename) to send to the server
3. Adds a parameter (q) to the url with the content of the dropdown box
4. Adds a random number to prevent the server from using a cached file
5. Call stateChanged when a change is triggered
6. Opens the XMLHTTP object with the given url.
7. Sends an HTTP request to the server

The PHP Page

The server page called by the JavaScript, is a simple PHP file called "getuser.php".

The page is written in PHP and uses a MySQL databse.

The code runs a SQL query against a database and returns the result as an HTML table:

```
<?php
$q=$_GET["q"];

$con = mysql_connect('localhost', 'peter', 'abc123');
if (!$con)
 {
 die('Could not connect: ' . mysql_error());
 }

mysql_select_db("ajax_demo", $con);

$sql="SELECT * FROM user WHERE id = '".$q."'";

$result = mysql_query($sql);

echo "<table border='1'>
<tr>
<th>Firstname</th>
<th>Lastname</th>
<th>Age</th>
<th>Hometown</th>
<th>Job</th>
</tr>";

while($row = mysql_fetch_array($result))
 {
 echo "<tr>";
 echo "<td>" . $row['FirstName'] . "</td>";
 echo "<td>" . $row['LastName'] . "</td>";
 echo "<td>" . $row['Age'] . "</td>";
 echo "<td>" . $row['Hometown'] . "</td>";
 echo "<td>" . $row['Job'] . "</td>";
 echo "</tr>";
 }
echo "</table>";

mysql_close($con);
?>
```

Example Explained

When the query is sent from the JavaScript to the PHP page the following happens:

1. PHP opens a connection to a MySQL server
2. The "user" with the specified name is found
3. A table is created and the data is inserted and sent to the "txtHint" placeholder

PHP and AJAX responseXML Example

AJAX can be used to return database information as XML.

AJAX Database as XML Example

In the AJAX example below we will demonstrate how a web page can fetch information from a MySQL database, convert it to an XML document, and use it to display information in several different places.

This example my seem a lot like the "PHP AJAX Database" example in the last chapter, however there is a big difference: in this example we get the data from the PHP page as XML using the responseXML function.

Receiving the response as an XML document allows us to update this page several places, instead of just receiving a PHP output and displaying it.

In this example we will update several <span> elements with the information we receive from the database.

Select a Name in the Box Below

Select a User: Peter Griffin

This example consists of four elements:

- a MySQL database
- a simple HTML form
- a JavaScript
- a PHP page

The Database

The database we will be using in this example looks like this:

id	FirstName	LastName	Age	Hometown	Job
1	Peter	Griffin	41	Quahog	Brewery
2	Lois	Griffin	40	Newport	Piano Teacher
3	Joseph	Swanson	39	Quahog	Police Officer
4	Glenn	Quagmire	41	Quahog	Pilot

The HTML Form

The example above contains a simple HTML form and a link to a JavaScript:

```
<html>
<head>
<script src="responsexml.js"></script>
</head>
<body>
<form>
Select a User:
<select name="users" onchange="showUser(this.value)">
<option value="1">Peter Griffin</option>
<option value="2">Lois Griffin</option>
<option value="3">Glenn Quagmire</option>
<option value="4">Joseph Swanson</option>
</select>
</form>
<h2><span id="firstname"></span>
&nbsp;<span id="lastname"></span></h2>
<span id="job"></span>
<div style="text-align: right">
<span id="age_text"></span>
<span id="age"></span>
<span id="hometown_text"></span>
<span id="hometown"></span>
</div>
</body>
</html>
```

Example Explained - The HTML Form

- The HTML form is a drop down box called "users" with names and the "id" from the database as option values.
- Below the form there are several different <span> elements which are used to as placeholders for the different values we will retrive.
- When the user selects data, a function called "showUser()" is executed. The execution of the function is triggered by the "onchange" event.

In other words: Each time the user changes the value in the drop down box, the function showUser() is called and outputs the result in the specified <span> elements.

The JavaScript

This is the JavaScript code stored in the file "responsexml.js":

```
var xmlHttp
function showUser(str)
 {
 xmlHttp=GetXmlHttpObject()
 if (xmlHttp==null)
  {
  alert ("Browser does not support HTTP Request")
  return
  }
 var url="responsexml.php"
 url=url+"?q="+str
 url=url+"&sid="+Math.random()
 xmlHttp.onreadystatechange=stateChanged
 xmlHttp.open("GET",url,true)
 xmlHttp.send(null)
 }
function stateChanged()
{
if (xmlHttp.readyState==4 || xmlHttp.readyState=="complete")
{
 xmlDoc=xmlHttp.responseXML;
 document.getElementById("firstname").innerHTML=
 xmlDoc.getElementsByTagName("firstname")[0].childNodes[0].nodeValue;
 document.getElementById("lastname").innerHTML=
 xmlDoc.getElementsByTagName("lastname")[0].childNodes[0].nodeValue;
 document.getElementById("job").innerHTML=
 xmlDoc.getElementsByTagName("job")[0].childNodes[0].nodeValue;
 document.getElementById("age_text").innerHTML="Age: ";
 document.getElementById("age").innerHTML=
 xmlDoc.getElementsByTagName("age")[0].childNodes[0].nodeValue;
 document.getElementById("hometown_text").innerHTML="<br/>From: ";
 document.getElementById("hometown").innerHTML=
 xmlDoc.getElementsByTagName("hometown")[0].childNodes[0].nodeValue;
 }
}
function GetXmlHttpObject()
 {
 var objXMLHttp=null
 if (window.XMLHttpRequest)
  {
  objXMLHttp=new XMLHttpRequest()
  }
 else if (window.ActiveXObject)
  {
  objXMLHttp=new ActiveXObject("Microsoft.XMLHTTP")
  }
 return objXMLHttp
 }
```

Example Explained

The showUser() and GetXmlHttpObject functions are the same as in the PHP AJAX Database chapter, you can go to there for an explanation of those.

The stateChanged() Function

If an item in the drop down box is selected the function executes the following:

1. Defines the "xmlDoc" variable as an xml document using the responseXML function
2. Retrieves data from the xml documents and places them in the correct <span> elements

The PHP Page

The server page called by the JavaScript, is a simple PHP file called "responsexml.php".

The page is written in PHP and uses a MySQL databse.

The code runs a SQL query against a database and returns the result as an XML document:

```
<?php
header('Content-Type: text/xml');
header("Cache-Control: no-cache, must-revalidate");
//A date in the past
header("Expires: Mon, 26 Jul 1997 05:00:00 GMT");
$q=$_GET["q"];
$con = mysql_connect('localhost', 'peter', 'abc123');
if (!$con)
 {
 die('Could not connect: ' . mysql_error());
 }
mysql_select_db("ajax_demo", $con);
$sql="SELECT * FROM user WHERE id = ".$q."";
$result = mysql_query($sql);
echo '<?xml version="1.0" encoding="ISO-8859-1"?>
<person>';
while($row = mysql_fetch_array($result))
 {
 echo "<firstname>" . $row['FirstName'] . "</firstname>";
 echo "<lastname>" . $row['LastName'] . "</lastname>";
 echo "<age>" . $row['Age'] . "</age>";
 echo "<hometown>" . $row['Hometown'] . "</hometown>";
 echo "<job>" . $row['Job'] . "</job>";
 }
echo "</person>";
mysql_close($con);
?>
```

Example Explained

When the query is sent from the JavaScript to the PHP page the following happens:

1. The content-type of the PHP document is set to be "text/xml"
2. The PHP document is set to "no-cache" to prevent caching
3. The $q variable is set to be the data sent from the html page
4. PHP opens a connection to a MySQL server
5. The "user" with the specified id is found
6. The data is outputted as an xml document

PHP and AJAX Live Search

AJAX can be used for a more user friendly and interactive search.

AJAX Live Search

In the AJAX example below we will demonstrate a live search, where the server gets search results while the user types.

Live search has many benefits compared to traditional searching:

- Matching results are shown as you type
- Results narrow as you continue typing
- If results become too narrow, remove characters to see a broader result

---

Search for a W3Schools page in the Box Below

This example consists of four pages:

- a simple HTML form
- a JavaScript
- a PHP page
- an XML document

In this example the results are found in an XML document (links.xml). To make this example small and simple, only eight results are available.

The HTML Form

This is the HTML page. It contains a simple HTML form, style for the form and a link to a JavaScript:

```
<html>
<head>
<script src="livesearch.js"></script>
<style type="text/css">
#livesearch
  {
  margin:0px;
  width:194px;
  }
#txt1
  {
  margin:0px;
  }
</style>
</head>
<body>
<form>
<input type="text" id="txt1" size="30"
onkeyup="showResult(this.value)">
<div id="livesearch"></div>
</form>
</body>
</html>
```

Example Explained - The HTML Form

As you can see, the HTML page above contains a simple HTML form with an input field called "txt1".

The form works like this:

1. An event is triggered when the user presses, and releases a key in the input field
2. When the event is triggered, a function called showResult() is executed.

3. Below the form is a <div> called "livesearch". This is used as a placeholder for the return data of the showResult() function.

The JavaScript

The JavaScript code is stored in "livesearch.js" and linked to the HTML document:

```
var xmlHttp
function showResult(str)
{
if (str.length==0)
 {
 document.getElementById("livesearch").
 innerHTML="";
 document.getElementById("livesearch").
 style.border="0px";
 return
 }
xmlHttp=GetXmlHttpObject()
if (xmlHttp==null)
 {
 alert ("Browser does not support HTTP Request")
 return
 }
var url="livesearch.php"
url=url+"?q="+str
url=url+"&sid="+Math.random()
xmlHttp.onreadystatechange=stateChanged
xmlHttp.open("GET",url,true)
xmlHttp.send(null)
}

function stateChanged()
{
if (xmlHttp.readyState==4 || xmlHttp.readyState=="complete")
 {
 document.getElementById("livesearch").
 innerHTML=xmlHttp.responseText;
 document.getElementById("livesearch").
 style.border="1px solid #A5ACB2";
 }
}
function GetXmlHttpObject()
{
var xmlHttp=null;
try
 {
 // Firefox, Opera 8.0+, Safari
 xmlHttp=new XMLHttpRequest();
 }
catch (e)
 {
 // Internet Explorer
 try
  {
  xmlHttp=new ActiveXObject("Msxml2.XMLHTTP");
  }
 catch (e)
  {
  xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
 }
return xmlHttp;
}
```

Example Explained

The GetXmlHttpObject function is the same as in the PHP AJAX Suggest chapter.

The showResult() Function

This function executes every time a character is entered in the input field.

If there is no input in the text field (str.length == 0) the function sets the return field to empty and removes any border around it.

However, if there is any input in the text field the function executes the following:

1. Defines the url (filename) to send to the server
2. Adds a parameter (q) to the url with the content of the input field
3. Adds a random number to prevent the server from using a cached file
4. Calls on the GetXmlHttpObject function to create an XMLHTTP object, and tells the object to execute a function called stateChanged when a change is triggered
5. Opens the XMLHTTP object with the given url.
6. Sends an HTTP request to the server

The stateChanged() Function

This function executes every time the state of the XMLHTTP object changes.

When the state changes to 4 (or to "complete"), the content of the txtHint placeholder is filled with the response text, and a border is set around the return field.

The PHP Page

The server page called by the JavaScript code is a PHP file called "livesearch.php".

The code in the "livesearch.php" checks the XML document "links.xml". This document contains titles and URL's of some pages on W3Schools.com.

The code searches the XML file for titles matching the search string and returns the result as HTML:

```
<?php
$xmlDoc = new DOMDocument();
$xmlDoc->load("links.xml");
$x=$xmlDoc->getElementsByTagName('link');
//get the q parameter from URL
$q=$_GET["q"];
//lookup all links from the xml file if length of q>0
if (strlen($q) > 0)
{
$hint="";
for($i=0; $i<($x->length); $i++)
 {
 $y=$x->item($i)->getElementsByTagName('title');
 $z=$x->item($i)->getElementsByTagName('url');
 if ($y->item(0)->nodeType==1)
```

```
  {
  //find a link matching the search text
  if (stristr($y->item(0)->childNodes->item(0)->nodeValue,$q))
   {
   if ($hint=="")
    {
    $hint="<a href='" .
    $z->item(0)->childNodes->item(0)->nodeValue .
    "' target='_blank'>" .
    $y->item(0)->childNodes->item(0)->nodeValue . "</a>";
    }
   else
    {
    $hint=$hint . "<br /><a href='" .
    $z->item(0)->childNodes->item(0)->nodeValue .
    "' target='_blank'>" .
    $y->item(0)->childNodes->item(0)->nodeValue . "</a>";
    }
   }
  }
 }
}
// Set output to "no suggestion" if no hint were found
// or to the correct values
if ($hint == "")
 {
 $response="no suggestion";
 }
else
 {
 $response=$hint;
 }
//output the response
echo $response;
?>
```

If there is any text sent from the JavaScript (strlen($q) > 0) the following happens:

1. PHP creates an XML DOM object of the "links.xml" file
2. All "title" elements (nodetypes = 1) are looped through to find a name matching the one sent from the JavaScript
3. The link containing the correct title is found and set as the "$response" variable. If more than one match is found, all matches are added to the variable
4. If no matches are found the $response variable is set to "no suggestion"
5. The $result variable is output and sent to the "livesearch" placeholder

PHP and AJAX RSS Reader

An RSS Reader is used to read RSS Feeds

RSS allows fast browsing for news and updates

AJAX RSS Reader

In the AJAX example below we will demonstrate an RSS reader where the content from the RSS is loaded into the webpage without refreshing.

Select an RSS News Feed in the Box Below

Select an RSS-Feed: Google News ▾
RSS Feed will be listed here.

This example consists of three pages:

- a simple HTML form
- a JavaScript
- a PHP page.

The HTML Form

This is the HTML page. It contains a simple HTML form and a link to a JavaScript:

```
<html>
<head>
<script type="text/javascript" src="getrss.js"></script>
</head>
<body>
<form>
Select an RSS-Feed:
<select onchange="showRSS(this.value)">
<option value="Google">Google News</option>
<option value="MSNBC">MSNBC News</option>
</select>
</form>
<p><div id="rssOutput">
<b>RSS Feed will be listed here.</b></div></p>
</body>
</html>
```

Example Explained - The HTML Form

As you can see, the HTML page above contains a simple HTML form with a drop-down box.

The form works like this:

1. An event is triggered when the user selects an option in the drop down box
2. When the event is triggered, a function called showRSS() is executed.
3. Below the form is a <div> called "rssOutput". This is used as a placeholder for the return data of the showRSS() function.

The JavaScript

The JavaScript code is stored in "getrss.js" and linked to the HTML document:

```
var xmlHttp
function showRSS(str)
 {
 xmlHttp=GetXmlHttpObject()
 if (xmlHttp==null)
  {
  alert ("Browser does not support HTTP Request")
  return
  }
 var url="getrss.php"
 url=url+"?q="+str
 url=url+"&sid="+Math.random()
 xmlHttp.onreadystatechange=stateChanged
 xmlHttp.open("GET",url,true)
 xmlHttp.send(null)
 }

function stateChanged()
 {
 if (xmlHttp.readyState==4 || xmlHttp.readyState=="complete")
  {
  document.getElementById("rssOutput")
  .innerHTML=xmlHttp.responseText
  }
 }
function GetXmlHttpObject()
{
var xmlHttp=null;
try
 {
 // Firefox, Opera 8.0+, Safari
 xmlHttp=new XMLHttpRequest();
 }
catch (e)
 {
 // Internet Explorer
 try
  {
  xmlHttp=new ActiveXObject("Msxml2.XMLHTTP");
  }
 catch (e)
  {
  xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
 }
return xmlHttp;
}
```

Example Explained

The stateChanged() and GetXmlHttpObject functions are the same as in the PHP AJAX Suggest chapter.

The showRSS() Function

Every time an option is selected in the input field this function executes the following:

1. Defines the url (filename) to send to the server
2. Adds a parameter (q) to the url with the selected option from the drop down box
3. Adds a random number to prevent the server from using a cached file
4. Calls on the GetXmlHttpObject function to create an XMLHTTP object, and tells the object to execute a function called stateChanged when a change is triggered
5. Opens the XMLHTTP object with the given url.
6. Sends an HTTP request to the server

The PHP Page

The server page called by the JavaScript code is a PHP file called "getrss.php":

```
<?php
//get the q parameter from URL
$q=$_GET["q"];
//find out which feed was selected
if($q=="Google")
 {
 $xml=("http://news.google.com/news?ned=us&topic=h&output=rss");
 }
elseif($q=="MSNBC")
 {
 $xml=("http://rss.msnbc.msn.com/id/3032091/device/rss/rss.xml");
 }
$xmlDoc = new DOMDocument();
$xmlDoc->load($xml);
//get elements from "<channel>"
$channel=$xmlDoc->getElementsByTagName('channel')->item(0);
$channel_title = $channel->getElementsByTagName('title')
->item(0)->childNodes->item(0)->nodeValue;
$channel_link = $channel->getElementsByTagName('link')
->item(0)->childNodes->item(0)->nodeValue;
$channel_desc = $channel->getElementsByTagName('description')
->item(0)->childNodes->item(0)->nodeValue;
//output elements from "<channel>"
echo("<p><a href='" . $channel_link
 . "'>" . $channel_title . "</a>");
echo("<br />");
echo($channel_desc . "</p>");
//get and output "<item>" elements
$x=$xmlDoc->getElementsByTagName('item');
for ($i=0; $i<=2; $i++)
 {
 $item_title=$x->item($i)->getElementsByTagName('title')
 ->item(0)->childNodes->item(0)->nodeValue;
 $item_link=$x->item($i)->getElementsByTagName('link')
 ->item(0)->childNodes->item(0)->nodeValue;
 $item_desc=$x->item($i)->getElementsByTagName('description')
 ->item(0)->childNodes->item(0)->nodeValue;
 echo ("<p><a href='" . $item_link
 . "'>" . $item_title . "</a>");
 echo ("<br />");
 echo ($item_desc . "</p>");
 }
?>
```

Example Explained - The PHP Page

When an option is sent from the JavaScript the following happens:

1. PHP finds out which RSS feed was selected
2. An XML DOM object is created for the selected RSS feed
3. The elements from the RSS channel are found and outputted
4. The three first elements from the RSS items are looped through and outputted

PHP and AJAX Poll

AJAX Suggest

In the AJAX example below we will demonstrate a poll where the web page can get the result without reloading.

Do you like PHP and AJAX so far?

Yes: ○

No: ○

This example consists of four pages:

- a simple HTML form
- a JavaScript
- a PHP page
- a text file to store the results

The HTML Form

This is the HTML page. It contains a simple HTML form and a link to a JavaScript:

```
<html>
<head>
<script src="poll.js"></script>
</head>
<body>
<div id="poll">
<h2>Do you like PHP and AJAX so far?</h2>
<form>
Yes:
<input type="radio" name="vote"
value="0" onclick="getVote(this.value)">
<br>No:
<input type="radio" name="vote"
value="1" onclick="getVote(this.value)">
</form>
</div>
</body>
</html>
```

Example Explained - The HTML Form

As you can see, the HTML page above contains a simple HTML form inside a "<div>" with two radio buttons.

The form works like this:

1. An event is triggered when the user selects the "yes" or "no" option
2. When the event is triggered, a function called getVote() is executed.
3. Around the form is a <div> called "poll". When the data is returned from the getVote() function, the return data will replace the form.

The Text File

The text file (poll_result.txt) is where we store the data from the poll.

It is stored like this:

```
0||0
```

The first number represents the "Yes" votes, the second number represents the "No" votes.

Note: Remember to allow your web server to edit the text file. Do **NOT** give everyone access, just the web server (PHP).

The JavaScript

The JavaScript code is stored in "poll.js" and linked to in the HTML document:

```
var xmlHttp

function getVote(int)
{
xmlHttp=GetXmlHttpObject()
if (xmlHttp==null)
 {
 alert ("Browser does not support HTTP Request")
 return
 }
var url="poll_vote.php"
url=url+"?vote="+int
url=url+"&sid="+Math.random()
xmlHttp.onreadystatechange=stateChanged
xmlHttp.open("GET",url,true)
xmlHttp.send(null)
}

function stateChanged()
{
 if (xmlHttp.readyState==4 || xmlHttp.readyState=="complete")
 {
 document.getElementById("poll").
 innerHTML=xmlHttp.responseText;
 }
}

function GetXmlHttpObject()
{
var objXMLHttp=null
if (window.XMLHttpRequest)
 {
 objXMLHttp=new XMLHttpRequest()
 }
else if (window.ActiveXObject)
 {
 objXMLHttp=new ActiveXObject("Microsoft.XMLHTTP")
 }
return objXMLHttp
}
```

Example Explained

The stateChanged() and GetXmlHttpObject functions are the same as in the PHP AJAX Suggest chapter.

The getVote() Function

This function executes when "yes" or "no" is selected in the HTML form.

1. Defines the url (filename) to send to the server
2. Adds a parameter (vote) to the url with the content of the input field
3. Adds a random number to prevent the server from using a cached file
4. Calls on the GetXmlHttpObject function to create an XMLHTTP object, and tells the object to execute a function called stateChanged when a change is triggered
5. Opens the XMLHTTP object with the given url.
6. Sends an HTTP request to the server

The PHP Page

The server page called by the JavaScript code is a simple PHP file called "poll_vote.php".

```
<?php
$vote = $_REQUEST['vote'];
//get content of textfile
$filename = "poll_result.txt";
$content = file($filename);
//put content in array
$array = explode("||", $content[0]);
$yes = $array[0];
$no = $array[1];
if ($vote == 0)
 {
 $yes = $yes + 1;
 }
if ($vote == 1)
 {
 $no = $no + 1;
 }
//insert votes to txt file
$insertvote = $yes."||".$no;
$fp = fopen($filename,"w");
fputs($fp,$insertvote);
fclose($fp);
?>
<h2>Result:</h2>
<table>
<tr>
<td>Yes:</td>
<td>
<img src="poll.gif"
width='<?php echo(100*round($yes/($no+$yes),2)); ?>'
height='20'>
<?php echo(100*round($yes/($no+$yes),2)); ?>%
</td>
</tr>
<tr>
<td>No:</td>
<td>
<img src="poll.gif"
width='<?php echo(100*round($no/($no+$yes),2)); ?>'
height='20'>
<?php echo(100*round($no/($no+$yes),2)); ?>%
</td>
</tr>
</table>
```

The selected value is sent from the JavaScript and the following happens:

1. Get the content of the "poll_result.txt" file

2. Put the content of the file in variables and add one to the selected variable
3. Write the result to the "poll_result.txt" file
4. Output a graphical representation of the poll result

PHP Array Functions

PHP Array Introduction

The array functions allow you to manipulate arrays.

PHP supports both simple and multi-dimensional arrays. There are also specific functions for populating arrays from database queries.

Installation

The array functions are part of the PHP core. There is no installation needed to use these functions.

PHP Array Functions

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

Function	Description	PHP
array()	Creates an array	3
array_change_key_case()	Returns an array with all keys in lowercase or uppercase	4
array_chunk()	Splits an array into chunks of arrays	4
array_combine()	Creates an array by using one array for keys and another for its values	5
array_count_values()	Returns an array with the number of occurrences for each value	4
array_diff()	Compares array values, and returns the differences	4
array_diff_assoc()	Compares array keys and values, and returns the differences	4
array_diff_key()	Compares array keys, and returns the differences	5
array_diff_uassoc()	Compares array keys and values, with an additional user-made function check, and returns the differences	5
array_diff_ukey()	Compares array keys, with an additional user-made function check, and returns the differences	5
array_fill()	Fills an array with values	4
array_filter()	Filters elements of an array using a user-made function	4
array_flip()	Exchanges all keys with their associated values in an array	4
array_intersect()	Compares array values, and returns the matches	4
array_intersect_assoc()	Compares array keys and values, and returns the matches	4
array_intersect_key()	Compares array keys, and returns the matches	5
array_intersect_uassoc()	Compares array keys and values, with an additional user-made function check, and returns the matches	5
array_intersect_ukey()	Compares array keys, with an additional user-made function check, and returns the matches	5
array_key_exists()	Checks if the specified key exists in the array	4
array_keys()	Returns all the keys of an array	4
array_map()	Sends each value of an array to a user-made function, which returns new values	4

array_merge()	Merges one or more arrays into one array	4
array_merge_recursive()	Merges one or more arrays into one array	4
array_multisort()	Sorts multiple or multi-dimensional arrays	4
array_pad()	Inserts a specified number of items, with a specified value, to an array	4
array_pop()	Deletes the last element of an array	4
array_product()	Calculates the product of the values in an array	5
array_push()	Inserts one or more elements to the end of an array	4
array_rand()	Returns one or more random keys from an array	4
array_reduce()	Returns an array as a string, using a user-defined function	4
array_reverse()	Returns an array in the reverse order	4
array_search()	Searches an array for a given value and returns the key	4
array_shift()	Removes the first element from an array, and returns the value of the removed element	4
array_slice()	Returns selected parts of an array	4
array_splice()	Removes and replaces specified elements of an array	4
array_sum()	Returns the sum of the values in an array	4
array_udiff()	Compares array values in a user-made function and returns an array	5
array_udiff_assoc()	Compares array keys, and compares array values in a user-made function, and returns an array	5
array_udiff_uassoc()	Compares array keys and array values in user-made functions, and returns an array	5
array_uintersect()	Compares array values in a user-made function and returns an array	5
array_uintersect_assoc()	Compares array keys, and compares array values in a user-made function, and returns an array	5
array_uintersect_uassoc()	Compares array keys and array values in user-made functions, and returns an array	5
array_unique()	Removes duplicate values from an array	4
array_unshift()	Adds one or more elements to the beginning of an array	4
array_values()	Returns all the values of an array	4
array_walk()	Applies a user function to every member of an array	3
array_walk_recursive()	Applies a user function recursively to every member of an array	5
arsort()	Sorts an array in reverse order and maintain index association	3
asort()	Sorts an array and maintain index association	3
compact()	Create array containing variables and their values	4
count()	Counts elements in an array, or properties in an object	3
current()	Returns the current element in an array	3
each()	Returns the current key and value pair from an array	3
end()	Sets the internal pointer of an array to its last element	3
extract()	Imports variables into the current symbol table from an array	3
in_array()	Checks if a specified value exists in an array	4
key()	Fetches a key from an array	3
krsort()	Sorts an array by key in reverse order	3
ksort()	Sorts an array by key	3
list()	Assigns variables as if they were an array	3
natcasesort()	Sorts an array using a case insensitive "natural order" algorithm	4
natsort()	Sorts an array using a "natural order" algorithm	4
next()	Advance the internal array pointer of an array	3
pos()	Alias of current()	3
prev()	Rewinds the internal array pointer	3
range()	Creates an array containing a range of elements	3
reset()	Sets the internal pointer of an array to its first element	3
rsort()	Sorts an array in reverse order	3
shuffle()	Shuffles an array	3

sizeof()	Alias of count()	3
sort()	Sorts an array	3
uasort()	Sorts an array with a user-defined function and maintain index association	3
uksort()	Sorts an array by keys using a user-defined function	3
usort()	Sorts an array by values using a user-defined function	3

PHP Array Constants

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

Constant	Description	PHP
CASE_LOWER	Used with array_change_key_case() to convert array keys to lower case	
CASE_UPPER	Used with array_change_key_case() to convert array keys to upper case	
SORT_ASC	Used with array_multisort() to sort in ascending order	
SORT_DESC	Used with array_multisort() to sort in descending order	
SORT_REGULAR	Used to compare items normally	
SORT_NUMERIC	Used to compare items numerically	
SORT_STRING	Used to compare items as strings	
SORT_LOCALE_STRING	Used to compare items as strings, based on the current locale	4
COUNT_NORMAL		
COUNT_RECURSIVE		
EXTR_OVERWRITE		
EXTR_SKIP		
EXTR_PREFIX_SAME		
EXTR_PREFIX_ALL		
EXTR_PREFIX_INVALID		
EXTR_PREFIX_IF_EXISTS		
EXTR_IF_EXISTS		
EXTR_REFS		

PHP Calendar Functions

PHP Calendar Introduction

The calendar functions are useful when working with different calendar formats. The standard it is based on is the Julian day count (Julian day count is a count of days starting from January 1, 4713 B.C.). Note that the Julian day count is not the same as the Julian calendar!

Note: To convert between calendar formats, you must first convert to Julian day count, then to the calendar format.

Installation

The windows version of PHP has built-in support for the calendar extension. So, the calendar functions will work automatically.

However, if you are running the Linux version of PHP, you will have to compile PHP with *--enable-calendar* to get the calendar functions to work.

PHP Calendar Functions

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

Function	Description	PHP
cal_days_in_month()	Returns the number of days in a month for a specified year and calendar	4
cal_from_jd()	Converts a Julian day count into a date of a specified calendar	4
cal_info()	Returns information about a given calendar	4
cal_to_jd()	Converts a date to Julian day count	4
easter_date()	Returns the Unix timestamp for midnight on Easter of a specified year	3
easter_days()	Returns the number of days after March 21, on which Easter falls for a specified year	3
FrenchToJD()	Converts a French Republican date to a Julian day count	3
GregorianToJD()	Converts a Gregorian date to a Julian day count	3
JDDayOfWeek()	Returns the day of a week	3
JDMonthName()	Returns a month name	3
JDToFrench()	Converts a Julian day count to a French Republican date	3
JDToGregorian()	Converts a Julian day count to a Gregorian date	3
jdtojewish()	Converts a Julian day count to a Jewish date	3
JDToJulian()	Converts a Julian day count to a Julian date	3
jdtounix()	Converts a Julian day count to a Unix timestamp	4
JewishToJD()	Converts a Jewish date to a Julian day count	3
JulianToJD()	Converts a Julian date to a Julian day count	3
unixtojd()	Converts a Unix timestamp to a Julian day count	4

PHP Calendar Constants

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

Constant	Description	PHP
CAL_GREGORIAN	Gregorian calendar	3
CAL_JULIAN	Julian calendar	3
CAL_JEWISH	Jewish calendar	3
CAL_FRENCH	French Republican calendar	3
CAL_NUM_CALS		3
CAL_DOW_DAYNO		3
CAL_DOW_SHORT		3
CAL_DOW_LONG		3
CAL_MONTH_GREGORIAN_SHORT		3
CAL_MONTH_GREGORIAN_LONG		3
CAL_MONTH_JULIAN_SHORT		3
CAL_MONTH_JULIAN_LONG		3
CAL_MONTH_JEWISH		3
CAL_MONTH_FRENCH		3
CAL_EASTER_DEFAULT		4
CAL_EASTER_DEFAULT		4
CAL_EASTER_ROMAN		4
CAL_EASTER_ALWAYS_GREGORIAN		4
CAL_EASTER_ALWAYS_JULIAN		4
CAL_JEWISH_ADD_ALAFIM_GERESH		5
CAL_JEWISH_ADD_ALAFIM		5
CAL_JEWISH_ADD_GERESHAYIM		5

PHP Date / Time Functions

PHP Date / Time Introduction

The date/time functions allow you to extract and format the date and time on the server.

Note: These functions depend on the locale settings of the server!

Installation

The date/time functions are part of the PHP core. There is no installation needed to use these functions.

Runtime Configuration

The behavior of the date/time functions is affected by settings in php.ini.

Date/Time configuration options:

Name	**Default**	**Description**	**Changeable**
date.default_latitude	"31.7667"	Specifies the default latitude (available since PHP 5). This option is used by date_sunrise() and date_sunset()	PHP_INI_ALL
date.default_longitude	"35.2333"	Specifies the default longitude (available since PHP 5). This option is used by date_sunrise() and date_sunset()	PHP_INI_ALL
date.sunrise_zenith	"90.83"	Specifies the default sunrise zenith (available since PHP 5). This option is used by date_sunrise() and date_sunset()	PHP_INI_ALL
date.sunset_zenith	"90.83"	Specifies the default sunset zenith (available since PHP 5). This option is used by date_sunrise() and date_sunset()	PHP_INI_ALL
date.timezone	""	Specifies the default timezone (available since PHP 5.1)	PHP_INI_ALL

PHP Date / Time Functions

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

Function	**Description**	**PHP**
checkdate()	Validates a Gregorian date	3
date_default_timezone_get()	Returns the default time zone	5
date_default_timezone_set()	Sets the default time zone	5
date_sunrise()	Returns the time of sunrise for a given day / location	5
date_sunset()	Returns the time of sunset for a given day / location	5
date()	Formats a local time/date	3
getdate()	Returns an array that contains date and time information for a Unix timestamp	3
gettimeofday()	Returns an array that contains current time information	3
gmdate()	Formats a GMT/UTC date/time	3
gmmktime()	Returns the Unix timestamp for a GMT date	3
gmstrftime()	Formats a GMT/UTC time/date according to locale settings	3

idate()	Formats a local time/date as integer	5
localtime()	Returns an array that contains the time components of a Unix timestamp	4
microtime()	Returns the microseconds for the current time	3
mktime()	Returns the Unix timestamp for a date	3
strftime()	Formats a local time/date according to locale settings	3
strptime()	Parses a time/date generated with strftime()	5
strtotime()	Parses an English textual date or time into a Unix timestamp	3
time()	Returns the current time as a Unix timestamp	3

PHP Date / Time Constants

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

Constant	Description	PHP
DATE_ATOM	Atom (example: 2005-08-15T16:13:03+0000)	
DATE_COOKIE	HTTP Cookies (example: Sun, 14 Aug 2005 16:13:03 UTC)	
DATE_ISO8601	ISO-8601 (example: 2005-08-14T16:13:03+0000)	
DATE_RFC822	RFC 822 (example: Sun, 14 Aug 2005 16:13:03 UTC)	
DATE_RFC850	RFC 850 (example: Sunday, 14-Aug-05 16:13:03 UTC)	
DATE_RFC1036	RFC 1036 (example: Sunday, 14-Aug-05 16:13:03 UTC)	
DATE_RFC1123	RFC 1123 (example: Sun, 14 Aug 2005 16:13:03 UTC)	
DATE_RFC2822	RFC 2822 (Sun, 14 Aug 2005 16:13:03 +0000)	
DATE_RSS	RSS (Sun, 14 Aug 2005 16:13:03 UTC)	
DATE_W3C	World Wide Web Consortium (example: 2005-08-14T16:13:03+0000)	

PHP Directory Functions

PHP Directory Introduction

The directory functions allow you to retrieve information about directories and their contents.

Installation

The directory functions are part of the PHP core. There is no installation needed to use these functions.

PHP Directory Functions

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

Function	Description	PHP
chdir()	Changes the current directory	3
chroot()	Changes the root directory of the current process	4
dir()	Opens a directory handle and returns an object	3
closedir()	Closes a directory handle	3
getcwd()	Returns the current directory	4
opendir()	Opens a directory handle	3

readdir()	Returns an entry from a directory handle	3
rewinddir()	Resets a directory handle	3
scandir()	Lists files and directories inside a specified path	5

PHP Directory Constants

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

Constant	Description	PHP
DIRECTORY_SEPARATOR		3
PATH_SEPARATOR		4

PHP Error and Logging Functions

PHP Error and Logging Introduction

The error and logging functions allows error handling and logging.

The error functions allow users to define error handling rules, and modify the way the errors can be logged.

The logging functions allow users to log applications and send log messages to email, system logs or other machines.

Installation

The error and logging functions are part of the PHP core. There is no installation needed to use these functions.

PHP Error and Logging Functions

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

Function	Description	PHP
debug_backtrace()	Generates a backtrace	4
debug_print_backtrace()	Prints a backtrace	5
error_get_last()	Gets the last error occurred	5
error_log()	Sends an error to the server error-log, to a file or to a remote destination	4
error_reporting()	Specifies which errors are reported	4
restore_error_handler()	Restores the previous error handler	4
restore_exception_handler()	Restores the previous exception handler	5
set_error_handler()	Sets a user-defined function to handle errors	4
set_exception_handler()	Sets a user-defined function to handle exceptions	5
trigger_error()	Creates a user-defined error message	4
user_error()	Alias of trigger_error()	4

PHP Error and Logging Constants

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

Value	Constant	Description	PHP
1	E_ERROR	Fatal run-time errors. Errors that cannot be recovered from. Execution of the script is halted	
2	E_WARNING	Non-fatal run-time errors. Execution of the script is not halted	
4	E_PARSE	Compile-time parse errors. Parse errors should only be generated by the parser	
8	E_NOTICE	Run-time notices. The script found something that might be an error, but could also happen when running a script normally	
16	E_CORE_ERROR	Fatal errors at PHP startup. This is like an E_ERROR in the PHP core	4
32	E_CORE_WARNING	Non-fatal errors at PHP startup. This is like an E_WARNING in the PHP core	4
64	E_COMPILE_ERROR	Fatal compile-time errors. This is like an E_ERROR generated by the Zend Scripting Engine	4
128	E_COMPILE_WARNING	Non-fatal compile-time errors. This is like an E_WARNING generated by the Zend Scripting Engine	4
256	E_USER_ERROR	Fatal user-generated error. This is like an E_ERROR set by the programmer using the PHP function trigger_error()	4
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()	4
1024	E_USER_NOTICE	User-generated notice. This is like an E_NOTICE set by the programmer using the PHP function trigger_error()	4
2048	E_STRICT	Run-time notices. PHP suggest changes to your code to help interoperability and compatibility of the code	5
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())	5
8191	E_ALL	All errors and warnings, except of level E_STRICT	5

PHP Filter Functions

PHP Filter Introduction

This PHP filters is used to validate and filter data coming from insecure sources, like user input.

Installation

The filter functions are part of the PHP core. There is no installation needed to use these functions.

PHP Filter Functions

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

Function	Description	PHP

filter_has_var()	Checks if a variable of a specified input type exist	5
filter_id()	Returns the ID number of a specified filter	5
filter_input()	Get input from outside the script and filter it	5
filter_input_array()	Get multiple inputs from outside the script and filters them	5
filter_list()	Returns an array of all supported filters	5
filter_var_array()	Get multiple variables and filter them	5
filter_var()	Get a variable and filter it	5

PHP Filters

ID Name	Description
FILTER_CALLBACK	Call a user-defined function to filter data
FILTER_SANITIZE_STRING	Strip tags, optionally strip or encode special characters
FILTER_SANITIZE_STRIPPED	Alias of "string" filter
FILTER_SANITIZE_ENCODED	URL-encode string, optionally strip or encode special characters
FILTER_SANITIZE_SPECIAL_CHARS	HTML-escape '"<>& and characters with ASCII value less than 32
FILTER_SANITIZE_EMAIL	Remove all characters, except letters, digits and !#$%&'*+-/=?^_`{|}~@.[]
FILTER_SANITIZE_URL	Remove all characters, except letters, digits and $-_.+!*'(),{}|\\^~[]`<>#%";/?:@&=
FILTER_SANITIZE_NUMBER_INT	Remove all characters, except digits and +-
FILTER_SANITIZE_NUMBER_FLOAT	Remove all characters, except digits, +- and optionally .,eE
FILTER_SANITIZE_MAGIC_QUOTES	Apply addslashes()
FILTER_UNSAFE_RAW	Do nothing, optionally strip or encode special characters
FILTER_VALIDATE_INT	Validate value as integer, optionally from the specified range
FILTER_VALIDATE_BOOLEAN	Return TRUE for "1", "true", "on" and "yes", FALSE for "0", "false", "off", "no", and "", NULL otherwise
FILTER_VALIDATE_FLOAT	Validate value as float
FILTER_VALIDATE_REGEXP	Validate value against regexp, a Perl-compatible regular expression
FILTER_VALIDATE_URL	Validate value as URL, optionally with required components
FILTER_VALIDATE_EMAIL	Validate value as e-mail
FILTER_VALIDATE_IP	Validate value as IP address, optionally only IPv4 or IPv6 or not from private or reserved ranges

PHP FTP Functions

PHP FTP Introduction

The FTP functions give client access to file servers through the File Transfer Protocol (FTP).

The FTP functions are used to open, login and close connections, as well as upload, download, rename, delete, and get information on files from file servers. Not all of the FTP functions will work with every server or return the same results. The FTP functions became available with PHP 3.

These functions are meant for detailed access to an FTP server. If you only wish to read from or write to a file on an FTP server, consider using the ftp:// wrapper with the Filesystem functions.

Installation

The windows version of PHP has built-in support for the FTP extension. So, the FTP functions will work automatically.

However, if you are running the Linux version of PHP, you will have to compile PHP with *--enable-ftp* (PHP 4+) or *--with-ftp* (PHP 3) to get the FTP functions to work.

PHP FTP Functions

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

Function	Description	PHP
ftp_alloc()	Allocates space for a file to be uploaded to the FTP server	5
ftp_cdup()	Changes the current directory to the parent directory on the FTP server	3
ftp_chdir()	Changes the current directory on the FTP server	3
ftp_chmod()	Sets permissions on a file via FTP	5
ftp_close()	Closes an FTP connection	4
ftp_connect()	Opens an FTP connection	3
ftp_delete()	Deletes a file on the FTP server	3
ftp_exec()	Executes a program/command on the FTP server	4
ftp_fget()	Downloads a file from the FTP server and saves it to an open file	3
ftp_fput()	Uploads from an open file and saves it to a file on the FTP server	3
ftp_get_option()	Returns runtime behaviors of the FTP connection	4
ftp_get()	Downloads a file from the FTP server	3
ftp_login()	Logs on to an FTP connection	3
ftp_mdtm()	Returns the last modified time of a specified file	3
ftp_mkdir()	Creates a new directory on the FTP server	3
ftp_nb_continue()	Continues retrieving/sending a file (non-blocking)	4
ftp_nb_fget()	Downloads a file from the FTP server and saves it to an open file (non-blocking)	4
ftp_nb_fput()	Uploads from an open file and saves it to a file on the FTP server (non-blocking)	4
ftp_nb_get()	Downloads a file from the FTP server (non-blocking)	4
ftp_nb_put()	Uploads a file to the FTP server (non-blocking)	4
ftp_nlist()	Lists the files in a specified directory on the FTP server	3
ftp_pasv()	Turns passive mode on or off	3
ftp_put()	Uploads a file to the FTP server	3
ftp_pwd()	Returns the current directory name	3
ftp_quit()	Alias of ftp_close()	3
ftp_raw()	Sends a raw command to the FTP server	5
ftp_rawlist()	Returns a detailed list of files in the specified directory	3
ftp_rename()	Renames a file or directory on the FTP server	3
ftp_rmdir()	Removes a directory on the FTP server	3
ftp_set_option()	Sets runtime options for the FTP connection	4
ftp_site()	Sends a SITE command to the server	3
ftp_size()	Returns the size of the specified file	3
ftp_ssl_connect()	Opens a secure SSL-FTP connection	4
ftp_systype()	Returns the system type identifier of the FTP server	3

PHP FTP Constants

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

Constant	Description	PHP
FTP_ASCII		3
FTP_TEXT		3
FTP_BINARY		3
FTP_IMAGE		3
FTP_TIMEOUT_SEC		3
FTP_AUTOSEEK		4
FTP_AUTORESUME	Determine resume position and start position for get and put requests automatically	4
FTP_FAILED	Asynchronous transfer has failed	4
FTP_FINISHED	Asynchronous transfer has finished	4
FTP_MOREDATA	Asynchronous transfer is still active	4

PHP HTTP Functions

PHP HTTP Introduction

The HTTP functions let you manipulate information sent to the browser by the Web server, before any other output has been sent.

Installation

The directory functions are part of the PHP core. There is no installation needed to use these functions.

PHP HTTP Functions

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

Function	Description	PHP
header()	Sends a raw HTTP header to a client	3
headers_list()	Returns a list of response headers sent (or ready to send)	5
headers_sent()	Checks if / where the HTTP headers have been sent	3
setcookie()	Sends an HTTP cookie to a client	3
setrawcookie()	Sends an HTTP cookie without URL encoding the cookie value	5

PHP HTTP Constants

None.

PHP libxml Functions

PHP libxml Introduction

The libxml functions and constants are used together with SimpleXML, XSLT and DOM functions.

Installation

These functions require the libxml package. Download at xmlsoft.org

PHP libxml Functions

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

Function	Description	PHP
libxml_clear_errors()	Clear libxml error buffer	5
libxml_get_errors()	Retrieve array of errors	5
libxml_get_last_error()	Retrieve last error from libxml	5
libxml_set_streams_context()	Set the streams context for the next libxml document load or write	5
libxml_use_internal_errors()	Disable libxml errors and allow user to fetch error information as needed	5

PHP libxml Constants

Function	Description	PHP
LIBXML_COMPACT	Set small nodes allocation optimization. This may improve the application performance	5
LIBXML_DTDATTR	Set default DTD attributes	5
LIBXML_DTDLOAD	Load external subset	5
LIBXML_DTDVALID	Validate with the DTD	5
LIBXML_NOBLANKS	Remove blank nodes	5
LIBXML_NOCDATA	Set CDATA as text nodes	5
LIBXML_NOEMPTYTAG	Change empty tags (e.g. <br/> to <br></br>), only available in the DOMDocument->save() and DOMDocument->saveXML() functions	5
LIBXML_NOENT	Substitute entities	5
LIBXML_NOERROR	Do not show error reports	5
LIBXML_NONET	Stop network access while loading documents	5
LIBXML_NOWARNING	Do not show warning reports	5
LIBXML_NOXMLDECL	Drop the XML declaration when saving a document	5
LIBXML_NSCLEAN	Remove excess namespace declarations	5
LIBXML_XINCLUDE	Use XInclude substitution	5
LIBXML_ERR_ERROR	Get recoverable errors	5
LIBXML_ERR_FATAL	Get fatal errors	5
LIBXML_ERR_NONE	Get no errors	5
LIBXML_ERR_WARNING	Get simple warnings	5
LIBXML_VERSION	Get libxml version (e.g. 20605 or 20617)	5
LIBXML_DOTTED_VERSION	Get dotted libxml version (e.g. 2.6.5 or 2.6.17)	5

PHP Mail Functions

PHP Mail Introduction

The mail() function allows you to send emails directly from a script.

Requirements

For the mail functions to be available, PHP requires an installed and working email system. The program to be used is defined by the configuration settings in the php.ini file.

Installation

The mail functions are part of the PHP core. There is no installation needed to use these functions.

Runtime Configuration

The behavior of the mail functions is affected by settings in the php.ini file.

Mail configuration options:

Name	Default	Description	Changeable
SMTP	"localhost"	Windows only: The DNS name or IP address of the SMTP server	PHP_INI_ALL
smtp_port	"25"	Windows only: The SMTP port number. Available since PHP 4.3	PHP_INI_ALL
sendmail_from	NULL	Windows only: Specifies the "from" address to be used in email sent from PHP	PHP_INI_ALL
sendmail_path	NULL	Unix systems only: Specifies where the sendmail program can be found (usually /usr/sbin/sendmail or /usr/lib/sendmail)	PHP_INI_SYSTEM

PHP Mail Functions

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

Function	Description	PHP
ezmlm_hash()	Calculates the hash value needed by the EZMLM mailing list system	3
mail()	Allows you to send emails directly from a script	3

PHP Mail Constants

None.

PHP Math Functions

PHP Math Introduction

The math functions can handle values within the range of integer and float types.

Installation

The math functions are part of the PHP core. There is no installation needed to use these functions.

PHP Math Functions

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

Function	**Description**	**PHP**
abs()	Returns the absolute value of a number	3
acos()	Returns the arccosine of a number	3
acosh()	Returns the inverse hyperbolic cosine of a number	4
asin()	Returns the arcsine of a number	3
asinh()	Returns the inverse hyperbolic sine of a number	4
atan()	Returns the arctangent of a number as a numeric value between -PI/2 and PI/2 radians	3
atan2()	Returns the angle theta of an (x,y) point as a numeric value between -PI and PI radians	3
atanh()	Returns the inverse hyperbolic tangent of a number	4
base_convert()	Converts a number from one base to another	3
bindec()	Converts a binary number to a decimal number	3
ceil()	Returns the value of a number rounded upwards to the nearest integer	3
cos()	Returns the cosine of a number	3
cosh()	Returns the hyperbolic cosine of a number	4
decbin()	Converts a decimal number to a binary number	3
dechex()	Converts a decimal number to a hexadecimal number	3
decoct()	Converts a decimal number to an octal number	3
deg2rad()	Converts a degree to a radian number	3
exp()	Returns the value of E^x	3
expm1()	Returns the value of E^x - 1	4
floor()	Returns the value of a number rounded downwards to the nearest integer	3
fmod()	Returns the remainder (modulo) of the division of the arguments	4
getrandmax()	Returns the maximum random number that can be returned by a call to the rand() function	3
hexdec()	Converts a hexadecimal number to a decimal number	3
hypot()	Returns the length of the hypotenuse of a right-angle triangle	4
is_finite()	Returns true if a value is a finite number	4
is_infinite()	Returns true if a value is an infinite number	4
is_nan()	Returns true if a value is not a number	4
lcg_value()	Returns a pseudo random number in the range of (0,1)	4
log()	Returns the natural logarithm (base E) of a number	3
log10()	Returns the base-10 logarithm of a number	3
log1p()	Returns log(1+number)	4
max()	Returns the number with the highest value of two specified numbers	3
min()	Returns the number with the lowest value of two specified numbers	3
mt_getrandmax()	Returns the largest possible value that can be returned by mt_rand()	3
mt_rand()	Returns a random integer using Mersenne Twister algorithm	3
mt_srand()	Seeds the Mersenne Twister random number generator	3
octdec()	Converts an octal number to a decimal number	3
pi()	Returns the value of PI	3

pow()	Returns the value of x to the power of y	3
rad2deg()	Converts a radian number to a degree	3
rand()	Returns a random integer	3
round()	Rounds a number to the nearest integer	3
sin()	Returns the sine of a number	3
sinh()	Returns the hyperbolic sine of a number	4
sqrt()	Returns the square root of a number	3
srand()	Seeds the random number generator	3
tan()	Returns the tangent of an angle	3
tanh()	Returns the hyperbolic tangent of an angle	4

PHP Math Constants

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

Constant	**Description**	**PHP**
M_E	Returns e (approx. 2.718)	4
M_EULER	Returns Euler's constant (approx. 0.577)	4
M_LNPI	Returns the natural logarithm of PI (approx. 1.144)	4
M_LN2	Returns the natural logarithm of 2 (approx. 0.693)	4
M_LN10	Returns the natural logarithm of 10 (approx. 2.302)	4
M_LOG2E	Returns the base-2 logarithm of E (approx. 1.442)	4
M_LOG10E	Returns the base-10 logarithm of E (approx. 0.434)	4
M_PI	Returns PI (approx. 3.14159)	3
M_PI_2	Returns PI/2 (approx. 1.570)	4
M_PI_4	Returns PI/4 (approx. 0.785)	4
M_1_PI	Returns 1/PI (approx. 0.318)	4
M_2_PI	Returns 2/PI (approx. 0.636)	4
M_SQRTPI	Returns the square root of PI (approx. 1.772)	4
M_2_SQRTPI	Returns 2/square root of PI (approx. 1.128)	4
M_SQRT1_2	Returns the square root of 1/2 (approx. 0.707)	4
M_SQRT2	Returns the square root of 2 (approx. 1.414)	4
M_SQRT3	Returns the square root of 3 (approx. 1.732)	4

PHP Misc. Functions

PHP Miscellaneous Introduction

The misc. functions were only placed here because none of the other categories seemed to fit.

Installation

The misc functions are part of the PHP core. There is no installation needed to use these functions.

Runtime Configuration

The behavior of the misc functions is affected by settings in the php.ini file.

Misc. configuration options:

Name	Default	Description	Changeable
ignore_user_abort	"0"	FALSE indicates that scripts will be terminated as soon as they try to output something after a client has aborted their connection	PHP_INI_ALL
highlight.string	"#DD0000"	Color for highlighting a string in PHP syntax	PHP_INI_ALL
highlight.comment	"#FF8000"	Color for highlighting PHP comments	PHP_INI_ALL
highlight.keyword	"#007700"	Color for syntax highlighting PHP keywords (e.g. parenthesis and semicolon)	PHP_INI_ALL
highlight.bg	"#FFFFFF"	Color for background	PHP_INI_ALL
highlight.default	"#0000BB"	Default color for PHP syntax	PHP_INI_ALL
highlight.html	"#000000"	Color for HTML code	PHP_INI_ALL
browscap	NULL	Name and location of browser-capabilities file (e.g. browscap.ini)	PHP_INI_SYSTEM

PHP Misc. Functions

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

Function	Description	PHP
connection_aborted()	Checks whether the client has disconnected	3
connection_status()	Returns the current connection status	3
connection_timeout()	Deprecated in PHP 4.0.5	3
constant()	Returns the value of a constant	4
define()	Defines a constant	3
defined()	Checks whether a constant exists	3
die()	Prints a message and exits the current script	3
eval()	Evaluates a string as PHP code	3
exit()	Prints a message and exits the current script	3
get_browser()	Returns the capabilities of the user's browser	3
highlight_file()	Outputs a file with the PHP syntax highlighted	4
highlight_string()	Outputs a string with the PHP syntax highlighted	4
ignore_user_abort()	Sets whether a remote client can abort the running of a script	3
pack()	Packs data into a binary string	3
php_check_syntax()	Deprecated in PHP 5.0.5	5
php_strip_whitespace()	Returns the source code of a file with PHP comments and whitespace removed	5
show_source()	Alias of highlight_file()	4
sleep()	Delays code execution for a number of seconds	3
time_nanosleep()	Delays code execution for a number of seconds and nanoseconds	5
time_sleep_until()	Delays code execution until a specified time	5
uniqid()	Generates a unique ID	3
unpack()	Unpacks data from a binary string	3
usleep()	Delays code execution for a number of microseconds	3

PHP Misc. Constants

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

Constant	Description	PHP
CONNECTION_ABORTED		
CONNECTION_NORMAL		

CONNECTION_TIMEOUT		
__COMPILER_HALT_OFFSET__		5

PHP MySQL Functions

PHP MySQL Introduction

The MySQL functions allows you to access MySQL database servers.

Installation

For the MySQL functions to be available, you must compile PHP with MySQL support.

For compiling, use *--with-mysql=DIR* (the optional DIR points to the MySQL directory).

Note: For full functionality of MySQL versions greater than 4.1., use the MySQLi extension instead. If you would like to install both the mysql extension and the mysqli extension you should use the same client library to avoid any conflicts.

Installation on Linux Systems

PHP 5+: MySQL and the MySQL library is not enabled by default. Use the *--with-mysql=DIR* configure option to include MySQL support and download headers and libraries from www.mysql.com.

Installation on Windows Systems

PHP 5+: MySQL is not enabled by default, so the php_mysql.dll must be enabled inside of php.ini. Also, PHP needs access to the MySQL client library. A file named libmysql.dll is included in the Windows PHP distribution, and in order for PHP to talk to MySQL this file needs to be available to the Windows systems PATH.

To enable any PHP extension, the PHP extension_dir setting (in the php.ini file) should be set to the directory where the PHP extensions are located. An example extension_dir value is c:\php\ext.

Note: If you get the following error when starting the web server: "Unable to load dynamic library './php_mysql.dll'", this is because php_mysql.dll or libmysql.dll cannot be found by the system.

Runtime Configuration

The behavior of the MySQL functions is affected by settings in the php.ini file.

MySQL configuration options:

Name	Default	Description	Changeable
mysql.allow_persistent	"1"	Whether or not to allow persistent connections	PHP_INI_SYSTEM
mysql.max_persistent	"-1"	The maximum number of persistent connections per process	PHP_INI_SYSTEM
mysql.max_links	"-1"	The maximum number of connections per process (persistent connections included)	PHP_INI_SYSTEM
mysql.trace_mode	"0"	Trace mode. When set to "1", warnings	PHP_INI_ALL

		and SQL-errors will be displayed. Available since PHP 4.3	
mysql.default_port	NULL	The default TCP port number to use	PHP_INI_ALL
mysql.default_socket	NULL	The default socket name to use. Available since PHP 4.0.1	PHP_INI_ALL
mysql.default_host	NULL	The default server host to use (doesn't apply in SQL safe mode)	PHP_INI_ALL
mysql.default_user	NULL	The default user name to use (doesn't apply in SQL safe mode)	PHP_INI_ALL
mysql.default_password	NULL	The default password to use (doesn't apply in SQL safe mode)	PHP_INI_ALL
mysql.connect_timeout	"60"	Connection timeout in seconds	PHP_INI_ALL

Resource Types

There are two resource types used in the MySQL extension. The first one is the link_identifier for a database connection, the second is a resource which holds the result of a query.

Note: Most MySQL functions accept link_identifier as the last optional parameter. If it is not provided, the last opened connection is used.

PHP MySQL Functions

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

Function	Description	PHP
mysql_affected_rows()	Returns the number of affected rows in the previous MySQL operation	3
mysql_change_user()	Deprecated. Changes the user of the current MySQL connection	3
mysql_client_encoding()	Returns the name of the character set for the current connection	4
mysql_close()	Closes a non-persistent MySQL connection	3
mysql_connect()	Opens a non-persistent MySQL connection	3
mysql_create_db()	Deprecated. Creates a new MySQL database. Use mysql_query() instead	3
mysql_data_seek()	Moves the record pointer	3
mysql_db_name()	Returns a database name from a call to mysql_list_dbs()	3
mysql_db_query()	Deprecated. Sends a MySQL query. Use mysql_select_db() and mysql_query() instead	3
mysql_drop_db()	Deprecated. Deletes a MySQL database. Use mysql_query() instead	3
mysql_errno()	Returns the error number of the last MySQL operation	3
mysql_error()	Returns the error description of the last MySQL operation	3
mysql_escape_string()	Deprecated. Escapes a string for use in a mysql_query. Use mysql_real_escape_string() instead	4
mysql_fetch_array()	Returns a row from a recordset as an associative array and/or a numeric array	3
mysql_fetch_assoc()	Returns a row from a recordset as an associative array	4
mysql_fetch_field()	Returns column info from a recordset as an object	3
mysql_fetch_lengths()	Returns the length of the contents of each field in a result row	3
mysql_fetch_object()	Returns a row from a recordset as an object	3
mysql_fetch_row()	Returns a row from a recordset as a numeric array	3
mysql_field_flags()	Returns the flags associated with a field in a recordset	3
mysql_field_len()	Returns the maximum length of a field in a recordset	3

mysql_field_name()	Returns the name of a field in a recordset	3
mysql_field_seek()	Moves the result pointer to a specified field	3
mysql_field_table()	Returns the name of the table the specified field is in	3
mysql_field_type()	Returns the type of a field in a recordset	3
mysql_free_result()	Free result memory	3
mysql_get_client_info()	Returns MySQL client info	4
mysql_get_host_info()	Returns MySQL host info	4
mysql_get_proto_info()	Returns MySQL protocol info	4
mysql_get_server_info()	Returns MySQL server info	4
mysql_info()	Returns information about the last query	4
mysql_insert_id()	Returns the AUTO_INCREMENT ID generated from the previous INSERT operation	3
mysql_list_dbs()	Lists available databases on a MySQL server	3
mysql_list_fields()	Deprecated. Lists MySQL table fields. Use mysql_query() instead	3
mysql_list_processes()	Lists MySQL processes	4
mysql_list_tables()	Deprecated. Lists tables in a MySQL database. Use mysql_query() instead	3
mysql_num_fields()	Returns the number of fields in a recordset	3
mysql_num_rows()	Returns the number of rows in a recordset	3
mysql_pconnect()	Opens a persistent MySQL connection	3
mysql_ping()	Pings a server connection or reconnects if there is no connection	4
mysql_query()	Executes a query on a MySQL database	3
mysql_real_escape_string()	Escapes a string for use in SQL statements	4
mysql_result()	Returns the value of a field in a recordset	3
mysql_select_db()	Sets the active MySQL database	3
mysql_stat()	Returns the current system status of the MySQL server	4
mysql_tablename()	Deprecated. Returns the table name of field. Use mysql_query() instead	3
mysql_thread_id()	Returns the current thread ID	4
mysql_unbuffered_query()	Executes a query on a MySQL database (without fetching / buffering the result)	4

PHP MySQL Constants

Since PHP 4.3 it has been possible to specify additional flags for the mysql_connect() and mysql_pconnect() functions:

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

Constant	**Description**	**PHP**
MYSQL_CLIENT_COMPRESS	Use compression protocol	4.3
MYSQL_CLIENT_IGNORE_SPACE	Allow space after function names	4.3
MYSQL_CLIENT_INTERACTIVE	Allow interactive timeout seconds of inactivity before closing the connection	4.3
MYSQL_CLIENT_SSL	Use SSL encryption (only available with version 4+ of the MySQL client library)	4.3

The mysql_fetch_array() function uses a constant for the different types of result arrays. The following constants are defined:

Constant	**Description**	**PHP**
MYSQL_ASSOC	Columns are returned into the array with the fieldname as the array index	
MYSQL_BOTH	Columns are returned into the array having both a numerical index and the fieldname as the array index	

MYSQL_NUM	Columns are returned into the array having a numerical index (index starts at 0)

PHP SimpleXML Functions

PHP SimpleXML Introduction

The SimpleXML functions lets you convert XML to an object.

This object can be processed, like any other object, with normal property selectors and array iterators.

Some of these functions requires the newest PHP build.

Installation

The SimpleXML functions are part of the PHP core. There is no installation needed to use these functions.

PHP SimpleXML Functions

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

Function	Description	PHP
__construct()	Creates a new SimpleXMLElement object	5
addAttribute()	Adds an attribute to the SimpleXML element	5
addChild()	Adds a child element the SimpleXML element	5
asXML()	Gets an XML string from a SimpleXML element	5
attributes()	Gets a SimpleXML element's attributes	5
children()	Gets the children of a specified node	5
getDocNamespaces()	Gets the namespaces of an XML document	5
getName()	Gets the name of a SimpleXML element	5
getNamespaces()	Gets the namespaces from XML data	5
registerXPathNamespace()	Creates a namespace context for the next XPath query	5
simplexml_import_dom()	Gets a SimpleXMLElement object from a DOM node	5
simplexml_load_file()	Gets a SimpleXMLElement object from an XML document	5
simplexml_load_string()	Gets a SimpleXMLElement object from an XML string	5
xpath()	Runs an XPath query on XML data	5

PHP SimpleXML Constants

None

PHP XML Parser Functions

PHP XML Parser Introduction

The XML functions lets you parse, but not validate, XML documents.

XML is a data format for standardized structured document exchange. More information on XML can be found in our XML Tutorial.

This extension uses the Expat XML parser.

Expat is an event-based parser, it views an XML document as a series of events. When an event occurs, it calls a specified function to handle it.

Expat is a non-validating parser, and ignores any DTDs linked to a document. However, if the document is not well formed it will end with an error message.

Because it is an event-based, non validating parser, Expat is fast and well suited for web applications.

The XML parser functions lets you create XML parsers and define handlers for XML events.

Installation

The XML functions are part of the PHP core. There is no installation needed to use these functions.

PHP XML Parser Functions

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

Function	Description	PHP
utf8_decode()	Decodes an UTF-8 string to ISO-8859-1	3
utf8_encode()	Encodes an ISO-8859-1 string to UTF-8	3
xml_error_string()	Gets an error string from the XML parser	3
xml_get_current_byte_index()	Gets the current byte index from the XML parser	3
xml_get_current_column_number()	Gets the current column number from the XML parser	3
xml_get_current_line_number()	Gets the current line number from the XML parser	3
xml_get_error_code()	Gets an error code from the XML parser	3
xml_parse()	Parses an XML document	3
xml_parse_into_struct()	Parse XML data into an array	3
xml_parser_create_ns()	Create an XML parser with namespace support	4
xml_parser_create()	Create an XML parser	3
xml_parser_free()	Free an XML parser	3
xml_parser_get_option()	Get options from an XML parser	3
xml_parser_set_option()	Set options in an XML parser	3
xml_set_character_data_handler()	Set handler function for character data	3
xml_set_default_handler()	Set default handler function	3
xml_set_element_handler()	Set handler function for start and end element of elements	3
xml_set_end_namespace_decl_handler()	Set handler function for the end of namespace declarations	4
xml_set_external_entity_ref_handler()	Set handler function for external entities	3
xml_set_notation_decl_handler()	Set handler function for notation declarations	3
xml_set_object()	Use XML Parser within an object	4
xml_set_processing_instruction_handler()	Set handler function for processing instruction	3
xml_set_start_namespace_decl_handler()	Set handler function for the start of namespace declarations	4
xml_set_unparsed_entity_decl_handler()	Set handler function for unparsed entity	3

	declarations	

PHP XML Parser Constants

Constant
XML_ERROR_NONE (integer)
XML_ERROR_NO_MEMORY (integer)
XML_ERROR_SYNTAX (integer)
XML_ERROR_NO_ELEMENTS (integer)
XML_ERROR_INVALID_TOKEN (integer)
XML_ERROR_UNCLOSED_TOKEN (integer)
XML_ERROR_PARTIAL_CHAR (integer)
XML_ERROR_TAG_MISMATCH (integer)
XML_ERROR_DUPLICATE_ATTRIBUTE (integer)
XML_ERROR_JUNK_AFTER_DOC_ELEMENT (integer)
XML_ERROR_PARAM_ENTITY_REF (integer)
XML_ERROR_UNDEFINED_ENTITY (integer)
XML_ERROR_RECURSIVE_ENTITY_REF (integer)
XML_ERROR_ASYNC_ENTITY (integer)
XML_ERROR_BAD_CHAR_REF (integer)
XML_ERROR_BINARY_ENTITY_REF (integer)
XML_ERROR_ATTRIBUTE_EXTERNAL_ENTITY_REF (integer)
XML_ERROR_MISPLACED_XML_PI (integer)
XML_ERROR_UNKNOWN_ENCODING (integer)
XML_ERROR_INCORRECT_ENCODING (integer)
XML_ERROR_UNCLOSED_CDATA_SECTION (integer)
XML_ERROR_EXTERNAL_ENTITY_HANDLING (integer)
XML_OPTION_CASE_FOLDING (integer)
XML_OPTION_TARGET_ENCODING (integer)
XML_OPTION_SKIP_TAGSTART (integer)
XML_OPTION_SKIP_WHITE (integer)

PHP Zip File Functions

PHP Zip File Introduction

The Zip files functions allows you to read ZIP files.

Installation

For the Zip file functions to work on your server, these libraries must be installed:

- The ZZIPlib library by Guido Draheim: Download the ZZIPlib library
- The Zip PELC extension: Download the Zip PELC extension

Installation on Linux Systems

PHP 5+: Zip functions and the Zip library is not enabled by default and must be downloaded from the links above. Use the *--with-zip=DIR* configure option to include Zip support.

Installation on Windows Systems

PHP 5+: Zip functions is not enabled by default, so the php_zip.dll and the ZZIPlib library must be downloaded from the link above. php_zip.dll must be enabled inside of php.ini.

To enable any PHP extension, the PHP extension_dir setting (in the php.ini file) should be set to the directory where the PHP extensions are located. An example extension_dir value is c:\php\ext.

PHP Zip File Functions

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

Function	Description	PHP
zip_close()	Closes a ZIP file	4
zip_entry_close()	Closes an entry in the ZIP file	4
zip_entry_compressedsize()	Returns the compressed size of an entry in the ZIP file	4
zip_entry_compressionmethod()	Returns the compression method of an entry in the ZIP file	4
zip_entry_filesize()	Returns the actual file size of an entry in the ZIP file	4
zip_entry_name()	Returns the name of an entry in the ZIP file	4
zip_entry_open()	Opens an entry in the ZIP file for reading	4
zip_entry_read()	Reads from an open entry in the ZIP file	4
zip_open()	Opens a ZIP file	4
zip_read()	Reads the next entry in a ZIP file	4

PHP Zip File Constants

NONE