PHP 5 Form Handling

« Previous

Next Chapter »

The PHP superglobals $_GET and $_POST are used to collect form-data.

PHP - A Simple HTML Form

The example below displays a simple HTML form with two input fields and a submit button:

Example

```
<html>
<body>

<form action="welcome.php" method="post">
Name: <input type="text" name="name"><br>
E-mail: <input type="text" name="email"><br>
<input type="submit">
</form>

</body>
</html>
```

Run example »

When the user fills out the form above and clicks the submit button, the form data is sent for processing to a PHP file named "welcome.php". The form data is sent with the HTTP POST method.

To display the submitted data you could simply echo all the variables. The "welcome.php" looks like this:

```
<html>
<body>

Welcome <?php echo $_POST["name"]; ?><br>
Your email address is: <?php echo $_POST["email"]; ?>

</body>
</html>
```

The output could be something like this:

```
Welcome John
Your email address is john.doe@example.com
```

The same result could also be achieved using the HTTP GET method:

Example

```
<html>
<body>

<form action="welcome_get.php" method="get">
Name: <input type="text" name="name"><br>
E-mail: <input type="text" name="email"><br>
<input type="submit">
</form>

</body>
</html>
```

Run example »

and "welcome_get.php" looks like this:

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

```
Welcome <?php echo $_GET["name"]; ?><br>
Your email address is: <?php echo $_GET["email"]; ?>

</body>
</html>
```

The code above is quite simple. However, the most important thing is missing. You need to validate form data to protect your script from malicious code.

> **Think SECURITY when processing PHP forms!**
>
> This page does not contain any form validation, it just shows how you can send and re
> data.
>
> However, the next pages will show how to process PHP forms with security in mind! P
> validation of form data is important to protect your form from hackers and spammers

---

GET vs. POST

Both GET and POST create an array (e.g. array(key => value, key2 => value2, key3 => value3, ...)). This array holds key/value pairs, where keys are the names of the form controls and values are the input data from the user.

Both GET and POST are treated as $_GET and $_POST. These are superglobals, which means that they are always accessible, regardless of scope - and you can access them from any function, class or file without having to do anything special.

$_GET is an array of variables passed to the current script via the URL parameters.

$_POST is an array of variables passed to the current script via the HTTP POST method.

When to use GET?

Information sent from a form with the GET method is **visible to everyone** (all variable names and values are displayed in the URL). GET also has limits on the amount of information to send. The limitation is about 2000 characters. However, because the variables are displayed in the URL, it is possible to bookmark the page. This can be useful in some cases.

GET may be used for sending non-sensitive data.

Note: GET should NEVER be used for sending passwords or other sensitive information!

When to use POST?

Information sent from a form with the POST method is **invisible to others** (all names/values are embedded within the body of the HTTP request) and has **no limits** on the amount of information to send.

Moreover POST supports advanced functionality such as support for multi-part binary input while uploading files to server.

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

> **Developers prefer POST for sending form data.**

Next, lets see how we can process PHP forms the secure way!

PHP 5 Form Validation

« Previous

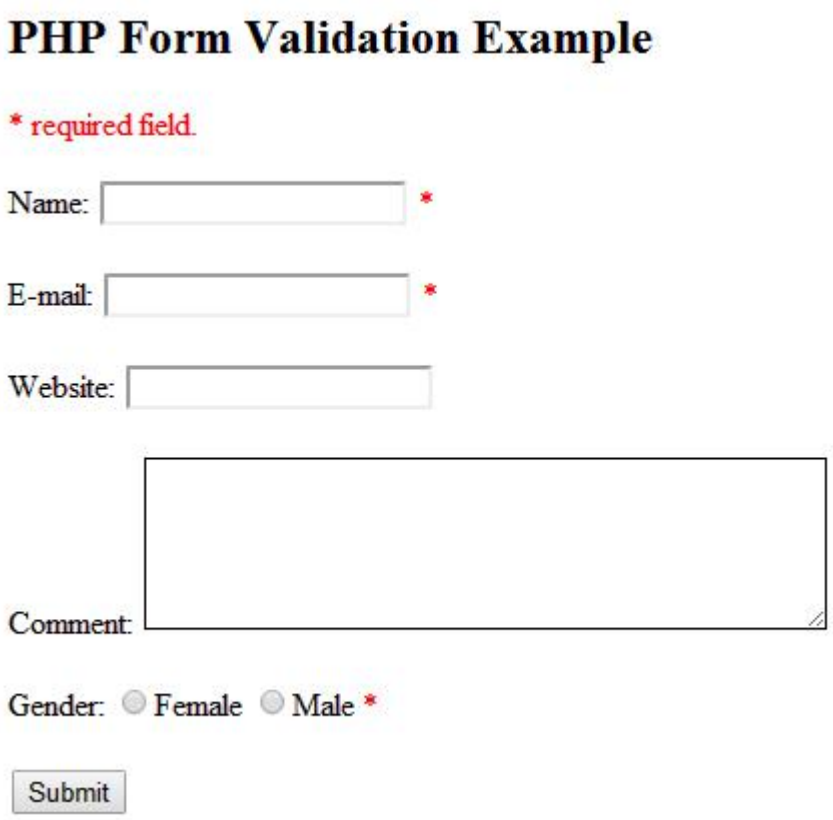
Next Chapter »

This and the next chapters show how to use PHP to validate form data.

PHP Form Validation

> **Think SECURITY when processing PHP forms!**
>
> These pages will show how to process PHP forms with security in mind. Proper validat
> data is important to protect your form from hackers and spammers!

The HTML form we will be working at in these chapters, contains various input fields: required and optional text fields, radio buttons, and a submit button:



The validation rules for the form above are as follows:

Field	Validation Rules

Name	Required. + Must only contain letters and whitespace
E-mail	Required. + Must contain a valid email address (with @ and .)
Website	Optional. If present, it must contain a valid URL
Comment	Optional. Multi-line input field (textarea)
Gender	Required. Must select one

First we will look at the plain HTML code for the form:

Text Fields

The name, email, and website fields are text input elements, and the comment field is a textarea. The HTML code looks like this:

```
Name: <input type="text" name="name">
E-mail: <input type="text" name="email">
Website: <input type="text" name="website">
Comment: <textarea name="comment" rows="5" cols="40"></textarea>
```

Radio Buttons

The gender fields are radio buttons and the HTML code looks like this:

```
Gender:
<input type="radio" name="gender" value="female">Female
<input type="radio" name="gender" value="male">Male
```

The Form Element

The HTML code of the form looks like this:

```
<form method="post" action="<?php echo
htmlspecialchars($_SERVER["PHP_SELF"]);?>">
```

When the form is submitted, the form data is sent with method="post".

> **What is the $_SERVER["PHP_SELF"] variable?**
>
> The $_SERVER["PHP_SELF"] is a super global variable that returns the filename of the currently executing script.

So, the $_SERVER["PHP_SELF"] sends the submitted form data to the page itself, instead of jumping to a different page. This way, the user will get error messages on the same page as the form.

> **What is the htmlspecialchars() function?**
>
> The htmlspecialchars() function converts special characters to HTML entities. This means that it will replace HTML characters like < and > with < and >. This prevents attackers from exploiting the code by injecting HTML or Javascript code (Cross-site Scripting attacks) in forms.

Big Note on PHP Form Security

The $_SERVER["PHP_SELF"] variable can be used by hackers!

If PHP_SELF is used in your page then a user can enter a slash (/) and then some Cross Site Scripting (XSS) commands to execute.

> **Cross-site scripting (XSS) is a type of computer security vulnerability typical Web applications. XSS enables attackers to inject client-side script into Web viewed by other users.**

Assume we have the following form in a page named "test_form.php":

```
<form method="post" action="<?php echo $_SERVER["PHP_SELF"];?>">
```

Now, if a user enters the normal URL in the address bar like "http://www.example.com/test_form.php", the above code will be translated to:

```
<form method="post" action="test_form.php">
```

So far, so good.

However, consider that a user enters the following URL in the address bar:

```
http://www.example.com/test_form.php/%22%3E%3Cscript%3Ealert('
hacked')%3C/script%3E
```

In this case, the above code will be translated to:

```
<form method="post"
action="test_form.php/"><script>alert('hacked')</script>
```

This code adds a script tag and an alert command. And when the page loads, the JavaScript code will be executed (the user will see an alert box). This is just a simple and harmless example how the PHP_SELF variable can be exploited.

Be aware of that **any JavaScript code can be added inside the <script> tag!** A hacker can redirect the user to a file on another server, and that file can hold malicious code that can alter the global variables or submit the form to another address to save the user data, for example.

How To Avoid $_SERVER["PHP_SELF"] Exploits?

$_SERVER["PHP_SELF"] exploits can be avoided by using the htmlspecialchars() function.

The form code should look like this:

```
<form method="post" action="<?php echo
htmlspecialchars($_SERVER["PHP_SELF"]);?>">
```

The htmlspecialchars() function converts special characters to HTML entities. Now if the user tries to exploit the PHP_SELF variable, it will result in the following output:

```
<form method="post"
action="test_form.php/&quot;&gt;&lt;script&gt;alert('hacked')&
lt;/script&gt;">
```

The exploit attempt fails, and no harm is done!

Validate Form Data With PHP

The first thing we will do is to pass all variables through PHP's htmlspecialchars() function.

When we use the htmlspecialchars() function; then if a user tries to submit the following in a text field:

<script>location.href('http://www.hacked.com')</script>

- this would not be executed, because it would be saved as HTML escaped code, like this:

<script>location.href('http://www.hacked.com')</script>

The code is now safe to be displayed on a page or inside an e-mail.

We will also do two more things when the user submits the form:

1. Strip unnecessary characters (extra space, tab, newline) from the user input data (with the PHP trim() function)
2. Remove backslashes (\) from the user input data (with the PHP stripslashes() function)

The next step is to create a function that will do all the checking for us (which is much more convenient than writing the same code over and over again).

We will name the function test_input().

Now, we can check each $_POST variable with the test_input() function, and the script looks like this:

Example

```php
<?php
// define variables and set to empty values
$name = $email = $gender = $comment = $website = "";

if ($_SERVER["REQUEST_METHOD"] == "POST") {
  $name = test_input($_POST["name"]);
  $email = test_input($_POST["email"]);
  $website = test_input($_POST["website"]);
  $comment = test_input($_POST["comment"]);
  $gender = test_input($_POST["gender"]);
}

function test_input($data) {
  $data = trim($data);
  $data = stripslashes($data);
  $data = htmlspecialchars($data);
  return $data;
}
?>
```

Run example »

Notice that at the start of the script, we check whether the form has been submitted using $_SERVER["REQUEST_METHOD"]. If the REQUEST_METHOD is POST, then the form has been submitted - and it should be validated. If it has not been submitted, skip the validation and display a blank form.

However, in the example above, all input fields are optional. The script works fine even if the user does not enter any data.

The next step is to make input fields required and create error messages if needed.

PHP 5 Forms - Required Fields

« Previous

Next Chapter »

This chapter shows how to make input fields required and create error messages if needed.

PHP - Required Fields

From the validation rules table on the previous page, we see that the "Name", "E-mail", and "Gender" fields are required. These fields cannot be empty and must be filled out in the HTML form.

Field	Validation Rules
Name	Required. + Must only contain letters and whitespace
E-mail	Required. + Must contain a valid email address (with @ and .)
Website	Optional. If present, it must contain a valid URL
Comment	Optional. Multi-line input field (textarea)
Gender	Required. Must select one

In the previous chapter, all input fields were optional.

In the following code we have added some new variables: $nameErr, $emailErr, $genderErr, and $websiteErr. These error variables will hold error messages for the required fields. We have also added an if else statement for each $_POST variable. This checks if the $_POST variable is empty (with the PHP empty() function). If it is empty, an error message is stored in the different error variables, and if it is not empty, it sends the user input data through the test_input() function:

```php
<?php
// define variables and set to empty values
$nameErr = $emailErr = $genderErr = $websiteErr = "";
$name = $email = $gender = $comment = $website = "";

if ($_SERVER["REQUEST_METHOD"] == "POST") {
  if (empty($_POST["name"])) {
    $nameErr = "Name is required";
  } else {
    $name = test_input($_POST["name"]);
  }

  if (empty($_POST["email"])) {
    $emailErr = "Email is required";
  } else {
    $email = test_input($_POST["email"]);
  }

  if (empty($_POST["website"])) {
    $website = "";
  } else {
    $website = test_input($_POST["website"]);
  }

  if (empty($_POST["comment"])) {
    $comment = "";
  } else {
    $comment = test_input($_POST["comment"]);
  }

  if (empty($_POST["gender"])) {
    $genderErr = "Gender is required";
  } else {
    $gender = test_input($_POST["gender"]);
  }
}
?>
```

PHP - Display The Error Messages

Then in the HTML form, we add a little script after each required field, which generates the correct error message if needed (that is if the user tries to submit the form without filling out the required fields):

Example

```
<form method="post" action="<?php echo htmlspecialchars($_SERV
ER["PHP_SELF"]);?>">

Name: <input type="text" name="name">
<span class="error">* <?php echo $nameErr;?></span>
<br><br>
E-mail:
<input type="text" name="email">
<span class="error">* <?php echo $emailErr;?></span>
<br><br>
Website:
<input type="text" name="website">
<span class="error"><?php echo $websiteErr;?></span>
<br><br>
Comment: <textarea name="comment" rows="5" cols="40"></textare
a>
<br><br>
Gender:
<input type="radio" name="gender" value="female">Female
<input type="radio" name="gender" value="male">Male
<span class="error">* <?php echo $genderErr;?></span>
<br><br>
<input type="submit" name="submit" value="Submit">

</form>
```

Run example »

The next step is to validate the input data, that is "Does the Name field contain only letters and whitespace?", and "Does the E-mail field contain a valid e-mail address syntax?", and if filled out, "Does the Website field contain a valid URL?".

PHP 5 Forms - Validate E-mail and URL

« Previous

Next Chapter »

This chapter shows how to validate names, e-mails, and URLs.

PHP - Validate Name

The code below shows a simple way to check if the name field only contains letters and whitespace. If the value of the name field is not valid, then store an error message:

```
$name = test_input($_POST["name"]);
if (!preg_match("/^[a-zA-Z ]*$/",$name)) {
  $nameErr = "Only letters and white space allowed";
}
```

> **The preg_match() function searches a string for pattern, returning true if th exists, and false otherwise.**

PHP - Validate E-mail

The easiest and safest way to check whether an email address is well-formed is to use PHP's filter_var() function.

In the code below, if the e-mail address is not well-formed, then store an error message:

```
$email = test_input($_POST["email"]);
if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
  $emailErr = "Invalid email format";
}
```

PHP - Validate URL

The code below shows a way to check if a URL address syntax is valid (this regular expression also allows dashes in the URL). If the URL address syntax is not valid, then store an error message:

```
$website = test_input($_POST["website"]);
if
(!preg_match("/\b(?:(?:https?|ftp):\/\/|www\.)[-a-z0-9+&@#\/%?
=~_|!:,.;]*[-a-z0-9+&@#\/%=~_|]/i",$website)) {
  $websiteErr = "Invalid URL";
}
```

PHP - Validate Name, E-mail, and URL

Now, the script looks like this:

Example

```
<?php
// define variables and set to empty values
$nameErr = $emailErr = $genderErr = $websiteErr = "";
$name = $email = $gender = $comment = $website = "";

if ($_SERVER["REQUEST_METHOD"] == "POST") {
  if (empty($_POST["name"])) {
```

```
    $nameErr = "Name is required";
  } else {
    $name = test_input($_POST["name"]);
    // check if name only contains letters and whitespace
    if (!preg_match("/^[a-zA-Z ]*$/",$name)) {
      $nameErr = "Only letters and white space allowed";
    }
  }

  if (empty($_POST["email"])) {
    $emailErr = "Email is required";
  } else {
    $email = test_input($_POST["email"]);
    // check if e-mail address is well-formed
    if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
      $emailErr = "Invalid email format";
    }
  }

  if (empty($_POST["website"])) {
    $website = "";
  } else {
    $website = test_input($_POST["website"]);
    // check if URL address syntax is valid (this regular expression
also allows dashes in the URL)
    if (!preg_match("/\b(?:(?:https?|ftp):\/\/|www\.)[-a-z0-9+
&@#\/%?=~_|!:,.;]*[-a-z0-9+&@#\/%=~_|]/i",$website)) {
      $websiteErr = "Invalid URL";
    }
  }

  if (empty($_POST["comment"])) {
    $comment = "";
  } else {
    $comment = test_input($_POST["comment"]);
  }

  if (empty($_POST["gender"])) {
    $genderErr = "Gender is required";
  } else {
    $gender = test_input($_POST["gender"]);
  }
}
?>
```

Run example »

The next step is to show how to prevent the form from emptying all the input fields when the user submits the form.

PHP 5 Complete Form Example

« Previous

Next Chapter »

This chapter shows how to keep the values in the input fields when the user hits the submit button.

PHP - Keep The Values in The Form

To show the values in the input fields after the user hits the submit button, we add a little PHP script inside the value attribute of the following input fields: name, email, and website. In the comment textarea field, we put the script between the <textarea> and </textarea> tags. The little script outputs the value of the $name, $email, $website, and $comment variables.

Then, we also need to show which radio button that was checked. For this, we must manipulate the checked attribute (not the value attribute for radio buttons):

```
Name: <input type="text" name="name" value="<?php echo $name;?>">

E-mail: <input type="text" name="email" value="<?php echo
$email;?>">

Website: <input type="text" name="website" value="<?php echo
$website;?>">
```

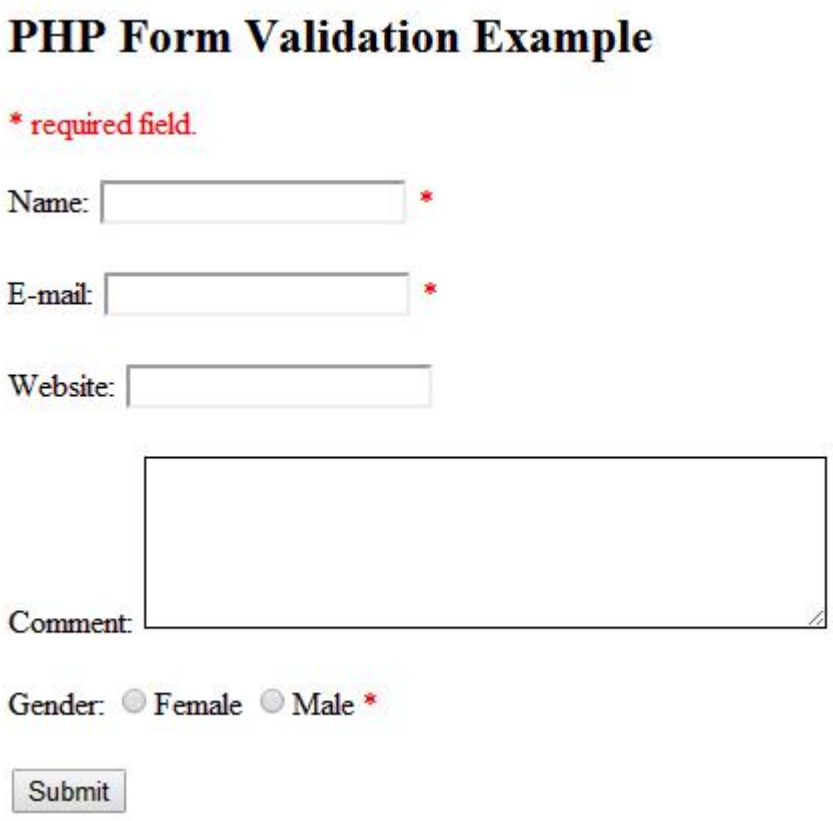
```
Comment: <textarea name="comment" rows="5" cols="40"><?php echo
$comment;?></textarea>

Gender:
<input type="radio" name="gender"
<?php if (isset($gender) && $gender=="female") echo "checked";?>
value="female">Female
<input type="radio" name="gender"
<?php if (isset($gender) && $gender=="male") echo "checked";?>
value="male">Male
```

PHP - Complete Form Example

Here is the complete code for the PHP Form Validation Example:

Example



Run example »

PHP 5 Multidimensional Arrays

« Previous

Next Chapter »

Earlier in this tutorial, we have described arrays that are a single list of key/value pairs.

However, sometimes you want to store values with more than one key.

This can be stored in multidimensional arrays.

PHP - Multidimensional Arrays

A multidimensional array is an array containing one or more arrays.

PHP understands multidimensional arrays that are two, three, four, five, or more levels deep. However, arrays more than three levels deep are hard to manage for most people.

The dimension of an array indicates the number of indices you need to select a

- For a two-dimensional array you need two indices to select an element
- For a three-dimensional array you need three indices to select an element

---

PHP - Two-dimensional Arrays

A two-dimensional array is an array of arrays (a three-dimensional array is an array of arrays of arrays).

First, take a look at the following table:

Name	Stock	Sold
Volvo	22	18
BMW	15	13
Saab	5	2
Land Rover	17	15

We can store the data from the table above in a two-dimensional array, like this:

```
$cars = array
  (
  array("Volvo",22,18),
  array("BMW",15,13),
  array("Saab",5,2),
  array("Land Rover",17,15)
  );
```

Now the two-dimensional $cars array contains four arrays, and it has two indices: row and column.

To get access to the elements of the $cars array we must point to the two indices (row and column):

Example

```
<?php
echo $cars[0][0].": In stock: ".$cars[0][1].", sold:
".$cars[0][2].".<br>";
echo $cars[1][0].": In stock: ".$cars[1][1].", sold:
".$cars[1][2].".<br>";
echo $cars[2][0].": In stock: ".$cars[2][1].", sold:
".$cars[2][2].".<br>";
```

```
echo $cars[3][0].": In stock: ".$cars[3][1].", sold:
".$cars[3][2].".<br>";
?>
```

Run example »

We can also put a For loop inside another For loop to get the elements of the $cars array (we still have to point to the two indices):

Example

```
<?php
for ($row = 0; $row < 4; $row++) {
  echo "<p><b>Row number $row</b></p>";
  echo "<ul>";
  for ($col = 0; $col < 3; $col++) {
    echo "<li>".$cars[$row][$col]."</li>";
  }
  echo "</ul>";
}
?>
```

Run example »

PHP 5 Date and Time

« Previous

Next Chapter »

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

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

> A timestamp is a sequence of characters, denoting the date and/or time at which a ce
> occurred.

Get a Simple Date

The required *format* parameter of the date() function specifies how to format the date (or time).

Here are some characters that are commonly used for dates:

- d - Represents the day of the month (01 to 31)
- m - Represents a month (01 to 12)
- Y - Represents a year (in four digits)
- l (lowercase 'L') - Represents the day of the week

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

The example below formats today's date in three different ways:

Example

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

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

Run example »

PHP Tip - Automatic Copyright Year

Use the date() function to automatically update the copyright year on your website:

Example

```php
&copy; 2010-<?php echo date("Y");?>
```

Run example »

Get a Simple Time

Here are some characters that are commonly used for times:

- h - 12-hour format of an hour with leading zeros (01 to 12)
- i - Minutes with leading zeros (00 to 59)
- s - Seconds with leading zeros (00 to 59)
- a - Lowercase Ante meridiem and Post meridiem (am or pm)

The example below outputs the current time in the specified format:

Example

```php
<?php
echo "The time is " . date("h:i:sa");
?>
```

Run example »

Note that the PHP date() function will return the current date/time of the server!

Get Your Time Zone

If the time you got back from the code is not the right time, it's probably because your server is in another country or set up for a different timezone.

So, if you need the time to be correct according to a specific location, you can set a timezone to use.

The example below sets the timezone to "America/New_York", then outputs the current time in the specified format:

Example

```php
<?php
date_default_timezone_set("America/New_York");
echo "The time is " . date("h:i:sa");
?>
```

Run example »

Create a Date With PHP mktime()

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

The mktime() function returns the Unix timestamp for a date. The Unix timestamp contains the number of seconds between the Unix Epoch (January 1 1970 00:00:00 GMT) and the time specified.

Syntax

```
mktime(hour,minute,second,month,day,year)
```

The example below creates a date and time from a number of parameters in the mktime() function:

Example

```php
<?php
$d=mktime(11, 14, 54, 8, 12, 2014);
echo "Created date is " . date("Y-m-d h:i:sa", $d);
?>
```

Run example »

Create a Date From a String With PHP strtotime()

The PHP strtotime() function is used to convert a human readable string to a Unix time.

Syntax

```
strtotime(time,now)
```

The example below creates a date and time from the strtotime() function:

Example

```php
<?php
$d=strtotime("10:30pm April 15 2014");
```

```
echo "Created date is " . date("Y-m-d h:i:sa", $d);
?>
```

Run example »

PHP is quite clever about converting a string to a date, so you can put in various values:

Example

```
<?php
$d=strtotime("tomorrow");
echo date("Y-m-d h:i:sa", $d) . "<br>";

$d=strtotime("next Saturday");
echo date("Y-m-d h:i:sa", $d) . "<br>";

$d=strtotime("+3 Months");
echo date("Y-m-d h:i:sa", $d) . "<br>";
?>
```

Run example »

However, strtotime() is not perfect, so remember to check the strings you put in there.

More Date Examples

The example below outputs the dates for the next six Saturdays:

Example

```
<?php
$startdate = strtotime("Saturday");
$enddate = strtotime("+6 weeks", $startdate);

while ($startdate < $enddate) {
```

```
  echo date("M d", $startdate) . "<br>";
  $startdate = strtotime("+1 week", $startdate);
}
?>
```

Run example »

The example below outputs the number of days until 4th of July:

Example

```
<?php
$d1=strtotime("July 04");
$d2=ceil(($d1-time())/60/60/24);
echo "There are " . $d2 ." days until 4th of July.";
?>
```

Run example »

Complete PHP Date Reference

For a complete reference of all date functions, go to our complete PHP Date Reference.

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

PHP 5 Include Files

« Previous

Next Chapter »

The include (or require) statement takes all the text/code/markup that exists in the specified file and copies it into the file that uses the include statement.

Including files is very useful when you want to include the same PHP, HTML, or text on multiple pages of a website.

PHP include and require Statements

It is possible to insert the content of one PHP file into another PHP file (before the server executes it), with the include or require statement.

The include and require statements are identical, except upon failure:

- require will produce a fatal error (E_COMPILE_ERROR) and stop the script
- include will only produce a warning (E_WARNING) and the script will continue

So, if you want the execution to go on and show users the output, even if the include file is missing, use the include statement. Otherwise, in case of FrameWork, CMS, or a complex PHP application coding, always use the require statement to include a key file to the flow of execution. This will help avoid compromising your application's security and integrity, just in-case one key file is accidentally missing.

Including files saves a lot of work. This means that you can create a standard header, footer, or menu file for all your web pages. Then, when the header needs to be updated, you can only update the header include file.

Syntax

```
include 'filename';

or

require 'filename';
```

PHP include Examples

Example 1

Assume we have a standard footer file called "footer.php", that looks like this:

```
<?php
echo "<p>Copyright &copy; 1999-" . date("Y") . " W3Schools.com</p>";
?>
```

To include the footer file in a page, use the include statement:

Example

```
<html>
<body>

<h1>Welcome to my home page!</h1>
<p>Some text.</p>
<p>Some more text.</p>
<?php include 'footer.php';?>

</body>
</html>
```

Run example »

Example 2

Assume we have a standard menu file called "menu.php":

```
<?php
echo '<a href="/default.asp">Home</a> -
```

```
<a href="/html/default.asp">HTML Tutorial</a> -
<a href="/css/default.asp">CSS Tutorial</a> -
<a href="/js/default.asp">JavaScript Tutorial</a> -
<a href="default.asp">PHP Tutorial</a>';
?>
```

All pages in the Web site should use this menu file. Here is how it can be done (we are using a element so that the menu easily can be styled with CSS later):

Example

```
<html>
<body>

<div class="menu">
<?php include 'menu.php';?>
</div>

<h1>Welcome to my home page!</h1>
<p>Some text.</p>
<p>Some more text.</p>

</body>
</html>
```

Run example »

Example 3

Assume we have a file called "vars.php", with some variables defined:

```
<?php
$color='red';
$car='BMW';
?>
```

Then, if we include the "vars.php" file, the variables can be used in the calling file:

Example

```
<html>
<body>

<h1>Welcome to my home page!</h1>
<?php include 'vars.php';
echo "I have a $color $car.";
?>

</body>
</html>
```

Run example »

PHP include vs. require

The require statement is also used to include a file into the PHP code.

However, there is one big difference between include and require; when a file is included with the**include** statement and PHP cannot find it, the script will continue to execute:

Example

```
<html>
<body>

<h1>Welcome to my home page!</h1>
<?php include 'noFileExists.php';
echo "I have a $color $car.";
?>
```

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

Run example »

If we do the same example using the **require** statement, the echo statement will not be executed because the script execution dies after the require statement returned a fatal error:

Example

```
<html>
<body>

<h1>Welcome to my home page!</h1>
<?php require 'noFileExists.php';
echo "I have a $color $car.";
?>

</body>
</html>
```

Run example »

Use **require** when the file is required by the application.

Use **include** when the file is not required and application should continue when file is

PHP 5 File Handling

« Previous

Next Chapter »

File handling is an important part of any web application. You often need to open and process a file for different tasks.

PHP Manipulating Files

PHP has several functions for creating, reading, uploading, and editing files.

> **Be careful when manipulating files!**
>
> When you are manipulating files you must be very careful. You can do a lot of damag
> something wrong. Common errors are: editing the wrong file, filling a hard-drive with
> data, and deleting the content of a file by accident.

PHP readfile() Function

The readfile() function reads a file and writes it to the output buffer.

Assume we have a text file called "webdictionary.txt", stored on the server, that looks like this:

```
AJAX = Asynchronous JavaScript and XML
CSS = Cascading Style Sheets
HTML = Hyper Text Markup Language
PHP = PHP Hypertext Preprocessor
SQL = Structured Query Language
SVG = Scalable Vector Graphics
XML = EXtensible Markup Language
```

The PHP code to read the file and write it to the output buffer is as follows (the readfile() function returns the number of bytes read on success):

Example

```php
<?php
echo readfile("webdictionary.txt");
?>
```

Run example »

The readfile() function is useful if all you want to do is open up a file and read its contents.

The next chapters will teach you more about file handling.

PHP 5 File Open/Read/Close

« Previous

Next Chapter »

In this chapter we will teach you how to open, read, and close a file on the server.

PHP Open File - fopen()

A better method to open files is with the fopen() function. This function gives you more options than the readfile() function.

We will use the text file, "webdictionary.txt", during the lessons:

```
AJAX = Asynchronous JavaScript and XML
CSS = Cascading Style Sheets
HTML = Hyper Text Markup Language
PHP = PHP Hypertext Preprocessor
SQL = Structured Query Language
SVG = Scalable Vector Graphics
XML = EXtensible Markup Language
```

The first parameter of fopen() contains the name of the file to be opened and the second parameter specifies in which mode the file should be opened. The following example also generates a message if the fopen() function is unable to open the specified file:

Example

```php
<?php
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");
echo fread($myfile,filesize("webdictionary.txt"));
fclose($myfile);
?>
```

Run example »

Tip: The fread() and the fclose() functions will be explained below.

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

Modes	Description
r	**Open a file for read only**. File pointer starts at the beginning of the file
w	**Open a file for write only**. Erases the contents of the file or creates a new file exist. File pointer starts at the beginning of the file
a	**Open a file for write only**. The existing data in file is preserved. File pointer st end of the file. Creates a new file if the file doesn't exist
x	**Creates a new file for write only**. Returns FALSE and an error if file already e
r+	**Open a file for read/write**. File pointer starts at the beginning of the file
w+	**Open a file for read/write**. Erases the contents of the file or creates a new file exist. File pointer starts at the beginning of the file
a+	**Open a file for read/write**. The existing data in file is preserved. File pointer s end of the file. Creates a new file if the file doesn't exist
x+	**Creates a new file for read/write**. Returns FALSE and an error if file already

---

PHP Read File - fread()

The fread() function reads from an open file.

The first parameter of fread() contains the name of the file to read from and the second parameter specifies the maximum number of bytes to read.

The following PHP code reads the "webdictionary.txt" file to the end:

```
fread($myfile,filesize("webdictionary.txt"));
```

PHP Close File - fclose()

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

> It's a good programming practice to close all files after you have finished with them. You
> an open file running around on your server taking up resources!

The fclose() requires the name of the file (or a variable that holds the filename) we want to close:

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

PHP Read Single Line - fgets()

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

The example below outputs the first line of the "webdictionary.txt" file:

Example

```
<?php
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open
```

```
file!");
echo fgets($myfile);
fclose($myfile);
?>
```

Run example »

Note: After a call to the fgets() function, the file pointer has moved to the next line.

PHP Check End-Of-File - feof()

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.

The example below reads the "webdictionary.txt" file line by line, until end-of-file is reached:

Example

```
<?php
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open
file!");
// Output one line until end-of-file
while(!feof($myfile)) {
  echo fgets($myfile) . "<br>";
}
fclose($myfile);
?>
```

Run example »

PHP Read Single Character - fgetc()

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

The example below reads the "webdictionary.txt" file character by character, until end-of-file is reached:

Example

```php
<?php
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");
// Output one character until end-of-file
while(!feof($myfile)) {
  echo fgetc($myfile);
}
fclose($myfile);
?>
```

Run example »

Note: After a call to the fgetc() function, the file pointer moves to the next character.

Complete PHP Filesystem Reference

For a complete reference of filesystem functions, go to our complete PHP Filesystem Reference.

PHP 5 File Create/Write

« Previous

Next Chapter »

In this chapter we will teach you how to create and write to a file on the server.

PHP Create File - fopen()

The fopen() function is also used to create a file. Maybe a little confusing, but in PHP, a file is created using the same function used to open files.

If you use fopen() on a file that does not exist, it will create it, given that the file is opened for writing (w) or appending (a).

The example below creates a new file called "testfile.txt". The file will be created in the same directory where the PHP code resides:

Example

```
$myfile = fopen("testfile.txt", "w")
```

PHP File Permissions

If you are having errors when trying to get this code to run, check that you have granted your PHP file access to write information to the hard drive.

PHP Write to File - fwrite()

The fwrite() function is used to write to a file.

The first parameter of fwrite() contains the name of the file to write to and the second parameter is the string to be written.

The example below writes a couple of names into a new file called "newfile.txt":

Example

```php
<?php
$myfile = fopen("newfile.txt", "w") or die("Unable to open file!");
$txt = "John Doe\n";
fwrite($myfile, $txt);
$txt = "Jane Doe\n";
fwrite($myfile, $txt);
fclose($myfile);
?>
```

Notice that we wrote to the file "newfile.txt" twice. Each time we wrote to the file we sent the string $txt that first contained "John Doe" and second contained "Jane Doe". After we finished writing, we closed the file using the fclose() function.

If we open the "newfile.txt" file it would look like this:

```
John Doe
Jane Doe
```

PHP Overwriting

Now that "newfile.txt" contains some data we can show what happens when we open an existing file for writing. All the existing data will be ERASED and we start with an empty file.

In the example below we open our existing file "newfile.txt", and write some new data into it:

Example

```php
<?php
$myfile = fopen("newfile.txt", "w") or die("Unable to open file!");
$txt = "Mickey Mouse\n";
fwrite($myfile, $txt);
```

```
$txt = "Minnie Mouse\n";
fwrite($myfile, $txt);
fclose($myfile);
?>
```

If we now open the "newfile.txt" file, both John and Jane have vanished, and only the data we just wrote is present:

```
Mickey Mouse
Minnie Mouse
```

Complete PHP Filesystem Reference

For a complete reference of filesystem functions, go to our complete PHP Filesystem Reference.

PHP 5 File Upload

« Previous

Next Chapter »

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

However, with ease comes danger, so always be careful when allowing file uploads!

Configure The "php.ini" File

First, ensure that PHP is configured to allow file uploads.

In your "php.ini" file, search for the file_uploads directive, and set it to On:

```
file_uploads = On
```

Create The HTML Form

Next, create an HTML form that allow users to choose the image file they want to upload:

```
<!DOCTYPE html>
<html>
<body>

<form action="upload.php" method="post" enctype="multipart/form-data">
    Select image to upload:
    <input type="file" name="fileToUpload" id="fileToUpload">
    <input type="submit" value="Upload Image" name="submit">
</form>

</body>
</html>
```

Some rules to follow for the HTML form above:

- Make sure that the form uses method="post"
- The form also needs the following attribute: enctype="multipart/form-data". It specifies which content-type to use when submitting the form

Without the requirements above, the file upload will not work.

Other things to notice:

- The type="file" attribute of the <input> tag shows the input field as a file-select control, with a "Browse" button next to the input control

The form above sends data to a file called "upload.php", which we will create next.

Create The Upload File PHP Script

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

```php
<?php
$target_dir = "uploads/";
$target_file = $target_dir .
basename($_FILES["fileToUpload"]["name"]);
$uploadOk = 1;
$imageFileType = pathinfo($target_file,PATHINFO_EXTENSION);
// Check if image file is a actual image or fake image
if(isset($_POST["submit"])) {
    $check = getimagesize($_FILES["fileToUpload"]["tmp_name"]);
    if($check !== false) {
        echo "File is an image - " . $check["mime"] . ".";
        $uploadOk = 1;
    } else {
        echo "File is not an image.";
        $uploadOk = 0;
    }
}
?>
```

PHP script explained:

- $target_dir = "uploads/" - specifies the directory where the file is going to be placed
- $target_file specifies the path of the file to be uploaded
- $uploadOk=1 is not used yet (will be used later)
- $imageFileType holds the file extension of the file
- Next, check if the image file is an actual image or a fake image

Note: You will need to create a new directory called "uploads" in the directory where "u
file resides. The uploaded files will be saved there.

Check if File Already Exists

Now we can add some restrictions.

First, we will check if the file already exists in the "uploads" folder. If it does, an error message is displayed, and $uploadOk is set to 0:

```
// Check if file already exists
if (file_exists($target_file)) {
    echo "Sorry, file already exists.";
    $uploadOk = 0;
}
```

Limit File Size

The file input field in our HTML form above is named "fileToUpload".

Now, we want to check the size of the file. If the file is larger than 500kb, an error message is displayed, and $uploadOk is set to 0:

```
 // Check file size
if ($_FILES["fileToUpload"]["size"] > 500000) {
    echo "Sorry, your file is too large.";
    $uploadOk = 0;
}
```

Limit File Type

The code below only allows users to upload JPG, JPEG, PNG, and GIF files. All other file types gives an error message before setting $uploadOk to 0:

```
// Allow certain file formats
if($imageFileType != "jpg" && $imageFileType != "png" &&
$imageFileType != "jpeg"
&& $imageFileType != "gif" ) {
    echo "Sorry, only JPG, JPEG, PNG & GIF files are allowed.";
    $uploadOk = 0;
}
```

Complete Upload File PHP Script

The complete "upload.php" file now looks like this:

```
<?php
$target_dir = "uploads/";
$target_file = $target_dir .
basename($_FILES["fileToUpload"]["name"]);
$uploadOk = 1;
$imageFileType = pathinfo($target_file,PATHINFO_EXTENSION);
// Check if image file is a actual image or fake image
if(isset($_POST["submit"])) {
    $check = getimagesize($_FILES["fileToUpload"]["tmp_name"]);
    if($check !== false) {
        echo "File is an image - " . $check["mime"] . ".";
        $uploadOk = 1;
    } else {
        echo "File is not an image.";
        $uploadOk = 0;
    }
}
// Check if file already exists
if (file_exists($target_file)) {
    echo "Sorry, file already exists.";
    $uploadOk = 0;
}
// Check file size
if ($_FILES["fileToUpload"]["size"] > 500000) {
    echo "Sorry, your file is too large.";
    $uploadOk = 0;
}
// Allow certain file formats
if($imageFileType != "jpg" && $imageFileType != "png" &&
$imageFileType !="jpeg"
&& $imageFileType != "gif" ) {
    echo "Sorry, only JPG, JPEG, PNG & GIF files are allowed.";
    $uploadOk = 0;
}
// Check if $uploadOk is set to 0 by an error
if ($uploadOk == 0) {
    echo "Sorry, your file was not uploaded.";
// if everything is ok, try to upload file
} else {
```

```
    if (move_uploaded_file($_FILES["fileToUpload"]["tmp_name"],
$target_file)) {
        echo "The file ".
basename( $_FILES["fileToUpload"]["name"]). " has been
uploaded.";
    } else {
        echo "Sorry, there was an error uploading your file.";
    }
}
?>
```

Complete PHP Filesystem Reference

For a complete reference of filesystem functions, go to our complete PHP Filesystem Reference.

PHP 5 Cookies

« Previous

Next Chapter »

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.

Create Cookies With PHP

A cookie is created with the setcookie() function.

Syntax

```
setcookie(name, value, expire, path, domain, secure, httponly);
```

Only the name parameter is required. All other parameters are optional.

PHP Create/Retrieve a Cookie

The following example creates a cookie named "user" with the value "John Doe". The cookie will expire after 30 days (86400 * 30). The "/" means that the cookie is available in entire website (otherwise, select the directory you prefer).

We then retrieve the value of the cookie "user" (using the global variable $_COOKIE). We also use the isset() function to find out if the cookie is set:

Example

```
<?php
$cookie_name = "user";
$cookie_value = "John Doe";
setcookie($cookie_name, $cookie_value, time() +
(86400 * 30), "/"); // 86400 = 1 day
?>
<html>
<body>

<?php
if(!isset($_COOKIE[$cookie_name])) {
    echo "Cookie named '" . $cookie_name . "' is not set!";
} else {
    echo "Cookie '" . $cookie_name . "' is set!<br>";
```

```
    echo "Value is: " . $_COOKIE[$cookie_name];
}
?>

</body>
</html>
```

Run example »

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

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

Modify a Cookie Value

To modify a cookie, just set (again) the cookie using the setcookie() function:

Example

```
<?php
$cookie_name = "user";
$cookie_value = "Alex Porter";
setcookie($cookie_name, $cookie_value, time() +
(86400 * 30), "/");
?>
<html>
<body>

<?php
if(!isset($_COOKIE[$cookie_name])) {
    echo "Cookie named '" . $cookie_name . "' is not set!";
} else {
    echo "Cookie '" . $cookie_name . "' is set!<br>";
    echo "Value is: " . $_COOKIE[$cookie_name];
```

```
}
?>

</body>
</html>
```

Run example »

Delete a Cookie

To delete a cookie, use the setcookie() function with an expiration date in the past:

Example

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

<?php
echo "Cookie 'user' is deleted.";
?>

</body>
</html>
```

Run example »

Check if Cookies are Enabled

The following example creates a small script that checks whether cookies are enabled. First, try to create a test cookie with the setcookie() function, then count the $_COOKIE array variable:

Example

```
<?php
setcookie("test_cookie", "test", time() + 3600, '/');
?>
<html>
<body>

<?php
if(count($_COOKIE) > 0) {
    echo "Cookies are enabled.";
} else {
    echo "Cookies are disabled.";
}
?>

</body>
</html>
```

Run example »

Complete PHP HTTP Reference

For a complete reference of HTTP functions, go to our complete PHP HTTP Reference.

PHP 5 Sessions

« Previous

Next Chapter »

A session is a way to store information (in variables) to be used across multiple pages.

Unlike a cookie, the information is not stored on the users computer.

What is a PHP Session?

When you work 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 or what you do, because the HTTP address doesn't maintain state.

Session variables solve this problem by storing user information to be used across multiple pages (e.g. username, favorite color, etc). By default, session variables last until the user closes the browser.

So; Session variables hold information about one single user, and are available to all pages in one application.

> **Tip:** If you need a permanent storage, you may want to store the data in a [database

---

Start a PHP Session

A session is started with the session_start() function.

Session variables are set with the PHP global variable: $_SESSION.

Now, let's create a new page called "demo_session1.php". In this page, we start a new PHP session and set some session variables:

Example

```
<?php
// Start the session
session_start();
?>
```

```
<!DOCTYPE html>
<html>
<body>

<?php
// Set session variables
$_SESSION["favcolor"] = "green";
$_SESSION["favanimal"] = "cat";
echo "Session variables are set.";
?>

</body>
</html>
```

Run example »

Note: The session_start() function must be the very first thing in your document. Befor
tags.

Get PHP Session Variable Values

Next, we create another page called "demo_session2.php". From this page, we will access the session information we set on the first page ("demo_session1.php").

Notice that session variables are not passed individually to each new page, instead they are retrieved from the session we open at the beginning of each page (session_start()).

Also notice that all session variable values are stored in the global $_SESSION variable:

Example

```
<?php
session_start();
?>
<!DOCTYPE html>
```

```
<html>
<body>

<?php
// Echo session variables that were set on previous page
echo "Favorite color is " . $_SESSION["favcolor"] . ".<br>";
echo "Favorite animal is " . $_SESSION["favanimal"] . ".";
?>

</body>
</html>
```

Run example »

Another way to show all the session variable values for a user session is to run the following code:

Example

```
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
print_r($_SESSION);
?>

</body>
</html>
```

Run example »

How does it work? How does it know it's me?

Most sessions set a user-key on the user's computer that looks something like this: 765487cf34ert8dede5a562e4f3a7e12. Then, when a session is opened on another pa the computer for a user-key. If there is a match, it accesses that session, if not, it sta session.

Modify a PHP Session Variable

To change a session variable, just overwrite it:

Example

```
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
// to change a session variable, just overwrite it
$_SESSION["favcolor"] = "yellow";
print_r($_SESSION);
?>

</body>
</html>
```

Run example »

Destroy a PHP Session

To remove all global session variables and destroy the session, use session_unset() and session_destroy():

Example

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

```
<!DOCTYPE html>
<html>
<body>

<?php
// remove all session variables
session_unset();

// destroy the session
session_destroy();
?>

</body>
</html>
```

Run example »

PHP Filters

« Previous

Next Chapter »

Validating data = Determine if the data is in proper form.

Sanitizing data = Remove any illegal character from the data.

The PHP Filter Extension

PHP filters are used to validate and sanitize external input.

The PHP filter extension has many of the functions needed for checking user input, and is designed to make data validation easier and quicker.

The filter_list() function can be used to list what the PHP filter extension offers:

Example

```
<table>
  <tr>
    <td>Filter Name</td>
    <td>Filter ID</td>
  </tr>
  <?php
  foreach (filter_list() as $id =>$filter) {
      echo '<tr><td>' . $filter . '</td><td>' .
filter_id($filter) . '</td></tr>';
  }
  ?>
</table>
```

Run example »

Why Use Filters?

Many web applications receive external input. External input/data can be:

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

> **You should always validate external data!**
> Invalid submitted data can lead to security problems and break your webpage!
> By using PHP filters you can be sure your application gets the correct input!

PHP filter_var() Function

The filter_var() function both validate and sanitize data.

The filter_var() function filters a single variable with a specified filter. It takes two pieces of data:

- The variable you want to check
- The type of check to use

Sanitize a String

The following example uses the filter_var() function to remove all HTML tags from a string:

Example

```php
<?php
$str = "<h1>Hello World!</h1>";
$newstr = filter_var($str, FILTER_SANITIZE_STRING);
echo $newstr;
?>
```

Run example »

Validate an Integer

The following example uses the filter_var() function to check if the variable $int is an integer. If $int is an integer, the output of the code above will be: "Integer is valid". If $int is not an integer, the output will be: "Integer is not valid":

Example

```php
<?php
$int = 100;

if (!filter_var($int, FILTER_VALIDATE_INT) === false) {
```

```
    echo("Integer is valid");
} else {
    echo("Integer is not valid");
}
?>
```

Run example »

Tip: filter_var() and Problem With 0

In the example above, if $int was set to 0, the function above will return "Integer is not valid". To solve this problem, use the code below:

Example

```
<?php
$int = 0;

if (filter_var($int, FILTER_VALIDATE_INT)
=== 0 || !filter_var($int, FILTER_VALIDATE_INT) === false) {
    echo("Integer is valid");
} else {
    echo("Integer is not valid");
}
?>
```

Run example »

Validate an IP Address

The following example uses the filter_var() function to check if the variable $ip is a valid IP address:

Example

```php
<?php
$ip = "127.0.0.1";

if (!filter_var($ip, FILTER_VALIDATE_IP) === false) {
    echo("$ip is a valid IP address");
} else {
    echo("$ip is not a valid IP address");
}
?>
```

Run example »

Sanitize and Validate an Email Address

The following example uses the filter_var() function to first remove all illegal characters from the $email variable, then check if it is a valid email address:

Example

```php
<?php
$email = "john.doe@example.com";

// Remove all illegal characters from email
$email = filter_var($email, FILTER_SANITIZE_EMAIL);

// Validate e-mail
if (!filter_var($email, FILTER_VALIDATE_EMAIL) === false) {
    echo("$email is a valid email address");
} else {
    echo("$email is not a valid email address");
}
?>
```

Run example »

Sanitize and Validate a URL

The following example uses the filter_var() function to first remove all illegal characters from a URL, then check if $url is a valid URL:

Example

```php
<?php
$url = "http://www.w3schools.com";

// Remove all illegal characters from a url
$url = filter_var($url, FILTER_SANITIZE_URL);

// Validate url
if (!filter_var($url, FILTER_VALIDATE_URL) === false) {
    echo("$url is a valid URL");
} else {
    echo("$url is not a valid URL");
}
?>
```

Run example »

Complete PHP Filter Reference

For a complete reference of all filter functions, go to our complete PHP Filter Reference. Check each filter to see what options and flags are available.

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

PHP Filters Advanced

« Previous

Next Chapter »

Validate an Integer Within a Range

The following example uses the filter_var() function to check if a variable is both of type INT, and between 1 and 200:

Example

```php
<?php
$int = 122;
$min = 1;
$max = 200;

if (filter_var($int,
FILTER_VALIDATE_INT, array("options" =>array("min_range"=>$min,
 "max_range"=>$max))) === false) {
    echo("Variable value is not within the legal range");
} else {
    echo("Variable value is within the legal range");
}
?>
```

Run example »

Validate IPv6 Address

The following example uses the filter_var() function to check if the variable $ip is a valid IPv6 address:

Example

```php
<?php
$ip = "2001:0db8:85a3:08d3:1319:8a2e:0370:7334";

if (!filter_var($ip, FILTER_VALIDATE_IP, FILTER_FLAG_IPV6) ===
false) {
    echo("$ip is a valid IPv6 address");
```

```
} else {
    echo("$ip is not a valid IPv6 address");
}
?>
```

Run example »

Validate URL - Must Contain QueryString

The following example uses the filter_var() function to check if the variable $url is a URL with a querystring:

Example

```
<?php
$url = "http://www.w3schools.com";

if (!filter_var($url, FILTER_VALIDATE_URL,
FILTER_FLAG_QUERY_REQUIRED) === false) {
    echo("$url is a valid URL");
} else {
    echo("$url is not a valid URL");
}
?>
```

Run example »

Remove Characters With ASCII Value > 127

The following example uses the filter_var() function to sanitize a string. It will both remove all HTML tags, and all characters with ASCII value > 127, from the string:

Example

```
<?php
$str = "<h1>Hello WorldÆØÅ!</h1>";

$newstr = filter_var($str, FILTER_SANITIZE_STRING,
FILTER_FLAG_STRIP_HIGH);
echo $newstr;
?>
```

Run example »

Complete PHP Filter Reference

For a complete reference of all filter functions, go to our complete PHP Filter Reference. Check each filter to see what options and flags are available.

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

PHP Error Handling

« Previous

Next Chapter »

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 prevent the user from getting an error message like the one above, we test whether 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. M 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 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 scri halted
8	E_NOTICE	Run-time notices. The script found something that error, but could also happen when running a scrip
256	E_USER_ERROR	Fatal user-generated error. This is like an E_ERROR programmer using the PHP function trigger_error(
512	E_USER_WARNING	Non-fatal user-generated warning. This is like an set by the programmer using the PHP function trigg
1024	E_USER_NOTICE	User-generated notice. This is like an E_NOTICE programmer using the PHP function trigger_error(
4096	E_RECOVERABLE_ERROR	Catchable fatal error. This is like an E_ERROR b caught by a user defined handle (see also set_error_
8191	E_ALL	All errors and warnings (E_STRICT became a part o PHP 5.4)

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:

```
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 server's 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 error 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

« Previous

Next Chapter »

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 retrieves 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 handle errors that slip through.

Creating a Custom Exception Class

To create a custom exception handler you must 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 the exception thrown were of the class customException and there were 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 are of no interest 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 its 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.