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");
?>
```

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

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 if it doesn't 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 starts at the 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 exists

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 if it doesn't 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 starts at the 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 exists

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 don't want 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);
?>
```

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
<?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);
?>
```

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);
?>
```

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
```

PHP 5 File Upload

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:

```html
<!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 "upload.php" 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.";
    }
}
?>
```

PHP User Defined Functions

Besides the built-in PHP functions, we can create our own functions.

A function is a block of statements that can be used repeatedly in a program.

A function will not execute immediately when a page loads.

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

Create a User Defined Function in PHP

A user defined function declaration starts with the word "function":

Syntax

```
function functionName() {
    code to be executed;
}
```

Note: A function name can start with a letter or underscore (not a number).

Tip: Give the function a name that reflects what the function does!

Function names are NOT case-sensitive.

In the example below, we create a function named "writeMsg()". The opening curly brace ({) indicates the beginning of the function code and the closing curly brace (}) indicates the end of the function. The function outputs "Hello world!". To call the function, just write its name:

Example

```
<?php
function writeMsg() {
    echo "Hello world!";
}

writeMsg(); // call the function
?>
```

PHP Function Arguments

Information can be passed to functions through arguments. An argument is just like a variable.

Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.

The following example has a function with one argument ($fname). When the familyName() function is called, we also pass along a name (e.g. Jani), and the name is used inside the function, which outputs several different first names, but an equal last name:

Example

```php
<?php
function familyName($fname) {
    echo "$fname Refsnes.<br>";
}

familyName("Jani");
familyName("Hege");
familyName("Stale");
familyName("Kai Jim");
familyName("Borge");
?>
```

The following example has a function with two arguments ($fname and $year):

Example

```php
<?php
function familyName($fname, $year) {
    echo "$fname Refsnes. Born in $year <br>";
}

familyName("Hege", "1975");
familyName("Stale", "1978");
familyName("Kai Jim", "1983");
?>
```

PHP Default Argument Value

The following example shows how to use a default parameter. If we call the function setHeight() without arguments it takes the default value as argument:

Example

```php
<?php
function setHeight($minheight = 50) {
    echo "The height is : $minheight <br>";
}

setHeight(350);
setHeight(); // will use the default value of 50
setHeight(135);
setHeight(80);
?>
```

PHP Functions - Returning values

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

Example

```
<?php
function sum($x, $y) {
    $z = $x + $y;
    return $z;
}

echo "5 + 10 = " . sum(5, 10) . "<br>";
echo "7 + 13 = " . sum(7, 13) . "<br>";
echo "2 + 4 = " . sum(2, 4);
?>
```

What is an Array?

An array is a special variable, which can hold more than one value at a time.

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

```
$cars1 = "Volvo";
$cars2 = "BMW";
$cars3 = "Toyota";
```

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

The solution is to create an array!

An array can hold many values under a single name, and you can access the values by referring to an index number.

Create an Array in PHP

In PHP, the array() function is used to create an array:

```
array();
```

In PHP, there are three types of arrays:

- **Indexed arrays** - Arrays with a numeric index
- **Associative arrays** - Arrays with named keys
- **Multidimensional arrays** - Arrays containing one or more arrays

PHP Indexed Arrays

There are two ways to create indexed arrays:

The index can be assigned automatically (index always starts at 0), like this:

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

or the index can be assigned manually:

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

The following example creates an indexed array named $cars, assigns three elements to it, and then prints a text containing the array values:

Example

```
<?php
$cars = array("Volvo", "BMW", "Toyota");
echo "I like " . $cars[0] . ", " . $cars[1] . " and " . $cars[2] . ".";
?>
```

Get The Length of an Array - The count() Function

The count() function is used to return the length (the number of elements) of an array:

Example

```
<?php
$cars = array("Volvo", "BMW", "Toyota");
echo count($cars);
?>
```

Loop Through an Indexed Array

To loop through and print all the values of an indexed array, you could use a for loop, like this:

Example

```
<?php
$cars = array("Volvo", "BMW", "Toyota");
$arrlength = count($cars);

for($x = 0; $x < $arrlength; $x++) {
    echo $cars[$x];
    echo "<br>";
}
?>
```

PHP Associative Arrays

Associative arrays are arrays that use named keys that you assign to them.

There are two ways to create an associative array:

```
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
```

or:

```
$age['Peter'] = "35";
$age['Ben'] = "37";
$age['Joe'] = "43";
```

The named keys can then be used in a script:

Example

```php
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
echo "Peter is " . $age['Peter'] . " years old.";
?>
```

Loop Through an Associative Array

To loop through and print all the values of an associative array, you could use a foreach loop, like this:

Example

```php
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");

foreach($age as $x => $x_value) {
    echo "Key=" . $x . ", Value=" . $x_value;
    echo "<br>";
}
?>
```

PHP - Sort Functions For Arrays

In this chapter, we will go through the following PHP array sort functions:

- sort() - sort arrays in ascending order
- rsort() - sort arrays in descending order
- asort() - sort associative arrays in ascending order, according to the value
- ksort() - sort associative arrays in ascending order, according to the key
- arsort() - sort associative arrays in descending order, according to the value
- krsort() - sort associative arrays in descending order, according to the key

Sort Array in Ascending Order - sort()

The following example sorts the elements of the $cars array in ascending alphabetical order:

Example

```php
<?php
$cars = array("Volvo", "BMW", "Toyota");
sort($cars);
?>
```

The following example sorts the elements of the $numbers array in ascending numerical order:

Example

```php
<?php
$numbers = array(4, 6, 2, 22, 11);
sort($numbers);
?>
```

Sort Array in Descending Order - rsort()

The following example sorts the elements of the $cars array in descending alphabetical order:

Example

```php
<?php
$cars = array("Volvo", "BMW", "Toyota");
rsort($cars);
?>
```

The following example sorts the elements of the $numbers array in descending numerical order:

Example

```php
<?php
$numbers = array(4, 6, 2, 22, 11);
rsort($numbers);
?>
```

Sort Array (Ascending Order), According to Value - asort()

The following example sorts an associative array in ascending order, according to the value:

Example

```php
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
asort($age);
?>
```

Sort Array (Ascending Order), According to Key - ksort()

The following example sorts an associative array in ascending order, according to the key:

Example

```php
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
ksort($age);
?>
```

Sort Array (Descending Order), According to Value - arsort()

The following example sorts an associative array in descending order, according to the value:

Example

```php
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
arsort($age);
?>
```

Sort Array (Descending Order), According to Key - krsort()

The following example sorts an associative array in descending order, according to the key:

Example

```php
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
krsort($age);
?>
```

PHP $_SERVER

$_SERVER is a PHP super global variable which holds information about headers, paths, and script locations.

The example below shows how to use some of the elements in $_SERVER:

Example

```php
<?php
echo $_SERVER['PHP_SELF'];
echo "<br>";
echo $_SERVER['SERVER_NAME'];
echo "<br>";
echo $_SERVER['HTTP_HOST'];
echo "<br>";
echo $_SERVER['HTTP_REFERER'];
echo "<br>";
echo $_SERVER['HTTP_USER_AGENT'];
echo "<br>";
echo $_SERVER['SCRIPT_NAME'];
?>
```

The following table lists the most important elements that can go inside $_SERVER:

Element/Code	**Description**
$_SERVER['PHP_SELF']	Returns the filename of the currently executing script
$_SERVER['GATEWAY_INTERFACE']	Returns the version of the Common Gateway Interface (CGI) the server is using
$_SERVER['SERVER_ADDR']	Returns the IP address of the host server
$_SERVER['SERVER_NAME']	Returns the name of the host server (such as www.w3schools.com)

$_SERVER['SERVER_SOFTWARE']	Returns the server identification string (such as Apache/2.2.24)
$_SERVER['SERVER_PROTOCOL']	Returns the name and revision of the information protocol (such as HTTP/1.1)
$_SERVER['REQUEST_METHOD']	Returns the request method used to access the page (such as POST)
$_SERVER['REQUEST_TIME']	Returns the timestamp of the start of the request (such as 1377687496)
$_SERVER['QUERY_STRING']	Returns the query string if the page is accessed via a query string
$_SERVER['HTTP_ACCEPT']	Returns the Accept header from the current request
$_SERVER['HTTP_ACCEPT_CHARSET']	Returns the Accept_Charset header from the current request (such as utf-8,ISO-8859-1)
$_SERVER['HTTP_HOST']	Returns the Host header from the current request
$_SERVER['HTTP_REFERER']	Returns the complete URL of the current page (not reliable because not all user-agents support it)
$_SERVER['HTTPS']	Is the script queried through a secure HTTP protocol

$_SERVER['REMOTE_ADDR']	Returns the IP address from where the user is viewing the current page
$_SERVER['REMOTE_HOST']	Returns the Host name from where the user is viewing the current page
$_SERVER['REMOTE_PORT']	Returns the port being used on the user's machine to communicate with the web server
$_SERVER['SCRIPT_FILENAME']	Returns the absolute pathname of the currently executing script
$_SERVER['SERVER_ADMIN']	Returns the value given to the SERVER_ADMIN directive in the web server configuration file (if your script runs on a virtual host, it will be the value defined for that virtual host) (such as someone@w3schools.com)
$_SERVER['SERVER_PORT']	Returns the port on the server machine being used by the web server for communication (such as 80)
$_SERVER['SERVER_SIGNATURE']	Returns the server version and virtual host name which are added to server-generated pages
$_SERVER['PATH_TRANSLATED']	Returns the file system based path to the current script
$_SERVER['SCRIPT_NAME']	Returns the path of the current script

$_SERVER['SCRIPT_URI']	Returns the URI of the current page

PHP $_REQUEST

PHP $_REQUEST is used to collect data after submitting an HTML form.

The example below shows a form with an input field and a submit button. When a user submits the data by clicking on "Submit", the form data is sent to the file specified in the action attribute of the <form> tag. In this example, we point to this file itself for processing form data. If you wish to use another PHP file to process form data, replace that with the filename of your choice. Then, we can use the super global variable $_REQUEST to collect the value of the input field:

Example

```
<html>
<body>

<form method="post" action="<?php echo $_SERVER['PHP_SELF'];?>">
  Name: <input type="text" name="fname">
  <input type="submit">
</form>

<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    // collect value of input field
    $name = $_REQUEST['fname'];
    if (empty($name)) {
        echo "Name is empty";
    } else {
        echo $name;
    }
}
?>

</body>
</html>
```

PHP $_POST

PHP $_POST is widely used to collect form data after submitting an HTML form with method="post". $_POST is also widely used to pass variables.

The example below shows a form with an input field and a submit button. When a user submits the data by clicking on "Submit", the form data is sent to the file specified in the action attribute of the <form> tag. In this example, we point to the file itself for processing form data. If you wish to use another PHP file to process form data, replace that with the filename of your choice. Then, we can use the super global variable $_POST to collect the value of the input field:

Example

```
<html>
<body>

<form method="post" action="<?php echo $_SERVER['PHP_SELF'];?>">
  Name: <input type="text" name="fname">
  <input type="submit">
</form>

<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    // collect value of input field
    $name = $_POST['fname'];
    if (empty($name)) {
        echo "Name is empty";
    } else {
        echo $name;
    }
}
?>

</body>
</html>
```

PHP $_GET

PHP $_GET can also be used to collect form data after submitting an HTML form with method="get".

$_GET can also collect data sent in the URL.

Assume we have an HTML page that contains a hyperlink with parameters:

```
<html>
<body>

<a href="test_get.php?subject=PHP&web=W3schools.com">Test $GET</a>

</body>
</html>
```

When a user clicks on the link "Test $GET", the parameters "subject" and "web" are sent to "test_get.php", and you can then access their values in "test_get.php" with $_GET.

The example below shows the code in "test_get.php":

Example

```
<html>
<body>

<?php
echo "Study " . $_GET['subject'] . " at " . $_GET['web'];
?>

</body>
</html>
```

PHP header() Function

Definition and Usage

The header() function sends a raw HTTP header to a client.

It is important to notice that header() must be called before any actual output is sent (In PHP 4 and later, you can use output buffering to solve this problem):

```
<html>
<?php
// This results in an error.
// The output above is before the header() call
header('Location: http://www.example.com/');
?>
```

Syntax

```
header(string,replace,http_response_code)
```

Parameter	Description
string	Required. Specifies the header string to send
replace	Optional. Indicates whether the header should replace previous or add a second header. Default is TRUE (will replace). FALSE (allows multiple headers of the same type)
http_response_code	Optional. Forces the HTTP response code to the specified value (available in PHP 4.3 and higher)

Tips and Notes

Note: Since PHP 4.4 this function prevents more than one header to be sent at once. This is a protection against header injection attacks.

Example 1

Prevent page caching:

```
<?php
// Date in the past
header("Expires: Mon, 26 Jul 1997 05:00:00 GMT");
header("Cache-Control: no-cache");
header("Pragma: no-cache");
?>

<html>
<body>

...
...
```

Note: There are options that users may set to change the browser's default caching settings. By sending the headers above, you should override any of those settings and force the browser to not cache!

Example 2

Let the user be prompted to save a generated PDF file (Content-Disposition header is used to supply a recommended filename and force the browser to display the save dialog box):

```
<?php
header("Content-type:application/pdf");

// It will be called downloaded.pdf
header("Content-Disposition:attachment;filename='downloaded.pdf'");

// The PDF source is in original.pdf
readfile("original.pdf");
?>

<html>
<body>

...
...
```

Note: There is a bug in Microsoft IE 5.5 that prevents this from working. The bug can be resolved by upgrading to Service Pack 2 or later.

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"]);
  }
}
?>
```

PHP Filters

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>
```

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

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
$int = 100;

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

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");
}
?>
```

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
$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");
}
?>
```

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
$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");
}
?>
```

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 = "https://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");
}
?>
```

PHP 5 Sessions

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>
```

Note: The session_start() function must be the very first thing in your document. Before any HTML 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>
```

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>
```

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 page, it scans the computer for a user-key. If there is a match, it accesses that session, if not, it starts a new 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>
```

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>
```

PHP 5 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.

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>
```

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>
```

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>
```

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>
```

PHP 5 String Functions

The PHP string functions are part of the PHP core. No installation is required to use these functions.

Function	Description
addcslashes()	Returns a string with backslashes in front of the specified characters
addslashes()	Returns a string with backslashes in front of predefined characters
bin2hex()	Converts a string of ASCII characters to hexadecimal values
chop()	Removes whitespace or other characters from the right end of a string
chr()	Returns a character from a specified ASCII value
chunk_split()	Splits a string into a series of smaller parts
convert_cyr_string()	Converts a string from one Cyrillic character-set to another
convert_uudecode()	Decodes a uuencoded string
convert_uuencode()	Encodes a string using the uuencode algorithm
count_chars()	Returns information about characters used in a string
crc32()	Calculates a 32-bit CRC for a string
crypt()	One-way string hashing
echo()	Outputs one or more strings
explode()	Breaks a string into an array
fprintf()	Writes a formatted string to a specified output stream
get_html_translation_table()	Returns the translation table used by htmlspecialchars() and htmlentities()
hebrev()	Converts Hebrew text to visual text
hebrevc()	Converts Hebrew text to visual text and new lines (\n) into
hex2bin()	Converts a string of hexadecimal values to ASCII characters
html_entity_decode()	Converts HTML entities to characters
htmlentities()	Converts characters to HTML entities
htmlspecialchars_decode()	Converts some predefined HTML entities to characters
htmlspecialchars()	Converts some predefined characters to HTML entities
implode()	Returns a string from the elements of an array
join()	Alias of implode()
lcfirst()	Converts the first character of a string to lowercase
levenshtein()	Returns the Levenshtein distance between two strings
localeconv()	Returns locale numeric and monetary formatting information
ltrim()	Removes whitespace or other characters from the left side of a string
md5()	Calculates the MD5 hash of a string
md5_file()	Calculates the MD5 hash of a file
metaphone()	Calculates the metaphone key of a string
money_format()	Returns a string formatted as a currency string
nl_langinfo()	Returns specific local information
nl2br()	Inserts HTML line breaks in front of each newline in a string
number_format()	Formats a number with grouped thousands
ord()	Returns the ASCII value of the first character of a string
parse_str()	Parses a query string into variables
print()	Outputs one or more strings
printf()	Outputs a formatted string
quoted_printable_decode()	Converts a quoted-printable string to an 8-bit string
quoted_printable_encode()	Converts an 8-bit string to a quoted printable string
quotemeta()	Quotes meta characters
rtrim()	Removes whitespace or other characters from the right side of a string
setlocale()	Sets locale information
sha1()	Calculates the SHA-1 hash of a string
sha1_file()	Calculates the SHA-1 hash of a file
similar_text()	Calculates the similarity between two strings

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

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

PHP mail() Function

Example

Send a simple email:

```
<?php
// the message
$msg = "First line of text\nSecond line of text";

// use wordwrap() if lines are longer than 70 characters
$msg = wordwrap($msg,70);

// send email
mail("someone@example.com","My subject",$msg);
?>
```

Definition and Usage

The mail() function allows you to send emails directly from 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. **Windows note:** If a full stop is found on the beginning of a line in the message, it might be removed. To solve this problem, replace the full stop with a double dot: <?php $txt = str_replace("\n.", "\n..", $txt); ?>
headers	Optional. Specifies additional headers, like From, Cc, and Bcc. The additional headers should be separated with a CRLF (\r\n). **Note:** When sending an email, it must contain a From header. This can be set with this parameter or in the php.ini file.
parameters	Optional. Specifies an additional parameter to the sendmail program (the one defined in the sendmail_path configuration setting). (i.e. this can be used to set the envelope sender address when using sendmail with the -f sendmail option)

Technical Details

Return Value:	Returns the hash value of the *address* parameter, or FALSE on failure. **Note:** Keep in mind that even if the email was accepted for delivery, it does NOT mean the email is actually sent and received!
PHP Version:	4+
PHP Changelog:	PHP 4.3.0: (Windows only) All custom headers (like From, Cc, Bcc and Date) are supported, and are not case-sensitive. PHP 4.2.3: The *parameter* parameter is disabled in safe mode PHP 4.0.5: The *parameter* parameter was added

Example 2

Send an email with extra headers:

```
<?php
$to = "somebody@example.com";
$subject = "My subject";
$txt = "Hello world!";
$headers = "From: webmaster@example.com" . "\r\n" .
"CC: somebodyelse@example.com";

mail($to,$subject,$txt,$headers);
?>
```

Example 3

Send an HTML email:

```
<?php
$to = "somebody@example.com, somebodyelse@example.com";
$subject = "HTML email";

$message = "
<html>
<head>
<title>HTML email</title>
</head>
<body>
<p>This email contains HTML Tags!</p>
<table>
<tr>
<th>Firstname</th>
<th>Lastname</th>
</tr>
<tr>
<td>John</td>
<td>Doe</td>
</tr>
</table>
</body>
</html>
";
```

```
// Always set content-type when sending HTML email
$headers = "MIME-Version: 1.0" . "\r\n";
$headers .= "Content-type:text/html;charset=UTF-8" . "\r\n";

// More headers
$headers .= 'From: <webmaster@example.com>' . "\r\n";
$headers .= 'Cc: myboss@example.com' . "\r\n";

mail($to,$subject,$message,$headers);
?>
```

pathinfo

(PHP 4 >= 4.0.3, PHP 5, PHP 7)

pathinfo — Returns information about a file path

Description ¶

```
mixed pathinfo ( string $path [, int $options = PATHINFO_DIRNAME |
PATHINFO_BASENAME | PATHINFO_EXTENSION | PATHINFO_FILENAME ] )
```

pathinfo() returns information about `path`: either an associative array or a string, depending on `options`.

Note:
For information on retrieving the current path info, read the section on predefined reserved variables.
Caution

pathinfo() is locale aware, so for it to parse a path containing multibyte characters correctly, the matching locale must be set using the setlocale() function.

Parameters ¶

`path`

The path to be parsed.

`options`

If present, specifies a specific element to be returned; one of **`PATHINFO_DIRNAME`**, **`PATHINFO_BASENAME`**,**`PATHINFO_EXTENSION`** or **`PATHINFO_FILENAME`**.

If `options` is not specified, returns all available elements.

Return Values ¶

If the `options` parameter is not passed, an associative array containing the following elements is returned:*dirname*, *basename*, *extension* (if any), and *filename*.

Note:
If the `path` has more than one extension, **`PATHINFO_EXTENSION`** returns only the last one and**`PATHINFO_FILENAME`** only strips the last one. (see first example below).
Note:
If the `path` does not have an extension, no *extension* element will be returned (see second example below).
Note:
If the *basename* of the `path` starts with a dot, the following characters are interpreted as *extension*, and the *filename* is empty (see third example below).

If `options` is present, returns a string containing the requested element.

Changelog ¶

Version	Description
5.2.0	The **`PATHINFO_FILENAME`** constant was added.

Examples ¶

Example #1 pathinfo() Example

```
<?php
$path_parts = pathinfo('/www/htdocs/inc/lib.inc.php');

echo $path_parts['dirname'], "\n";
echo $path_parts['basename'], "\n";
echo $path_parts['extension'], "\n";
echo $path_parts['filename'], "\n"; // since PHP 5.2.0
?>
```

The above example will output:

```
/www/htdocs/inc
lib.inc.php
php
lib.inc
```

basename

(PHP 4, PHP 5, PHP 7)

basename — Returns trailing name component of path

Description ¶

```
string basename ( string $path [, string $suffix ] )
```

Given a string containing the path to a file or directory, this function will return the trailing name component.

Note:
basename() operates naively on the input string, and is not aware of the actual filesystem, or path components such as "..".
Caution

basename() is locale aware, so for it to see the correct basename with multibyte character paths, the matching locale must be set using the setlocale() function.

Parameters ¶

`path`

A path.
On Windows, both slash (/) and backslash (\) are used as directory separator character. In other environments, it is the forward slash (/).

`suffix`

If the name component ends in **`suffix`** this will also be cut off.

Return Values ¶

Returns the base name of the given **`path`**.

Examples ¶

Example #1 basename() example

```php
<?php
echo "1) ".basename("/etc/sudoers.d", ".d").PHP_EOL;
echo "2) ".basename("/etc/sudoers.d").PHP_EOL;
echo "3) ".basename("/etc/passwd").PHP_EOL;
echo "4) ".basename("/etc/").PHP_EOL;
echo "5) ".basename(".").PHP_EOL;
echo "6) ".basename("/");
?>
```

The above example will output:

```
1) sudoers
2) sudoers.d
3) passwd
4) etc
5) .
6)
```

getimagesize

(PHP 4, PHP 5, PHP 7)

getimagesize — Get the size of an image

Description ¶

```
array getimagesize ( string $filename [, array &$imageinfo ] )
```

The getimagesize() function will determine the size of any supported given image file and return the dimensions along with the file type and a *height/width* text string to be used inside a normal HTML `IMG` tag and the correspondent HTTP content type.

getimagesize() can also return some more information in **`imageinfo`** parameter.

Note: Note that JPC and JP2 are capable of having components with different bit depths. In this case, the value for "bits" is the highest bit depth encountered. Also, JP2 files may contain *multiple JPEG 2000 codestreams*. In this case, getimagesize() returns the values for the first codestream it encounters in the root of the file.
Note: The information about icons are retrieved from the icon with the highest bitrate.

Parameters ¶

`filename`

This parameter specifies the file you wish to retrieve information about. It can reference a local file or (configuration permitting) a remote file using one of the supported streams.

`imageinfo`

This optional parameter allows you to extract some extended information from the image file. Currently, this will return the different JPG APP markers as an associative array. Some programs use these APP markers to embed text information in images. A very common one is to embed » IPTC information in the APP13 marker. You can use the iptcparse() function to parse the binary APP13 marker into something readable.

Return Values ¶

Returns an array with up to 7 elements. Not all image types will include the *channels* and *bits* elements.

Index 0 and 1 contains respectively the width and the height of the image.

Note:
Some formats may contain no image or may contain multiple images. In these cases, getimagesize() might not be able to properly determine the image size. getimagesize() will return zero for width and height in these cases.

Index 2 is one of the IMAGETYPE_XXX constants indicating the type of the image.

Index 3 is a text string with the correct *height="yyy" width="xxx"* string that can be used directly in an IMG tag.

mime is the correspondant MIME type of the image. This information can be used to deliver images with the correct HTTP *Content-type* header:

Example #1 getimagesize() and MIME types

```
<?php
$size = getimagesize($filename);
$fp = fopen($filename, "rb");
if ($size && $fp) {
    header("Content-type: {$size['mime']}");
    fpassthru($fp);
    exit;
} else {
    // error
}
?>
```

channels will be 3 for RGB pictures and 4 for CMYK pictures.

bits is the number of bits for each color.

For some image types, the presence of *channels* and *bits* values can be a bit confusing. As an example, GIF always uses 3 channels per pixel, but the number of bits per pixel cannot be calculated for an animated GIF with a global color table.

On failure, **`FALSE`** is returned.

Errors/Exceptions ¶

If accessing the **`filename`** image is impossible getimagesize() will generate an error of level **`E_WARNING`**. On read error, getimagesize() will generate an error of level **`E_NOTICE`**.

Changelog ¶

Version	Description
7.1.0	Added WebP support.
5.3.0	Added icon support.
5.2.3	Read errors generated by this function downgraded to **`E_NOTICE`** from **`E_WARNING`**.
4.3.2	Support for JPC, JP2, JPX, JB2, XBM, and WBMP became available.
4.3.2	JPEG 2000 support was added for the **`imageinfo`** parameter.
4.3.0	*bits* and *channels* are present for other image types, too.
4.3.0	Support for SWC and IFF was added.
4.2.0	Support for TIFF was added.
4.0.6	Support for BMP and PSD was added.

Examples ¶

Example #2 getimagesize() example

```
<?php
list($width, $height, $type, $attr) = getimagesize("img/flag.jpg");
echo "<img src=\"img/flag.jpg\" $attr alt=\"getimagesize() example\" />";
?>
```

Example #3 getimagesize (URL)

```
<?php
$size = getimagesize("http://www.example.com/gifs/logo.gif");

// if the file name has space in it, encode it properly
$size = getimagesize("http://www.example.com/gifs/lo%20go.gif");

?>
```

Example #4 getimagesize() returning IPTC

```
<?php
$size = getimagesize("testimg.jpg", $info);
if (isset($info["APP13"])) {
    $iptc = iptcparse($info["APP13"]);
    var_dump($iptc);
}
?>
```

Check if the file type is allowed

We only want to allow .png files to be uploaded, so we're going to have to make sure that the type is image/png, again simply:

```
if($_FILES['file_upload']['type'] != 'image/png'){

	die('Unsupported filetype uploaded.');

}
```

Check that the file is under our file size limit

Sizes are all based in bytes, so if we don't want any files bigger than 500kb uploading, we'll have to check that the size is less than 500000:

```
if($_FILES['file_upload']['size'] > 500000){

	die('File uploaded exceeds maximum upload size.');

}
```

Check that the file doesn't already exist (based on name)

We're going to be putting our files in a directory called upload. You need to make sure that you CHMOD this directory to have the required permissions for writing. We will simply use the PHP function file_exists():

```
if(file_exists('upload/' . $_FILES['file_upload']['name'])){

	die('File with that name already exists.');

}
```

Finally upload the file. Finally

we want to upload the file, as at this point if the script is still running the file is safe to upload. We will simply move the file using the move_uploaded_file function from its current temporary location to where we want it:

```
if(!move_uploaded_file($_FILES['file_upload']['tmp_name'], 'upload/' . $_FILES['file_upload']['name'])){

	die('Error uploading file - check destination is writeable.');

}
```