

PHP - GET & POST Methods

There are two ways the browser client can send information to the web server.

- The GET Method
- The POST Method

Before the browser sends the information, it encodes it using a scheme called URL encoding. In this scheme, name/value pairs are joined with equal signs and different pairs are separated by the ampersand.

```
name1=value1&name2=value2&name3=value3
```

Spaces are removed and replaced with the + character and any other nonalphanumeric characters are replaced with a hexadecimal values. After the information is encoded it is sent to the server.

The GET Method

The GET method sends the encoded user information appended to the page request. The page and the encoded information are separated by the ? character.

```
http://www.test.com/index.htm?name1=value1&name2=value2
```

- The GET method produces a long string that appears in your server logs, in the browser's Location: box.
- The GET method is restricted to send upto 1024 characters only.
- Never use GET method if you have password or other sensitive information to be sent to the server.
- GET can't be used to send binary data, like images or word documents, to the server.
- The data sent by GET method can be accessed using QUERY_STRING environment variable.
- The PHP provides **\$_GET** associative array to access all the sent information using GET method.

Try out following example by putting the source code in test.php script.

```
<?php
    if( $_GET["name"] || $_GET["age"] ) {
        echo "Welcome ". $_GET['name']. "<br />";
        echo "You are ". $_GET['age']. " years old.";
```

```

        exit();
    }
?>
<html>
    <body>

        <form action = "<?php $_PHP_SELF ?>" method = "GET">
            Name: <input type = "text" name = "name" />
            Age: <input type = "text" name = "age" />
            <input type = "submit" />
        </form>

    </body>
</html>

```

It will produce the following result –

The image shows a rendered HTML form. It consists of a rectangular container with a light gray border. Inside, the text "Name:" is followed by a text input field. To the right of this is the text "Age:" followed by another text input field. To the right of the second input field is a button labeled "Submit".

The POST Method

The POST method transfers information via HTTP headers. The information is encoded as described in case of GET method and put into a header called QUERY_STRING.

- The POST method does not have any restriction on data size to be sent.
- The POST method can be used to send ASCII as well as binary data.
- The data sent by POST method goes through HTTP header so security depends on HTTP protocol. By using Secure HTTP you can make sure that your information is secure.
- The PHP provides **\$_POST** associative array to access all the sent information using POST method.

Try out following example by putting the source code in test.php script.

```

<?php
    if( $_POST["name"] || $_POST["age"] ) {
        if (preg_match("/^[A-Za-z'-]$/", $_POST['name'] )) {
            die ("invalid name and name should be alpha");
        }
        echo "Welcome ". $_POST['name']. "<br />";
        echo "You are ". $_POST['age']. " years old.";
    }

```

```

        exit();
    }
?>
<html>
    <body>

        <form action = "<?php $_PHP_SELF ?>" method = "POST">
            Name: <input type = "text" name = "name" />
            Age: <input type = "text" name = "age" />
            <input type = "submit" />
        </form>

    </body>
</html>

```

It will produce the following result –



The screenshot shows a simple web form. It contains two text input fields. The first field is labeled "Name:" and the second is labeled "Age:". To the right of the "Age:" field is a button labeled "Submit". The entire form is enclosed in a light gray border.

The \$_REQUEST variable

The PHP \$_REQUEST variable contains the contents of both \$_GET, \$_POST, and \$_COOKIE. We will discuss \$_COOKIE variable when we will explain about cookies.

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

Try out following example by putting the source code in test.php script.

```

<?php
    if( $_REQUEST["name"] || $_REQUEST["age"] ) {
        echo "Welcome ". $_REQUEST['name']. "<br />";
        echo "You are ". $_REQUEST['age']. " years old.";
        exit();
    }
?>
<html>
    <body>

        <form action = "<?php $_PHP_SELF ?>" method = "POST">
            Name: <input type = "text" name = "name" />
            Age: <input type = "text" name = "age" />

```

```
        <input type = "submit" />
    </form>

    </body>
</html>
```

Here `$_PHP_SELF` variable contains the name of self script in which it is being called.

It will produce the following result –



The screenshot shows a simple web form. It consists of two text input fields. The first field is labeled "Name:" and the second is labeled "Age:". To the right of these fields is a button labeled "Submit". The entire form is enclosed in a thin border.

Cookies

Cookies are text files stored on the client computer and they are kept of use tracking purpose. PHP transparently supports HTTP cookies.

There are three steps involved in identifying returning users –

- Server script sends a set of cookies to the browser. For example name, age, or identification number etc.
- Browser stores this information on local machine for future use.
- When next time browser sends any request to web server then it sends those cookies information to the server and server uses that information to identify the user.

Setting Cookies with PHP

PHP provided **setcookie()** function to set a cookie. This function requires upto six arguments and should be called before `<html>` tag. For each cookie this function has to be called separately.

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

Here is the detail of all the arguments –

- **Name** – This sets the name of the cookie and is stored in an environment variable called HTTP_COOKIE_VARS. This variable is used while accessing cookies.
- **Value** – This sets the value of the named variable and is the content that you actually want to store.
- **Expiry** – This specifies a future time in seconds since 00:00:00 GMT on 1st Jan 1970. After this time cookie will become inaccessible. If this parameter is not set then cookie will automatically expire when the Web Browser is closed.
- **Path** – This specifies the directories for which the cookie is valid. A single forward slash character permits the cookie to be valid for all directories.
- **Domain** – This can be used to specify the domain name in very large domains and must contain at least two periods to be valid. All cookies are only valid for the host and domain which created them.
- **Security** – This can be set to 1 to specify that the cookie should only be sent by secure transmission using HTTPS otherwise set to 0 which mean cookie can be sent by regular HTTP.

Following example will create two cookies **name** and **age** these cookies will be expired after one hour.

```
<?php
    setcookie("name", "John Watkin", time()+3600,
"/", "", 0);
    setcookie("age", "36", time()+3600, "/", "", 0);
?>
<html>

    <head>
        <title>Setting Cookies with PHP</title>
    </head>

    <body>
        <?php echo "Set Cookies"?>
    </body>

</html>
```

Accessing Cookies with PHP

PHP provides many ways to access cookies. Simplest way is to use either `$_COOKIE` or `$HTTP_COOKIE_VARS` variables. Following example will access all the cookies set in above example.

```
<html>

  <head>
    <title>Accessing Cookies with PHP</title>
  </head>

  <body>

    <?php
      echo $_COOKIE["name"] . "<br />";

      /* is equivalent to */
      echo $HTTP_COOKIE_VARS["name"] . "<br />";

      echo $_COOKIE["age"] . "<br />";

      /* is equivalent to */
      echo $HTTP_COOKIE_VARS["age"] . "<br />";
    ?>

  </body>
</html>
```

You can use **isset()** function to check if a cookie is set or not.

```
<html>

  <head>
    <title>Accessing Cookies with PHP</title>
  </head>

  <body>

    <?php
      if ( isset($_COOKIE["name"]))
```

```

        echo "Welcome " . $_COOKIE["name"] . "<br
/>";

        else
            echo "Sorry... Not recognized" . "<br />";
    ?>

</body>
</html>

```

Deleting Cookie with PHP

Officially, to delete a cookie you should call `setcookie()` with the name argument only but this does not always work well, however, and should not be relied on.

It is safest to set the cookie with a date that has already expired –

```

<?php
    setcookie( "name", "", time()- 60, "/", "", 0);
    setcookie( "age", "", time()- 60, "/", "", 0);
?>
<html>

    <head>
        <title>Deleting Cookies with PHP</title>
    </head>

    <body>
        <?php echo "Deleted Cookies" ?>
    </body>

</html>

```

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["favourite"] = "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["fanimal"] . ".";
?>

</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>
```

File Handling

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

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

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

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);
?>
```

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 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 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
$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
$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
```

Exception Handling

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.

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.

