PHP - GET & POST

Since PHP is mostly used for web application development, the data sent by the browser client is mainly with the GET and POST types of HTTP request methods. The HTTP protocol also defines other methods for sending the request to the server. They are PUT, DELETE, HEAD and OPTIONS (in addition to GET and POST methods). In this chapter, we shall concentrate on how PHP handles the GET and POST methods.

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.

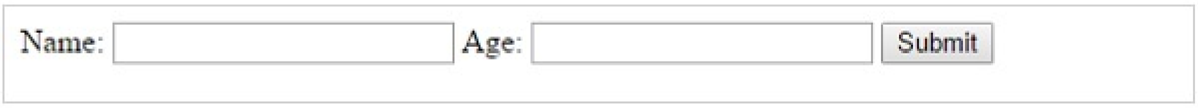
Try out following example by putting the source code in test.php script.

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

      exit();
   }
?>
<form action = "<?php <b>$_PHP_SELF</b> ?>" method = "GET">
```

```
    Name: <input type = "text" name = "name" />
    Age: <input type = "text" name = "age" />
    <input type = "submit" />
</form>
```

It will produce the following result −



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.

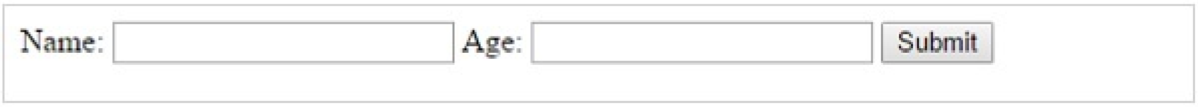
- The POST method does not have any restriction on data size to be sent.
- The POST method can be used to send ASCII as well as binary data.
- The data sent by POST method goes through HTTP header so security depends on HTTP protocol. By using Secure HTTP you can make sure that your information is secure.
- 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();
   }
?>
<form action = "<?php <b>$_PHP_SELF</b> ?>" method = "POST">
   Name: <input type = "text" name = "name" />
   Age: <input type = "text" name = "age" />
   <input type = "submit" />
</form>
```

It will produce the following result −



Explore our latest online courses and learn new skills at your own pace. Enroll and become a certified expert to boost your career.

Difference between GET and POST

The main difference between the GET and POST methods is that while the request parameters appended to the URL are exposed in the browser's URL, the POST data is included in the message body, and not revealed in the URL. Hence, the GET method shouldn't be used to send sensitive data to the server.

Secondly, the request data in GET method cannot exceed 2048 characters and can consist of ASCII characters only, while with POST method, there is no limit to the request data which can be in binary also (the default maximum size of POST data is determined by post_max_size setting in php.ini file)

PHP provides the following three **superglobals** to retrieve and process the request parameters −

- **$_GET** − an associative array to access all the sent information using GET method.
- **$_POST** − an associative array to access all the sent information using POST method.
- **$_REQUEST** − an associative array that can be used to get the result from form data sent with both the GET and POST methods.

$_GET Array

You can pass the request parameters in the form of query string directly appended to the URL.

Save the following PHP script in the document root folder (**htdocs**) as "hello.php" −

```php
<?php
   echo "First name: " . $_REQUEST['first_name'] . " " .
      "Last Name: " . $_REQUEST['last_name'] . "";
?>
```

Enter **http://localhost/hello.php?first_name=Amar&last_name=Sharma** as the URL in a browser window (ensure that the PHP server is running).

The $_GET array is populated from the request and the output is displayed as below −

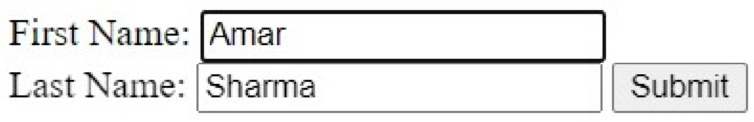
```
First name: Amar Last Name: Sharma
```

You can also populate the $_GET array with the HTML form data if its method attribute is GET.

Use the following HTML form to collect the data and send it to "hello.php". Under the document root, save the following script as "hello.html" −

```
<form action="hello.php" method="get">
   First Name: <input type="text" name="first_name"/>  <br/>
   Last Name: <input type="text" name="last_name" />
   <input type="submit" value="Submit" />
</form>
```

In your browser, enter the URL "http://localhost/hello.html" −


First Name: Amar
Last Name: Sharma

You should get the similar **output** in the browser window.

$_POST Array

The easiest way to send data to a server with the POST request is specifying the method attribute of HTML form as POST. Assuming that the URL in the browser is "http://localhost/hello.php", method=POST is set in a HTML form "hello.html" as in the earlier example −

```
<form action="hello.php" method="post">
   First Name: <input type="text" name="first_name"/>  <br/>
   Last Name: <input type="text" name="last_name" />
   <input type="submit" value="Submit" />
</form>
```

The "hello.php" script (in document root folder) retrieves the form data in the $_POST array and renders it as the HTTP response back to the browser –

```
<?php
   echo "First name: " . $_POST['first_name'] . " " .
      "Last Name: " . $_POST['last_name'] . "";
?>
```

Open **"http://localhost/hello.html"** in your browser. The data entered is retrieved by the server, and rendered back to the client, as in the earlier example.