HTML Form Action: POST and GET

The method attribute in the <form> element specifies how the data is sent to the server.

HTTP methods declare what action is to be performed on the data that is submitted to the server. HTTP Protocol provides several methods, and the HTML Form element is able to use two methods to send user data:

GET method - used to request data from a specified resource

POST method - used to send data to a server to update a resource

The GET Method

The HTML GET method is used to get a resource from the server. For example,

```
<form method="get" action="www.programiz.com/search">

    <input type="search" name="location" 
placeholder="Search.." />

    <input type="submit" value="Go" />

</form>
```

When we submit the above form by entering *California* in the input field, the request sent to the server will be www.programiz.com/search/?location=California.

The HTTP GET method adds a query string at the end of the URL to send data to the server. The query string is in the form of key-value pair followed by*?* symbol.

From the URL, the server can parse the user-submitted value where:

- key - location
- value - California

Note: If there is more than one query, the query string will be separated by a & symbol.

The POST method

The HTTP POST method is used to send data to the server for further processing. For example,

```
<form method="post" action="www.programiz.com/user">
    <label for="firstname">First name:</label>
    <input type="text" name="firstname" /><br />
    <label for="lastname">Last name:</label>
    <input type="text" name="lastname" /><br />
    <input type="submit" />
</form>
```

When we submit the form, it will add the user input data to the body of the request sent to the server. The request would look like

```
POST /user HTTP/2.0
Host: www.programiz.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 33

firstname=Robin&lastname=Williams
```

The data sent is not easily visible to the user. However, we can check the sent data using special tools like the browsers' dev tools.

GET vs POST

GET	POST
Data sent with the GET method is visible in the URL.	Data sent with the POST method is not visible.
GET requests can be bookmarked.	POST requests can't be bookmarked.
GET requests can be cached.	POST requests can't be cached.
GET requests have a character limit of **2048** characters.	POST requests do not have a limit.
Only ASCII characters are allowed in GET requests.	All data is allowed in POST request

PHP - $_REQUEST

$_REQUEST
$_REQUEST is a PHP super global variable which contains submitted form data, and all cookie data.

In other words, $_REQUEST is an array containing data from $_GET, $_POST, and $_COOKIE.

You can access this data with the $_REQUEST keyword followed by the name of the form field, or cookie, like this:

```
$_REQUEST['firstname']
```

Using $_REQUEST on $_POST Requests

POST request are usually data submitted from an HTML form.

Here is an example of how a HTML form could look like:

HTML form

```
<html>
<body>
<form method="post" action="demo_request.php">
  Name: <input type="text" name="fname">
  <input type="submit">
</form>
</body>
</html>
```

When a user clicks the submit button, the form data is sent to a PHP file specified in the action attribute of the <form> tag.

In the action file we can use the $_REQUEST variable to collect the value of the input field.

PHP file

```
$name = $_REQUEST['fname'];

echo $name;
```

In the example below we have put the HTML form and PHP code in the same PHP file.

We have also added some extra lines for security.

```
<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") {
  $name = htmlspecialchars($_REQUEST['fname']);
  if (empty($name)) {
    echo "Name is empty";
  } else {
    echo $name;
  }
}
?>

</body>
</html>
```

Using $_REQUEST on $_GET Requests
GET request can be form submissions as in the example above, with the method attribute of the HTML <form> element set to GET.

GET requests can also be data from a query string (information added after a URL address).

Here is an example of how an HTML hyperlink, with a query string could look like:

HTML link

```
<html>

<body>

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

</body>

</html>
```

When a user clicks the link, the query string data is sent to demo_phpfile.php.

In the PHP file we can use the $_REQUEST variable to collect the value of the query string.

Example

The PHP file demo_phpfile.php:

```
<html>
<body>

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

</body>
</html>
```

PHP $_POST

$_POST contains an array of variables received via the HTTP POST method.

There are two main ways to send variables via the HTTP Post method:

- HTML forms
- JavaScript HTTP requests

$_POST in HTML Forms

A HTML form submits information via the HTTP POST method if the form's method attribute is set to "POST".

To demonstrate this, we start by creating a simple HTML form:

HTML Form

```
<html>
<body>
<form method="POST" action="demo_request.php">
  Name: <input type="text" name="fname">
  <input type="submit">
</form>
</body>
</html>
```

When a user clicks the submit button, the form data is sent to a PHP file specified in the action attribute of the <form> tag.

In the action file we can use the $_POST variable to collect the value of the input field.

PHP $_GET

$_GET contains an array of variables received via the HTTP GET method.

There are two main ways to send variables via the HTTP GET method:

- Query strings in the URL
- HTML Forms

Query string in the URL

A query string is data added at the end of a URL. In the link below, everything after the ? sign is part of the query string:

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

The query string above contains two key/value pairs:

```
subject=PHP
web=W3schools.com
```

In the PHP file we can use the $_GET variable to collect the value of the query string.

Example

The PHP file demo_phpfile.php:

```
<html>

<body>

<?php

echo "Study " . $_GET['subject'] . " at " .
$_GET['web'];

?>


</body>

</html>
```

$_GET in HTML Forms
A HTML form submits information via the HTTP GET method if the form's method attribute is set to "GET".

To demonstrate this, we start by creating a simple HTML form:

HTML Form

```
<html>

<body>

<form action="welcome_get.php" method="GET">

  Name: <input type="text" name="name">

  E-mail: <input type="text" name="email">

  <input type="submit">

</form>

</body>

</html>
```

When a user clicks the submit button, the form data is sent to a PHP file specified in the action attribute of the <form> tag.

The form fields are sent to the PHP file, with your input, as query strings:

```
welcome_get.php?name=John&email=john@example.com
```

In the action file we can use the $_GET variable to collect the value of the input fields.

Example

PHP code inside the welcome_get.php page:

```
<html>
<body>

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

</body>
</html>
```

PHP 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>
```

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.

PHP Cookies

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).

PHP Mail

PHP mail() function is used to send email in PHP. You can send text message, html message and attachment with message using PHP mail() function.

PHP mail() function

Syntax

```
bool mail ( string $to , string $subject , string
$message [, string $additional_headers [, string
$additional_parameters ]] )
```

$to: specifies receiver or receivers of the mail. The receiver must be specified one of the following forms.

- user@example.com
- user@example.com, anotheruser@example.com
- User <user@example.com>
- User <user@example.com>, Another User anotheruser@example.com

$subject: represents subject of the mail.

$message: represents message of the mail to be sent.

Note: Each line of the message should be separated with a CRLF (\r\n) and lines should not be larger than 70 characters.

$additional_headers (optional): specifies the additional headers such as From, CC, BCC etc. Extra additional headers should also be separated with CRLF (\r\n).

PHP Mail Example

```
<?php
   ini_set("sendmail_from",
"sonoojaiswal@javatpoint.com");
   $to = "sonoojaiswal1987@gmail.com";//change
receiver address
   $subject = "This is subject";
   $message = "This is simple text message.";
   $header = "From:sonoojaiswal@javatpoint.com \r\n";

   $result = mail ($to,$subject,$message,$header);

   if( $result == true ){
      echo "Message sent successfully...";
   }else{
      echo "Sorry, unable to send mail...";
   }
?>
```

If you run this code on the live server, it will send an email to the specified receiver.

Connect PHP to MySQL

What is MySQL?

MySQL is an open-source relational database management system (RDBMS). It is the most popular database system used with PHP.

Structured Query Language (SQL). The data in a MySQL database are stored in tables that consist of columns and rows.

MySQL is a database system that runs on a server. MySQL is ideal for both small and large applications. MySQL is a very fast, reliable, and easy-to-use database system. It uses standard SQL. MySQL compiles on a number of platforms.

How we can connect PHP to MySQL?

PHP 5 and later can work with a MySQL database using:

- MySQLi extension (the 'i' is abbreviation for improved)
- PDO (PHP Data Objects)

Which one should we use MySQLi or PDO?

Both MySQLi and PDO have their recompenses:

- PDO will work with 12 different database systems, whereas MySQLi will only work with MySQL databases.
- So, if you have to shift your project to use alternative database, PDO makes the process easy. You only have to change the connection string and a few queries. With MySQLi, you will need to rewrite the complete code — queries included.

- Both are object-oriented, but MySQLi also offers a procedural API.

In short, you can choose whichever you want if you want to stick to MySQL otherwise you should go with PDO.

Connection to MySQL using MySQLi

PHP provides mysql_connect() function to open a database connection.

This function takes a single parameter, which is a connection returned by the mysql_connect() function.

You can disconnect from the MySQL database anytime using another PHP function mysql_close().

There is also a procedural approach of MySQLi to establish a connection to MySQL database from a PHP script.

It can be done in two ways:

MySQLi Object-Oriented

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";

// Connection
$conn = new mysqli($servername,
        $username, $password);

// For checking if connection is
// successful or not
if ($conn->connect_error) {
die("Connection failed: "
    . $conn->connect_error);
}
echo "Connected successfully";
?>
```