



**Institut
supérieur
d'informatique**

PHP Global Variables and Forms

Web1909A



Amirhossein Ghasemi

May 2020

255 Crémazie Est, bureau 100, Montréal (Québec) H2M 1M2
Téléphone : (514) 842-2426, télécopieur : (514) 842-2084
www.isi-mtl.com

Table of Contents

1	PHP Global Variables - Superglobals	3
1.1	PHP Superglobal - \$_GLOBALS	3
1.2	PHP Superglobal - \$_SERVER	5
1.3	PHP Superglobal - \$_REQUEST	8
1.4	PHP Superglobal - \$_POST	10
1.5	PHP Superglobal - \$_GET	11
2	PHP Form Handling	12
2.1	PHP - A Simple HTML Form	12
2.2	PHP Form Validation	16
2.3	PHP - Required Fields	26
2.4	PHP - Display The Error Messages	28
2.5	PHP Forms - Validate E-mail and URL	33
2.6	PHP Complete Form Example	39
3	PHP Registration Form using GET, POST Methods with Example.....	45
3.1	PHP POST method	48
3.2	PHP GET method	49
3.3	GET vs POST Methods	50
3.4	More examples	53
3.5	Working with check boxes and radio buttons	55

1 PHP Global Variables - Superglobals

Superglobals were introduced in PHP 4.1.0, and are built-in variables that are always available in all scopes.

Some predefined variables in PHP are "Superglobals", which means that they are always accessible, regardless of scope - and you can access them from any function, class or file without having to do anything special.

The PHP superglobal variables are:

- `$GLOBALS`
- `$_SERVER`
- `$_REQUEST`
- `$_POST`
- `$_GET`
- `$_FILES`
- `$_ENV`
- `$_COOKIE`
- `$_SESSION`

The next chapters will explain some of the superglobals, and the rest will be explained in later chapters.

1.1 PHP Superglobal - `$GLOBALS`

Super global variables are built-in variables that are always available in all scopes.

\$GLOBALS is a PHP super global variable which is used to access global variables from anywhere in the PHP script (also from within functions or methods).

PHP stores all global variables in an array called \$GLOBALS[index]. The index holds the name of the variable.

The example below shows how to use the super global variable \$GLOBALS:

Example (E1):

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6     $x = 75;
7     $y = 25;
8
9     function addition() {
10         $GLOBALS['z'] = $GLOBALS['x'] + $GLOBALS['y'];
11     }
12
13     addition();
14     echo $z;
15 ?>
16
17 </body>
18 </html>
```

OUTPUT

100

In the example above, since z is a variable present within the \$GLOBALS array, it is also accessible from outside the function!

1.2 PHP Superglobal - \$_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 (E2):

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6     echo $_SERVER['PHP_SELF'];
7     echo "<br>";
8     echo $_SERVER['SERVER_NAME'];
9     echo "<br>";
10    echo $_SERVER['HTTP_HOST'];
11    echo "<br>";
12    echo $_SERVER['SERVER_SOFTWARE'];
13    echo "<br>";
14    echo $_SERVER['SERVER_PROTOCOL'];
15    echo "<br>";
16    echo $_SERVER['SCRIPT_NAME'];
17 ?>
18
19 </body>
20 </html>
```

OUTPUT

```
/Day3/E2.php
localhost
localhost
Apache/2.4.41 (Win64) OpenSSL/1.1.1c PHP/7.4.3
HTTP/1.1
/Day3/E2.php
```

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

Element/Code	Description
<code>\$_SERVER['PHP_SELF']</code>	Returns the filename of the currently executing script
<code>\$_SERVER['GATEWAY_INTERFACE']</code>	Returns the version of the Common Gateway Interface (CGI) the server is using
<code>\$_SERVER['SERVER_ADDR']</code>	Returns the IP address of the host server
<code>\$_SERVER['SERVER_NAME']</code>	Returns the name of the host server (such as www.w3schools.com)
<code>\$_SERVER['SERVER_SOFTWARE']</code>	Returns the server identification string (such as Apache/2.2.24)
<code>\$_SERVER['SERVER_PROTOCOL']</code>	Returns the name and revision of the information protocol (such as HTTP/1.1)
<code>\$_SERVER['REQUEST_METHOD']</code>	Returns the request method used to access the page (such as POST)
<code>\$_SERVER['REQUEST_TIME']</code>	Returns the timestamp of the start of the request (such as 1377687496)

<code>\$_SERVER['QUERY_STRING']</code>	Returns the query string if the page is accessed via a query string
<code>\$_SERVER['HTTP_ACCEPT']</code>	Returns the Accept header from the current request
<code>\$_SERVER['HTTP_ACCEPT_CHARSET']</code>	Returns the Accept_Charset header from the current request (such as utf-8,ISO-8859-1)
<code>\$_SERVER['HTTP_HOST']</code>	Returns the Host header from the current request
<code>\$_SERVER['HTTP_REFERER']</code>	Returns the complete URL of the current page (not reliable because not all user-agents support it)
<code>\$_SERVER['HTTPS']</code>	Is the script queried through a secure HTTP protocol
<code>\$_SERVER['REMOTE_ADDR']</code>	Returns the IP address from where the user is viewing the current page
<code>\$_SERVER['REMOTE_HOST']</code>	Returns the Host name from where the user is viewing the current page

<code>\$_SERVER['REMOTE_PORT']</code>	Returns the port being used on the user's machine to communicate with the web server
<code>\$_SERVER['SCRIPT_FILENAME']</code>	Returns the absolute pathname of the currently executing script
<code>\$_SERVER['SERVER_ADMIN']</code>	Returns the value given to the <code>SERVER_ADMIN</code> 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 <code>someone@w3schools.com</code>)
<code>\$_SERVER['SERVER_PORT']</code>	Returns the port on the server machine being used by the web server for communication (such as 80)
<code>\$_SERVER['SERVER_SIGNATURE']</code>	Returns the server version and virtual host name which are added to server-generated pages
<code>\$_SERVER['PATH_TRANSLATED']</code>	Returns the file system based path to the current script
<code>\$_SERVER['SCRIPT_NAME']</code>	Returns the path of the current script
<code>\$_SERVER['SCRIPT_NAME']</code>	Returns the path of the current script
<code>\$_SERVER['SCRIPT_URI']</code>	Returns the URI of the current page

1.3 PHP Superglobal - `$_REQUEST`

PHP `$_REQUEST` is a PHP super global variable which 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 (E3):

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <form method="post" action="<?php echo $_SERVER['PHP_SELF'];?>"
6     Name: <input type="text" name="fname">
7     <input type="submit">
8 </form>
9
10 <?php
11 if ($_SERVER["REQUEST_METHOD"] == "POST") {
12     // collect value of input field
13     // $name = $_REQUEST['fname'];
14     $name = htmlspecialchars($_REQUEST['fname']);
15     if (empty($name)) {
16         echo "Name is empty";
17     } else {
18         echo $name;
19     }
20 }
21 ?>
22
23 </body>
24 </html>
```

OUTPUT

Name:

1.4 PHP Superglobal - \$_POST

PHP \$_POST is a PHP super global variable which is used to collect form data after submitting an HTML form with method="post". \$_POST is also widely used to pass variables.

Example (E4):

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <form method="post" action="<?php echo $_SERVER['PHP_SELF'];?>">
6     Name: <input type="text" name="fname">
7     <input type="submit">
8 </form>
9
10 <?php
11 if ($_SERVER["REQUEST_METHOD"] == "POST") {
12     // collect value of input field
13     $name = $_POST['fname'];
14     if (empty($name)) {
15         echo "Name is empty";
16     } else {
17         echo $name;
18     }
19 }
20 ?>
21
22 </body>
23 </html>
```

OUTPUT

Name:

1.5 PHP Superglobal - \$_GET

PHP \$_GET is a PHP super global variable which is 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= isi-international.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 (E5):



```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <a href="test_get.php?subject=PHP&web=isi-international.com ">Test $GET</a>
6
7 </body>
8 </html>
```

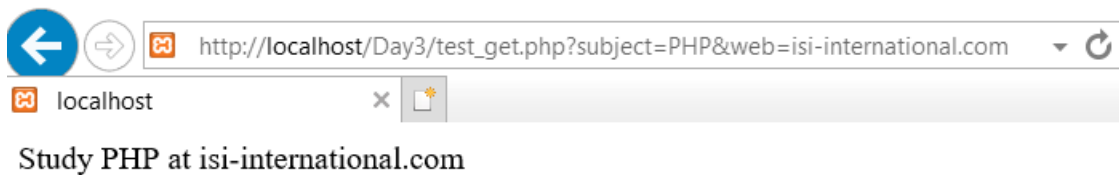
OUTPUT

[Test \\$GET](#)

test_get.php :

```
1 <html>
2 <body>
3
4 <?php
5     echo "Study " . $_GET['subject'] . " at " . $_GET['web'];
6
7     ?>
8
9 </body>
</html>
```

After clicking on Test\$GET:



2 PHP Form Handling

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

2.1 PHP - A Simple HTML Form

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

Example (E6):

```

1 <!DOCTYPE HTML>
2 <html>
3 <body>
4
5 <form action="welcome.php" method="post">
6   Name: <input type="text" name="name"><br>
7   E-mail: <input type="text" name="email"><br>
8   <input type="submit">
9 </form>
10
11 </body>
12 </html>

```

OUTPUT

Name:

E-mail:

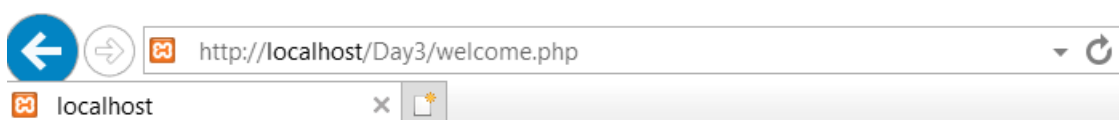
When the user fills out the form above and clicks the submit button, the form data is sent for processing to a PHP file named "welcome.php". The form data is sent with the HTTP POST method. To display the submitted data you could simply echo all the variables. The "welcome.php" looks like this:

```

1 <html>
2 <body>
3
4 Welcome <?php echo $_POST["name"]; ?><br>
5 Your email address is: <?php echo $_POST["email"]; ?>
6
7 </body>
8 </html>

```

output:



Welcome Bill Gates
Your email address is: bill.gates@example.com

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

Example (E7):

```
1 <!DOCTYPE HTML>
2 <html>
3 <body>
4
5 <form action="welcome_get.php" method="get">
6   Name: <input type="text" name="name"><br>
7   E-mail: <input type="text" name="email"><br>
8   <input type="submit">
9 </form>
10
11 </body>
12 </html>
```

OUTPUT

Name:

E-mail:

and "welcome_get.php" looks like this:

```
1 <html>
2 <body>
3
4 Welcome <?php echo $_GET["name"]; ?><br>
5 Your email address is: <?php echo $_GET["email"]; ?>
6
7 </body>
8 </html>
```

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

Think SECURITY when processing PHP forms!

This part does not contain any form validation, it just shows how you can send and retrieve form data.

However, the next parts will show how to process PHP forms with security in mind! Proper validation of form data is important to protect your form from hackers and spammers!

GET vs. POST

Both GET and POST create an array (e.g. `array(key1 => value1, key2 => value2, key3 => value3, ...)`). This array holds key/value pairs, where keys are the names of the form controls and values are the input data from the user.

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

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

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

When to use GET?

Information sent from a form with the GET method is visible to everyone (all variable names and values are displayed in the URL). GET also has limits on the amount of information to send. The limitation is about 2000 characters. However, because the

variables are displayed in the URL, it is possible to bookmark the page. This can be useful in some cases.

GET may be used for sending non-sensitive data.

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

When to use POST?

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

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

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

Developers prefer POST for sending form data.

Next, let's see how we can process PHP forms the secure way!

2.2 PHP Form Validation

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

Think SECURITY when processing PHP forms!

These parts will show how to process PHP forms with security in mind. Proper validation of form data is important to protect your form from hackers and spammers!

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

PHP Form Validation Example

* required field

Name: *

E-mail: *

Website:

Comment:

Gender: ☐ Female ☐ Male ☐ Other *

Your Input:

The validation rules for the form above are as follows:

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

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

Text Fields

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

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

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

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

Website: <input type="text" name="website">

Comment: <textarea name="comment" rows="5" cols="40"></textarea>

Radio Buttons

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

Gender:

```
<input type="radio" name="gender" value="female">Female  
<input type="radio" name="gender" value="male">Male  
<input type="radio" name="gender" value="other">Other
```

Gender:

<input type="radio" name="gender" value="female">Female

<input type="radio" name="gender" value="male">Male

<input type="radio" name="gender" value="other">Other

The Form Element

The HTML code of the form looks like this:

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

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

What is the \$_SERVER["PHP_SELF"] variable?

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

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

What is the htmlspecialchars() function?

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

Big Note on PHP Form Security

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

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

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

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

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

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

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

So far, so good.

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

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

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

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

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

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

How to Avoid `$_SERVER["PHP_SELF"]` Exploits?

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

The form code should look like this:

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

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

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

The exploit attempt fails, and no harm is done!

Validate Form Data With PHP

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

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

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

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

```
&lt;script&gt;location.href('http://www.hacked.com')&lt;/script
&gt;
```

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

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

- Strip unnecessary characters (extra space, tab, newline) from the user input data (with the PHP trim() function)

- Remove backslashes (\) from the user input data (with the PHP stripslashes() function)

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

We will name the function test_input().

Now, we can check each \$_POST variable with the test_input() function, and the script looks like this (E8):

```
<!DOCTYPE HTML>

<html>

<head>

</head>

<body>


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

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

```
}
```

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

<h2>PHP Form Validation Example</h2>

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

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

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

Website: <input type="text" name="website">

Comment: <textarea name="comment" rows="5" cols="40"></textarea>

Gender:

<input type="radio" name="gender" value="female">Female

<input type="radio" name="gender" value="male">Male

<input type="radio" name="gender" value="other">Other

<input type="submit" name="submit" value="Submit">

</form>


```
<?php
echo "<h2>Your Input:</h2>";
echo $name;
echo "<br>";
echo $email;
echo "<br>";
echo $website;
echo "<br>";
echo $comment;
echo "<br>";
echo $gender;
?>

</body>
</html>
```

OUTPUT

PHP Form Validation Example

Name:

E-mail:

Website:

Comment:

Gender: ☐ Female ☐ Male ☐ Other

Your Input:

A. G h a s e m i

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

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

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

2.3 PHP - Required Fields

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

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

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

Gender Required. Must select one

In the previous chapter, all input fields were optional.

In the following code we have added some new variables: \$nameErr, \$emailErr, \$genderErr, and \$websiteErr. These error variables will hold error messages for the required fields. We have also added an if else statement for each \$_POST variable. This checks if the \$_POST variable is empty (with the PHP empty() function). If it is empty, an error message is stored in the different error variables, and if it is not empty, it sends the user input data through the test_input() function (E9):

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

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

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

```
}

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

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

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

2.4 PHP - Display The Error Messages

Then in the HTML form, we add a little script after each required field, which generates the correct error message if needed (that

is if the user tries to submit the form without filling out the required fields):

Example (10):

```
<!DOCTYPE HTML>

<html>

<head>

<style>

.error {color: #FF0000;}

</style>

</head>

<body>


<?php

// define variables and set to empty values

$nameErr = $emailErr = $genderErr = $websiteErr = "";

$name = $email = $gender = $comment = $website = "";


if ($_SERVER["REQUEST_METHOD"] == "POST") {

    if (empty($_POST["name"])) {

        $nameErr = "Name is required";

    } else {

        $name = test_input($_POST["name"]);

    }


    if (empty($_POST["email"])) {

        $emailErr = "Email is required";
```

```

} else {
    $email = test_input($_POST["email"]);
}

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

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

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

function test_input($data) {
    $data = trim($data);
    $data = stripslashes($data);
    $data = htmlspecialchars($data);
}

```

```
    return $data;
}
?>
```

<h2>PHP Form Validation Example</h2>

<p>* required field</p>

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

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

* <?php echo \$nameErr;?>

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

* <?php echo \$emailErr;?>

Website: <input type="text" name="website">

<?php echo \$websiteErr;?>

Comment: <textarea name="comment" rows="5" cols="40"></textarea>

Gender:

<input type="radio" name="gender" value="female">Female

<input type="radio" name="gender" value="male">Male

<input type="radio" name="gender" value="other">Other

* <?php echo \$genderErr;?>

<input type="submit" name="submit" value="Submit">

</form>

<?php

```
echo "<h2>Your Input:</h2>";  
echo $name;  
echo "<br>";  
echo $email;  
echo "<br>";  
echo $website;  
echo "<br>";  
echo $comment;  
echo "<br>";  
echo $gender;  
?>
```

```
</body>
```

```
</html>
```

OUTPUT

PHP Form Validation Example

* required field

Name: *

E-mail: *

Website:

Comment:

Gender: ☐ Female ☐ Male ☐ Other *

Your Input:

A. G h a s e m i

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

2.5 PHP Forms - Validate E-mail and URL

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

PHP - Validate Name

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

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

The `preg_match()` function searches a string for pattern, returning true if the pattern exists, and false otherwise.

PHP - Validate E-mail

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

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

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

PHP - Validate URL

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

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

PHP - Validate Name, E-mail, and URL

Now, the script looks like this:

Example (E11):

```
<!DOCTYPE HTML>
<html>
<head>
<style>
.error {color: #FF0000;}
</style>
</head>
```

<body>

<?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
    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"]);
}
}

function test_input($data) {
    $data = trim($data);

```

```
$data = stripslashes($data);  
$data = htmlspecialchars($data);  
return $data;  
}  
?>
```

<h2>PHP Form Validation Example</h2>

<p>* required field</p>

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

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

* <?php echo \$nameErr;?>

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

* <?php echo \$emailErr;?>

Website: <input type="text" name="website">

<?php echo \$websiteErr;?>

Comment: <textarea name="comment" rows="5" cols="40"></textarea>

Gender:

<input type="radio" name="gender" value="female">Female

<input type="radio" name="gender" value="male">Male

<input type="radio" name="gender" value="other">Other

* <?php echo \$genderErr;?>

<input type="submit" name="submit" value="Submit">

```

</form>

<?php
echo "<h2>Your Input:</h2>";

echo $name;

echo "<br>";

echo $email;

echo "<br>";

echo $website;

echo "<br>";

echo $comment;

echo "<br>";

echo $gender;

?>

</body>

</html>

```

OUTPUT

PHP Form Validation Example

* required field

Name: *

E-mail: *

Website:

Comment:

Gender: ☐ Female ☐ Male ☐ Other *

Your Input:

A. G h a s e m i

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

2.6 PHP Complete Form Example

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

PHP - Keep The Values in The Form

To show the values in the input fields after the user hits the submit button, we add a little PHP script inside the value attribute of the following input fields: name, email, and website. In the comment textarea field, we put the script between the `<textarea>` and `</textarea>` tags. The little script outputs the value of the `$name`, `$email`, `$website`, and `$comment` variables. Then, we also need to show which radio button that was checked. For this, we must manipulate the checked attribute (not the value attribute for radio buttons):

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

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

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

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

Gender:

```
<input type="radio" name="gender"
```

```
<?php if (isset($gender) && $gender=="female") echo  
"checked";?>
```

```
value="female">Female
```

```
<input type="radio" name="gender"
```

```
<?php if (isset($gender) && $gender=="male") echo  
"checked";?>
```

```
value="male">Male
```

```
<input type="radio" name="gender"
```

```
<?php if (isset($gender) && $gender=="other") echo  
"checked";?>
```

```
value="other">Other
```

PHP - Complete Form Example

Here is the complete code for the PHP Form Validation Example (E12):

```
<!DOCTYPE HTML>
```

```
<html>
```

```
<head>
```

```
<style>
```

```
.error {color: #FF0000;}
```

```
</style>
```

```
</head>
```


<body>

<?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"]);
}

function test_input($data) {
    $data = trim($data);

```

```

    $data = stripslashes($data);
    $data = htmlspecialchars($data);
    return $data;
}
?>

```

<h2>PHP Form Validation Example</h2>

<p>* required field</p>

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

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

* <?php echo \$nameErr;?>

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

* <?php echo \$emailErr;?>

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

<?php echo \$websiteErr;?>

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

Gender:

<input type="radio" name="gender" <?php if (isset(\$gender) && \$gender=="female") echo "checked";?> value="female">Female

<input type="radio" name="gender" <?php if (isset(\$gender) && \$gender=="male") echo "checked";?> value="male">Male

<input type="radio" name="gender" <?php if (isset(\$gender) && \$gender=="other") echo "checked";?> value="other">Other

```
<span class="error">* <?php echo $genderErr;?></span>
<br><br>
<input type="submit" name="submit" value="Submit">
</form>

<?php
echo "<h2>Your Input:</h2>";
echo $name;
echo "<br>";
echo $email;
echo "<br>";
echo $website;
echo "<br>";
echo $comment;
echo "<br>";
echo $gender;
?>

</body>
</html>
```

OUTPUT

PHP Form Validation Example

* required field

Name: *

E-mail: *

Website:

Comment:

Gender: ☐ Female ☐ Male ☐ Other *

Your Input:

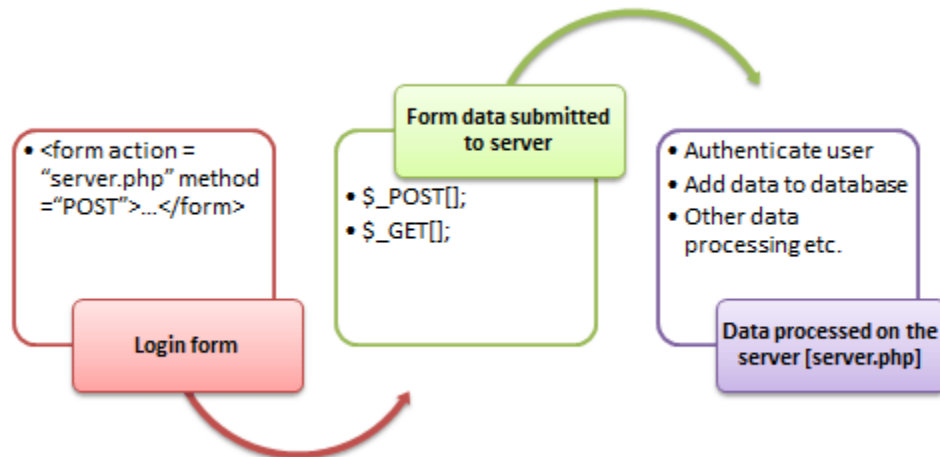
3 PHP Registration Form using GET, POST Methods with Example

What is Form?

When you login into a website or into your mailbox, you are interacting with a form.

Forms are used to get input from the user and submit it to the web server for processing.

The diagram below illustrates the form handling process.



A form is an HTML tag that contains graphical user interface items such as input box, check boxes radio buttons etc.

The form is defined using the `<form>...</form>` tags and GUI items are defined using form elements such as input.

When and why we are using forms?

Forms come in handy when developing flexible and dynamic applications that accept user input.

Forms can be used to edit already existing data from the database

Create a form

We will use HTML tags to create a form. Below is the minimal list of things you need to create a form.

- Opening and closing form tags `<form>...</form>`
- Form submission type POST or GET
- Submission URL that will process the submitted data

- Input fields such as input boxes, text areas, buttons, checkboxes etc.

The code below creates a simple registration form (E13):

```

1 <html>
2 <head>
3     <title>Registration Form</title>
4     <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
5 </head>
6 <body>
7
8     <h2>Registration Form</h2>
9
10    <form action="registration_form.php" method="POST"> First name:
11
12        <input type="text" name="firstname"> <br> Last name:
13
14        <input type="text" name="lastname">
15
16        <input type="hidden" name="form_submitted" value="1" />
17
18        <input type="submit" value="Submit">
19
20    </form>
21 </body>
22 </html>

```

Viewing the above code in a web browser displays the following form:

Registration Form

First name:

Last name:

Submit

Button

The diagram shows the visual output of the HTML code. It features a title 'Registration Form' at the top. Below it, there are two text input fields. The first field is preceded by the label 'First name:' and the second by 'Last name:'. Red arrows point from the word 'Labels' to these two labels. Another red arrow points from the word 'Input boxes' to the two text input fields. Below the input fields is a 'Submit' button. A red arrow points from the word 'Button' to this button.

HERE,

- `<form...>...</form>` are the opening and closing form tags
- `action="registration_form.php" method="POST">` specifies the destination URL and the submission type.
- First/Last name: are labels for the input boxes
- `<input type="text" ...>` are input box tags
- `
` is the new line tag
- `<input type="hidden" name="form_submitted" value="1"/>` is a hidden value that is used to check whether the form has been submitted or not
- `<input type="submit" value="Submit">` is the button that when clicked submits the form to the server for processing

Submitting the form data to the server

The action attribute of the form specifies the submission URL that processes the data. The method attribute specifies the submission type.

3.1 PHP POST method

- This is the built in PHP super global array variable that is used to get values submitted via HTTP POST method.
- The array variable can be accessed from any script in the program; it has a global scope.
- This method is ideal when you do not want to display the form post values in the URL.
- A good example of using post method is when submitting login details to the server.

It has the following syntax.

```
<?php  
$_POST['variable_name'];  
?>
```

HERE,

- “\$_POST[...]” is the PHP array
- “variable_name” is the URL variable name.

3.2 PHP GET method

- This is the built in PHP super global array variable that is used to get values submitted via HTTP GET method.
- The array variable can be accessed from any script in the program; it has a global scope.
- This method displays the form values in the URL.
- It is ideal for search engine forms as it allows the users to book mark the results.

It has the following syntax.

```
<?php  
$_GET['variable_name'];  
?>
```

HERE,

- “\$_GET[...]” is the PHP array
- “variable_name” is the URL variable name.

3.3 GET vs POST Methods

POST	GET
Values not visible in the URL	Values visible in the URL
Has not limitation of the length of the values since they are submitted via the body of HTTP	Has limitation on the length of the values usually 255 characters. This is because the values are displayed in the URL. Note the upper limit of the characters is dependent on the browser.
Has lower performance compared to Php_GET method due to time spent encapsulation the Php_POST values in the HTTP body	Has high performance compared to POST method dues to the simple nature of appending the values in the URL.
Supports many different data types such as string, numeric, binary etc.	Supports only string data types because the values are displayed in the URL
Results cannot be book marked	Results can be book marked due to the visibility of the values in the URL

The below diagram shows the difference between get and post:

FORM SUBMISSION POST METHOD

```
<form action="registration_form.php" method="POST">
  First name: <input type="text" name="firstname"><br>
  Last name: <input type="text" name="lastname">
  <br>
  <input type="hidden" name="form_submitted" value="1"/>
  <input type="submit" value="Submit">
</form>
```

Submission URL does not show form values

localhost/tuttis/registration_form.php

FORM SUBMISSION GET METHOD

```
<form action="registration_form.php" method="GET">
  First name: <input type="text" name="firstname"><br>
  Last name: <input type="text" name="lastname">
  <br>
  <input type="hidden" name="form_submitted" value="1"/>
  <input type="submit" value="Submit">
</form>
```

SUBMISSION URL SHOWS FORM VALUES

localhost/tuttis/registration_form.php?firstname=Smith&lastname=Jones&form_submitted=1

Processing the registration form data

The registration form submits data to itself as specified in the action attribute of the form.

When a form has been submitted, the values are populated in the `$_POST` super global array.

We will use the PHP `isset` function to check if the form values have been filled in the `$_POST` array and process the data.

We will modify the registration form to include the PHP code that processes the data. Below is the modified code (E14).

```

1 <html>
2 <head>
3     <title>Registration Form</title>
4     <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
5
6 </head>
7 <body>
8
9     <?php if (isset($_POST['form_submitted'])): ?> //this code is executed when the form is submitted
10
11         <h2>Thank You <?php echo $_POST['firstname']; ?> </h2>
12
13         <p>You have been registered as
14             <?php echo $_POST['firstname'] . ' ' . $_POST['lastname']; ?>
15         </p>
16
17         <p>Go <a href="/registration_form.php">back</a> to the form</p>
18
19     <?php else: ?>
20
21         <h2>Registration Form</h2>
22
23         <form action="registration_form.php" method="POST">
24
25             First name:
26             <input type="text" name="firstname">
27
28             <br> Last name:
29             <input type="text" name="lastname">
30
31             <input type="hidden" name="form_submitted" value="1" />
32
33             <input type="submit" value="Submit">
34
35         </form>
36
37     <?php endif;
38     ?>
39 </body>
40 </html>

```

Registration Form

First name:

Last name:

HERE,

- <?php if (isset(\$_POST['form_submitted'])): ?> checks if the form_submitted hidden field has been filled in the \$_POST[] array and display a thank you and first name message.

If the `form_fobmitted` field hasn't been filled in the `$_POST[]` array, the form is displayed.

3.4 More examples

Simple search engine

We will design a simple search engine that uses the `PHP_GET` method as the form submission type.

For simplicity's sake, we will use a PHP If statement to determine the output.

We will use the same HTML code for the registration form above and make minimal modifications to it (E15).

```

1 <html>
2 <head>
3     <title>Simple Search Engine</title>
4     <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
5 </head>
6 <body>
7
8     <?php if (isset($_GET['form_submitted'])): ?>
9
10        <h2>Search Results For <?php echo $_GET['search_term']; ?> </h2>
11        <?php if ($_GET['search_term'] == "GET"): ?>
12
13            <p>The GET method displays its values in the URL</p>
14
15        <?php else: ?>
16            <p>Sorry, no matches found for your search term</p>
17
18        <?php endif; ?>
19
20        <p>Go <a href="/search_engine.php">back</a> to the form</p>
21
22        <?php else: ?>
23
24            <h2>Simple Search Engine - Type in GET </h2>
25
26            <form action="search_engine.php" method="GET">
27
28                Search Term:
29                <input type="text" name="search_term">
30                <br>
31
32                <input type="hidden" name="form_submitted" value="1" />
33
34                <input type="submit" value="Submit">
35
36            </form>
37        <?php endif; ?>
38 </body>
39 </html>

```

View the above page in a web browser the following form will be shown:

Simple Search Engine - Type in GET

Search Term:

Type GET in upper case letter then click on submit button.

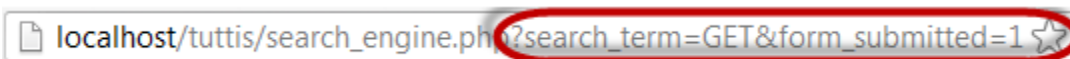
The following will be shown

Search Results For GET

The GET method displays its values in the URL

Go [back](#) to the form

The diagram below shows the URL for the above results



localhost/tuttis/search_engine.php?search_term=GET&form_submitted=1

Note the URL has displayed the value of search_term and form_submitted. Try to enter anything different from GET then click on submit button and see what results you will get.

3.5 Working with check boxes and radio buttons

If the user does not select a check box or radio button, no value is submitted, if the user selects a check box or radio button, the value one (1) or true is submitted.

We will modify the registration form code and include a check button that allows the user to agree to the terms of service (E16).

```

1 <html>
2 <head>
3     <title>Registration Form</title>
4     <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
5 </head>
6 <body>
7
8     <?php if (isset($_POST['form_submitted'])): ?>
9
10        <?php if (!isset($_POST['agree'])): ?>
11
12            <p>You have not accepted our terms of service</p>
13
14        <?php else: ?>
15
16            <h2>Thank You <?php echo $_POST['firstname']; ?></h2>
17
18            <p>You have been registered as
19                <?php echo $_POST['firstname'] . ' ' . $_POST['lastname']; ?>
20            </p>
21
22            <p> Go <a href="/registration_form2.php">back</a> to the form</p>
23
24        <?php endif; ?>
25
26        <?php else: ?>
27
28            <h2>Registration Form</h2>
29
30            <form action="registration_form2.php" method="POST">
31
32                First name:
33                <input type="text" name="firstname">
34
35                <br> Last name:
36                <input type="text" name="lastname">
37
38                <br> Agree to Terms of Service:
39                <input type="checkbox" name="agree">
40                <br>
41
42                <input type="hidden" name="form_submitted" value="1" .
43
44                <input type="submit" value="Submit">
45
46            </form>
47        <?php endif; ?>
48 </body>
49 </html>

```

View the above form in a browser:

Registration Form

First name:

Last name:

Agree to Terms of Service: ☐

Fill in the first and last names

Note the Agree to Terms of Service checkbox has not been selected.

Click on submit button

You will get the following results

You have not accepted our terms of service

Go [back](#) to the form

Click on back to the form link and then select the checkbox

Registration Form

First name:

Last name:

Agree to Terms of Service: ☒

Click on submit button

You will get the following results

Thank You Smith

You have been registered as Smith Jones

Go [back](#) to the form

Summary

- Forms are used to get data from the users
- Forms are created using HTML tags
- Forms can be submitted to the server for processing using either POST or GET method
- Form values submitted via the POST method are encapsulated in the HTTP body.
- Form values submitted via the GET method are appended and displayed in the URL.