COMP30680
Web Application Development

PHP – Forms and filters.

David Coyle
d.coyle@ucd.ie

PHP 5 Form Handling

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

Both GET and POST create an array, e.g. array(key => value, 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.

$_GET and $_POST.

They are superglobals. 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.

PHP 5 Form Handling

The example below uses POST. It displays a simple HTML form with two input fields and a submit button.

```
<html>
<body>

<form action="welcome.php" method="post">
Name: <input type="text" name="name"><br>
E-mail: <input type="text" name="email"><br>
<input type="submit">
</form>

</body>
</html>
```

```
<html>
<body>

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

</body>
</html>
```

```
Welcome John
Your email address is john.doe@example.com
```

The same thing with GET

```
<html>
<body>

<form action="welcome_get.php" method="get">
Name: <input type="text" name="name"><br>
E-mail: <input type="text" name="email"><br>
<input type="submit">
</form>

</body>
</html>
```

```
<html>
<body>

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

</body>
</html>
```

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 a 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.

$_SERVER["PHP_SELF"]

In the previous examples information was submitted (using GET or POST) to another php file. A php file can also send information to itself.

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.

One useful application is to display error messages on the same page as the form.

Scripting attacks

Look at the file **xss_unsafe.php**.

It posts to itself using $_SERVER["PHP_SELF"].

```
13
14 ▼ <form method="post" action="<?php echo $_SERVER["PHP_SELF"];?>">
15       Name: <input type="text" name="name">
16       <input type="submit" value="Submit to this page">
17   </form>
18
19
20   <?php
21       echo "<h2>Your Input:</h2>";
22       echo "Name = " . $name;
23   ?>
24
```

This can cause an alert box to appear with the word hacked.***

This is because the text gets passed to the webpage and runs as JavaScript.

*** But it it probably won't, because browsers are now pretty good at catching simple scripting attacks.

Try submitting the following text in the form: **<script>alert('hacked')</script>**

Avoiding $_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, the exploit attempt fails, and no harm is done!

Look at the file **xss_safe.php**. This time htmlspecialchars() is used.

PHP Form Validation Example

* required field.

Name: *

E-mail: *

Website:

Comment:

Gender: Female Male *

Submit

http://www.w3schools.com/php/php_form_validation.asp

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

A basic, version.

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

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

?>

<h2>PHP Form Validation Example</h2>
<form method="post" action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>">
   Name: <input type="text" name="name">
   <br><br>
   E-mail: <input type="text" name="email">
   <br><br>
   Website: <input type="text" name="website">
   <br><br>
   Comment: <textarea name="comment" rows="5" cols="40"></textarea>
   <br><br>
   Gender:
   <input type="radio" name="gender" value="female">Female
   <input type="radio" name="gender" value="male">Male
   <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;
?>
```

See **form_start.php**

A basic, version.

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

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

?>

<h2>PHP Form Validation Example</h2>
<form method="post" action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>">
   Name: <input type="text" name="name">
   <br><br>
   E-mail: <input type="text" name="email">
   <br><br>
```

At the start of the script, check whether the form has been submitted using $_SERVER["REQUEST_METHOD"].

If the REQUEST_METHOD is POST, then the form has been submitted information should be validated and displayed.

If it has not been submitted, skip and display a blank form.

```php
echo "<br>";
echo $gender;
?>
```

See **form_start.php**

Further basic testing.

You can also do two more things when the user submits the form:

Strip unnecessary characters (extra space, tab, newline) from the user input data.

- Do this with the PHP trim() function.

Remove backslashes (\) from the user input data .

- Do this with the PHP stripslashes() function.

It’s useful to have a single function that takes care of all of this.

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

See **form_basic_testing.php**

Required fields

In the previous version all input fields were 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.

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

Required fields

See **form_validation_1.php**

New variables have been added: $nameErr, $emailErr, $genderErr, and $websiteErr.
These error variables will hold error messages for the required fields.

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
- If it is not empty, it sends the user input data through the test_input() function:

```
// 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"]);
  }
```

Required fields

See **form_validation_1.php**

A script is also included after each required field, which generates the correct error message if the user tries to submit the form without filling out the required fields:

```
Name: <input type="text" name="name">
<span class="error">* <?php echo $nameErr;?></span>
<br><br>
E-mail:
<input type="text" name="email">
<span class="error">* <?php echo $emailErr;?></span>
<br><br>
Website:
<input type="text" name="website">
<span class="error"><?php echo $websiteErr;?></span>
<br><br>
```

Validate the Name field

See **form_validation_2.php**

Check for letter and whitespace only.

```
$name = test_input($_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.

Validate the Email field

See **form_validation_2.php**

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";
}
```

Validate the URL

See **form_validation_2.php**

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";
}
```

Keep The Values in The Form

See **form_final.php**

```
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
```

PHP Filters

Validating data = Determine if the data is in proper form.

Sanitizing data = Remove any illegal character from the data.

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.

Try running the file **list_filters.php**

Filter Name	Filter ID
int	257
boolean	258
float	259
validate_regexp	272
validate_url	273
validate_email	274
validate_ip	275
validate_mac	276
string	513
stripped	513
encoded	514
special_chars	515
full_special_chars	522
unsafe_raw	516
email	517
url	518
number_int	519
number_float	520
magic_quotes	521
callback	1024

PHP filter_var() Function

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

Invalid submitted data can lead to security problems and break your web app.

By using PHP filters you can help to ensure your application gets the correct input.

Filter examples

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

Uses the filter_var() function to remove all HTML tags from a string

```php
<?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");
}
?>
```

Uses the filter_var() function to check if the variable $int is an integer.

Check an IP address

```php
<?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");
}
?>
```

Uses the filter_var() function to check if the variable $ip is a valid IP address

Check a URL

```php
<?php
$url = "http://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");
}
?>
```

Uses the filter_var() function to

- first remove all illegal characters from a URL
- then check if $url is a valid URL

Check an email address

```php
<?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");
}
?>
```

Uses the filter_var() function to:

- first remove all illegal characters from the $email variable
- then check if it is a valid email address

For even more examples of Filters see: http://www.w3schools.com/php/php_filter_advanced.asp

Questions, Suggestions?

Materials and further reading:

w3Schools PHP forms: http://www.w3schools.com/php/php_forms.asp

Next:

Cookies and sessions