PHP Form Validation

In this tutorial you'll learn how to sanitize and validate form data using PHP filters.

Sanitizing and Validating Form Data

As you have seen in the previous tutorial, the process of capturing and displaying the submitted form data is quite simple. In this tutorial you will learn how to implement a simple contact form on your website that allows the user to send their comment and feedback through email. We will use the same PHP `mail()` function to send the emails.

We are also going to implement some basic security feature like sanitization and validation of the user's input so that user can not insert potentially harmful data that compromise the website security or might break the application.

The following is our all-in-one PHP script which does the following things:

- It will ask the users to enter his comments about the website.
- The same script displays the contact form and process the submitted form data.
- The script sanitizes and validates the user inputs. If any required field (marked with `*`) is missing or validation failed due to incorrect inputs the script redisplays the form with an error message for corresponding form field.
- The script remembers which fields the user has already filled in, and prefills those fields when the form redisplayed due to validation error.
- If the data submitted by the user are acceptable and everything goes well it will send an email to the website administrator and display a success message to the user.

Type the following code in "contact.php" file and save in your project root directory:

Example

Download

```
<?php
// Functions to filter user inputs
function filterName($field){
    // Sanitize user name
    $field = filter_var(trim($field), FILTER_SANITIZE_STRING);

    // Validate user name
    if(filter_var($field, FILTER_VALIDATE_REGEXP,
array("options"=>array("regexp"=>"/^[a-zA-Z\s]+$/")))){
        return $field;
    } else{
        return FALSE;
    }
}
function filterEmail($field){
    // Sanitize e-mail address
    $field = filter_var(trim($field), FILTER_SANITIZE_EMAIL);

    // Validate e-mail address
    if(filter_var($field, FILTER_VALIDATE_EMAIL)){
        return $field;
    } else{
        return FALSE;
    }
}
function filterString($field){
    // Sanitize string
    $field = filter_var(trim($field), FILTER_SANITIZE_STRING);
    if(!empty($field)){
        return $field;
    } else{
        return FALSE;
    }
}

// Define variables and initialize with empty values
$nameErr = $emailErr = $messageErr = "";
$name = $email = $subject = $message = "";

// Processing form data when form is submitted
if($_SERVER["REQUEST_METHOD"] == "POST"){

    // Validate user name
    if(empty($_POST["name"])){
        $nameErr = "Please enter your name.";
    } else{
        $name = filterName($_POST["name"]);
        if($name == FALSE){
            $nameErr = "Please enter a valid name.";
        }
    }

    // Validate email address
    if(empty($_POST["email"])){
        $emailErr = "Please enter your email address.";
    } else{
        $email = filterEmail($_POST["email"]);
        if($email == FALSE){
            $emailErr = "Please enter a valid email address.";
        }
    }

    // Validate message subject
    if(empty($_POST["subject"])){
        $subject = "";
    } else{
        $subject = filterString($_POST["subject"]);
    }

    // Validate user comment
    if(empty($_POST["message"])){
        $messageErr = "Please enter your comment.";
    } else{
        $message = filterString($_POST["message"]);
        if($message == FALSE){
            $messageErr = "Please enter a valid comment.";
        }
    }

    // Check input errors before sending email
    if(empty($nameErr) && empty($emailErr) && empty($messageErr)){
        // Recipient email address
        $to = 'webmaster@example.com';

        // Create email headers
        $headers = 'From: '. $email . "\r\n" .
        'Reply-To: '. $email . "\r\n" .
        'X-Mailer: PHP/' . phpversion();

        // Sending email
        if(mail($to, $subject, $message, $headers)){
            echo '<p class="success">Your message has been sent 
successfully!</p>';
        } else{
            echo '<p class="error">Unable to send email. Please try again!</p>';
        }
    }
}
?>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Contact Form</title>
    <style type="text/css">
        .error{ color: red; }
        .success{ color: green; }
    </style>
</head>
<body>
    <h2>Contact Us</h2>
    <p>Please fill in this form and send us.</p>
    <form action="contact.php" method="post">
        <p>
            <label for="inputName">Name:<sup>*</sup></label>
            <input type="text" name="name" id="inputName" value="<?php echo 
$name; ?>">
            <span class="error"><?php echo $nameErr; ?></span>
        </p>
        <p>
            <label for="inputEmail">Email:<sup>*</sup></label>
            <input type="text" name="email" id="inputEmail" value="<?php echo 
$email; ?>">
            <span class="error"><?php echo $emailErr; ?></span>
        </p>
        <p>
            <label for="inputSubject">Subject:</label>
            <input type="text" name="subject" id="inputSubject" value="<?php 
echo $subject; ?>">
        </p>
        <p>
            <label for="inputComment">Message:<sup>*</sup></label>
            <textarea name="message" id="inputComment" rows="5" cols="30"><?php 
echo $message; ?></textarea>
            <span class="error"><?php echo $messageErr; ?></span>
        </p>
        <input type="submit" value="Send">
        <input type="reset" value="Reset">
    </form>
</body>
</html>
```

Explanation of code

You might think what that code was all about. OK, let's get straight into it.

- The `filterName()` function (*line no-03*) validate input value as person's name. A valid name can only contain alphabetical characters (a-z, A-Z).
- The `filterEmail()` function (*line no-14*) validate input value as email address.
- The `filterString()` function (*line no-25*) only sanitize the input value by stripping HTML tags and special characters. It doesn't validate the input value against anything.
- The attribute `action="contact.php"` (*line no-111*) inside the `<form>` tag specifies that the same `contact.php` file display the form as well as process the form data.
- The PHP code inside the value attribute of `<input>` and `<textarea>` e.g. `<?php echo $name; ?>` display prefilled value when form is redisplayed upon validation error.
- The PHP code inside the `.error` class e.g. `<span class="error"><?php echo $nameErr; ?></span>` display error for corresponding field.

Rest the thing we have already covered in previous chapters. To learn more about sanitize and validate filters, please check out the PHP Filter reference.

Note: You need to setup a mail server on your machine for the PHP `mail()` function to work. If you just want to implement the form validation you can replace the mail part (line no. `81` to `94`) with your own custom code.

https://www.tutorialrepublic.com/php-tutorial/php-form-validation.php

PHP Form Validation

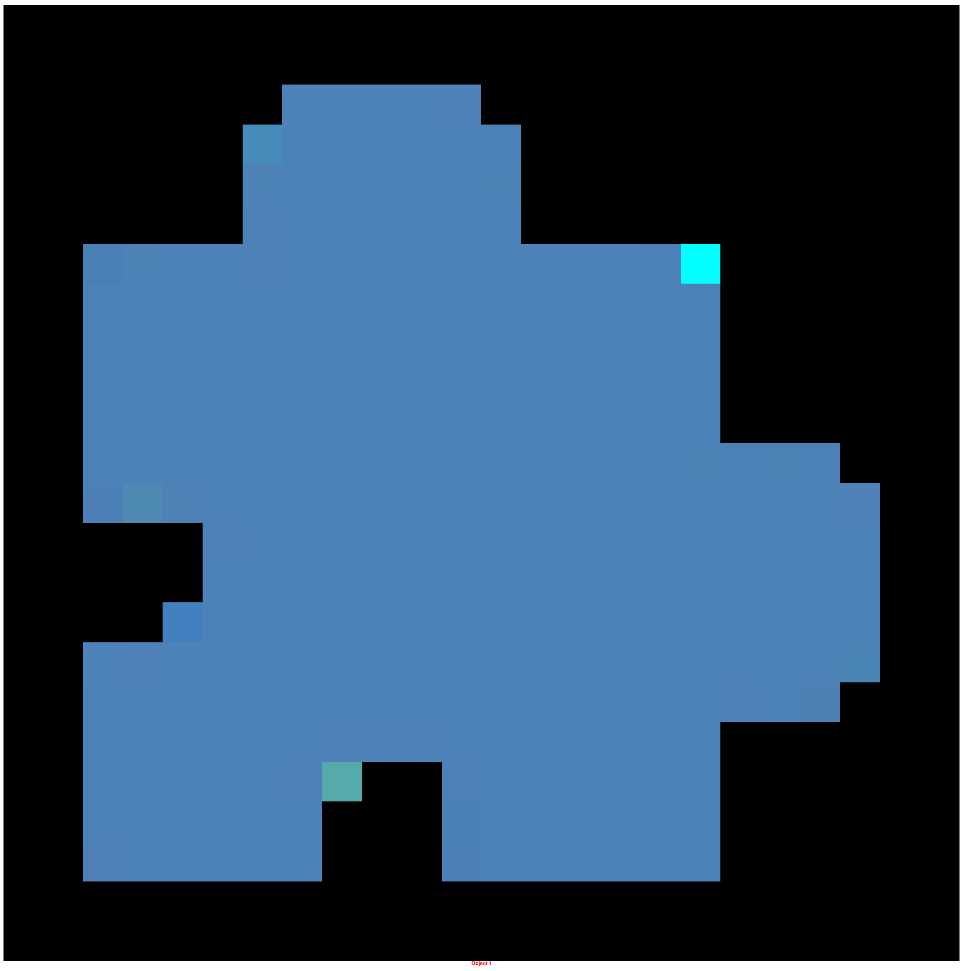
This and the next chapters show how to use PHP to validate form data.

PHP Form Validation

Think SECURITY when processing PHP forms!

These pages 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:



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 textarea. 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>
```

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
```

The Form Element

The HTML code of the form looks like this:

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

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 &lt; and &gt;. 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

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:

1. Strip unnecessary characters (extra space, tab, newline) from the user input data (with the PHP trim() function)
2. 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 over and over again).

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:

Example

```
<?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;
}
?>
```

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.

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

PHP Forms Advanced Validation

In this chapter, we will validate name, email, password, website, description, gender, remember me inputs.

Input Field	Validation
Name	Required. Should only contain letters, numbers and white spaces.
Email	Required. Should be a valid email.
Password	Required. Should be longer than 6 characters.
Website	Optional. If set, should be a valid URL.
Description	Optional. Multi-line text area input.
Gender	Required. Radio Button Input.
Remember Me	True of False. Default is false. Check box input.

Primary Validating Function

In the previous examples, we validated string using both *trim()* and *htmlspecialchars()* function like. *trim(htmlspecialchars($string))*. But, it is a really bad practice for a good developer as it can make errors. To prevent this code repetition error, let's create our own function to do both in one function call.

We will name it *validate()*. This function will remove white spaces and escape html to prevent xss at the same time.

Validate Function

```
<?php
function validate($str) {
        return trim(htmlspecialchars($str));
}

// calling validate function
echo '<pre>';
echo validate('  <script>  ');
echo '</pre>';
```

Complete HTML Form

Here we will create a complete HTML form which has all kinds of input fields.

HTML Form

```
<html>
<head>
        <title></title>
</head>
<body>
<form method="POST" action="">
        Name: <input type="text" name="name"> <br>
        Email: <input type="text" name="email"> <br>
        Password: <input type="password" name="password"> <br>
        Website: <input type="text" name="website"> <br>
        Description: <textarea name="description"></textarea> <br>
        Gender: Male<input type="radio" name="gender" value="male"> Female<input
type="radio" name="gender" value="female"> <br>
        Remember Me: <input type="checkbox" name="remember">
</form>
</body>
</html>
```

Name Validation

This code will check whether the name only contains letters, numbers and white spaces. If it contains invalid characters, the error message will be stored in *$nameError* variable to show later in our form.

```
$name = validate($_POST['name']);
if (!preg_match('/^[a-zA-Z0-9\s]+$/', $name)) {
        $nameError = 'Name can only contain letters, numbers and white spaces';
}
```

Email Validation

We use in-built function *filter_var()* with *FILTER_VALIDATE_EMAIL* flag to validate emails. The *filter_var()* can be used for many purposes. To say that we are using it to validate an email, we have to set the second parameter (called as flag) to *FILTER_VALIDATE_EMAIL*.

```
$email = validate($_POST['email']);
if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
        $emailError = 'Invalid Email';
}
```

Password Validation

We made a rule that password should be longer than 6 characters. We will validate it here.

```
$password = validate($_POST['password']);
if (strlen($password) < 6) {
        $passwordError = 'Please enter a long password';
}
```

URL (Website) Validation

Here we use *filter_var()* function with *FILTER_VALIDATE_URL* flag.s

```
$website = validate($_POST['website']);
if (!filter_var($website, FILTER_VALIDATE_URL)) {
        $websiteError = 'Invalid URL';
}
```

The only validation should be done to description input is, sending the input though the *validate()* function we created earlier.

Then, we need to check whether the gender is set.

Check Box (Remember Me) Validation

Most of browsers set value of check box to "on" if it is checked. We use *filter_var()* function with *FILTER_VALIDATE_BOOLEAN* flag to convert it to boolean. This function will convert "on" to true, which makes later processes easy for us.

```
$remember = validate($_POST['remember']);
$remember = filter_var($remember, FILTER_VALIDATE_BOOLEAN);
// now $remember is a boolean
```

https://tutorials.supunkavinda.blog/php/forms-validation-advanced