CREATING AND USING FORMS

To create a fully functional web application, you need to be able to interact with your users. The common way to receive information from web users is through a form. Forms have evolved to be quite all-encompassing.

On the surface, web forms are merely Hypertext Markup Language (HTML) elements. The way that the elements are processed, however, relies on the processing script that will take care of the elements. PHP 5 is built so that it seamlessly integrates with form elements. Over the past few versions of PHP, its methodology for dealing with form information has gradually evolved and is now quite robust.

UNDERSTANDING COMMON FORM ISSUES:

When dealing with forms, the most important aspect to remember is that you are limited to a certain variety of fields that can be applied to a form. The fields that have been created are non-negotiable and work in only the way they were created to work. It is important, therefore, to fully understand what is available and how best to use the form features to your advantage.

The table given below lists the form elements that are available to you.

Element	Description
TEXT INPUT	A simple text box
PASSWORD INPUT	A text box that hides the characters inputted
HIDDEN INPUT	A field that does not show on the form but can contain data
SELECT	A drop-down box with options
LIST	A select box that can have multiple options selected
CHECKBOX	A box that can be checked
RADIO	A radio button that can act as a choice
TEXTAREA	A larger box that can contain paragraph-style entries
FILE	An element that allows you to browse your computer for a file
SUBMIT	A button that will submit the form
RESET	A button that will reset the form to its original state

GET vs. POST:

When dealing with forms, you must specify the way that the information entered into the form is transmitted to its destination (method=""). The two ways available to a web developer are GET and POST.

When sending data using the GET method, all fields are appended to the Uniform Resource Locator (URL) of the browser and sent along with the address as data. With the POST method, values are sent as standard input.

Sending data using the GET method means that fields are generally capped at 150 characters, which is certainly not the most effective means of passing information. It is also not a secure means of passing data, because many people know how to send information to a script using an address bar.

Sending data using the POST method is quite a bit more secure (because the method cannot be altered by appending information to the address bar) and can contain as much information as you choose to send. Therefore, whenever possible, use the POST method for sending information and then adjust your script to handle it.

PHP 5's current methods for dealing with GET and POST variables are the ***$_GET*** and ***$_POST*** Superglobals, respectively. By using these two Superglobals, you can designate exactly where the information should be coming from and subsequently handle the data in the way you want. The following example shows the difference between using the GET and POST methods.

The Code:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<title>Sample 13.1</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
</head>
<body>
<div style="width: 500px; text-align: left;">
<?php
//Handle incoming data.
//This will trigger if you submit using GET
if ($_GET['submitted'] == "yes"){
if (trim ($_GET['yourname']) != ""){
echo "Your Name (with GET): " . $_GET['yourname'];
} else {
echo "You must submit a value.";
}
?><br /><a href="sample13_1.php">Try Again</a><?php
}
if ($_POST['submitted'] == "yes"){
if (trim ($_POST['yourname']) != ""){
echo "Your Name (with POST): " . $_POST['yourname'];
} else {
echo "You must submit a value.";
}
?><br /><a href="sample13_1.php">Try Again</a><?php
}
?>
<?php
//Show the forms only if you don't already have a submittal.
if ($_GET['submitted'] != "yes" && $_POST['submitted'] != "yes"){
?>
<form action="sample13_1.php" method="get">
<p>GET Example:</p>
<input type="hidden" name="submitted" value="yes" />
Your Name: <input type="text" name="yourname" maxlength="150" /><br />
<input type="submit" value="Submit with GET" style="margin-top: 10px;" />
</form>
<form action="sample13_1.php" method="post">
<p>POST Example:</p>
<input type="hidden" name="submitted" value="yes" />
Your Name: <input type="text" name="yourname" maxlength="150" /><br />
<input type="submit" value="Submit with POST" style="margin-top: 10px;" />
</form>
<?php
}
?>
</div>
</body>
</html>
```

This block of code demonstrates the difference between the GET and POST methods using the two different forms. You should remember a few things when using such code. Specifically, try hitting the Refresh button after submitting data using the POST form. You will note that the browser will ask you if you want to resubmit the data that was passed to it previously. If you want to resend the data, you must select Yes to this option. On the other hand, when using the GET method, you will not be presented with this issue. (The browser will automatically send the data again.)

Superglobals Vs. Globals:

Before the advent of Superglobals, data was passed along from script to script with loose security. In the php.ini file, you can change a value called ***register_globals*** to either on or off. If you leave it on, then whenever you pass a value using the GET or POST method, you can access the variable simply by putting an ampersand (&) character in front of the name of the element you are passing. The problem with this method is that malicious users can insert values into your code to bypass the form entirely.

Therefore, if you want your code to be as secure as possible, you should definitely code your applications with register_globals turned off and ensure that you receive your values from where you expect them to come. Using Superglobals allows you to do this.

The following example shows how you can submit values using globals or superglobals.

Note that for this example to work properly, you must temporarily switch your

register_globals value to on (don't forget to turn it off afterward!).

The Code:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<title>Sample 13.2</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
</head>
<body>
<div style="width: 500px; text-align: left;">
<?php
//Handle the incoming data.
//Here is how you could handle it with register_globals turned on.
if ($submitted == "yes"){
```

```
if (trim ($yourname) != ""){
echo "Your Name: $yourname.";
} else {
echo "You must submit a value.";
}
?><br /><a href="sample13_2.php">Try Again</a><br /><?php
}
//Now, here is how it SHOULD be handled with register_globals turned off.
if ($_POST['submitted'] == "yes"){
if (trim ($_POST['yourname']) != ""){
echo "Your Name: " . $_POST['yourname'] . ".";
} else {
echo "You must submit a value.";
}
?><br /><a href="sample13_2.php">Try Again</a><br /><?php
}
?>
<?php
//Show the forms only if you don't already have a submittal.
if ($_POST['submitted'] != "yes"){
?>
<form action="sample13_2.php" method="post">
<p>Example:</p>
<input type="hidden" name="submitted" value="yes" />
Your Name: <input type="text" name="yourname" maxlength="150" /><br />
<input type="submit" value="Submit" style="margin-top: 10px;" />
</form>
<?php
}
?>
</div>
</body>
</html>
```

VALIDATING FORM INPUT:

In this day and age of constant attacks on websites, one of the biggest issues is attacking forms directly. To ensure a suitable submission of form data, validation is key. You have many ways to validate a form and many form elements to consider. Generally, you need to determine what qualities you want a piece of data to adhere to and then ensure that the submitted data comes in the correct form. If the data comes in a format that is not to your liking, you must be ready to take care of this. The following example shows a few examples of form validation using PHP.

The Code:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<title>Sample 13.3</title>
```

```
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
</head>
<body>
<div style="width: 500px; text-align: left;">
<?php
//Function to determine a valid e-mail address
function validemail($email){
return     preg_match("/^([a-zA-Z0-9])+([.a-zA-Z0-9_-])*@([a-zA-Z0-9_-])+(.[a-zA-Z0-9_-]+)+[a-zA-Z0-9_-
]$/",$email);
}
//Handle the incoming data.
if ($_POST['submitted'] == "yes"){
//Let's declare a submission value that tells you if you are fine.
$goodtogo = true;
//Validate the name.
try {
if (trim ($_POST['yourname']) == ""){
$goodtogo = false;
throw new exception ("Sorry, you must enter your name.<br />");
}
} catch (exception $e) {
echo $e->getmessage();
}
//Validate the select box.
try {
if ($_POST['myselection'] == "nogo"){
$goodtogo = false;
throw new exception ("Please make a selection.<br />");
}
} catch (exception $e) {
echo $e->getmessage();
}
//And lastly, validate for a proper e-mail addy.
try {
if (!validemail (trim ($_POST['youremail']))){
$goodtogo = false;
throw new exception ("Please enter a valid email address.<br />");
}
} catch (exception $e) {
echo $e->getmessage();
}
//Now, if there were no errors, you can output the results.
if ($goodtogo){
echo "Your Name: " . $_POST['yourname'] . "<br />";
echo "Your Selection: " . $_POST['myselection'] . "<br />";
echo "Your Email Address: " . $_POST['youremail'] . "<br />";
}
?><br /><a href="sample13_3.php">Try Again</a><br /><?php
}
?>
<?php
//Show the forms only if you don't already have a submittal.
if ($_POST['submitted'] != "yes"){
?>
<form action="sample13_3.php" method="post">
<p>Example:</p>
<input type="hidden" name="submitted" value="yes" />
Your Name: <input type="text" name="yourname" maxlength="150" /><br /><br />
Selection:
<select name="myselection">
<option value="nogo">make a selection...</option>
<option value="1">Choice 1</option>
<option value="2">Choice 2</option>
<option value="3">Choice 3</option>
</select><br /><br />
Your Email: <input type="text" name="youremail" maxlength="150" /><br />
<input type="submit" value="Submit" style="margin-top: 10px;" />
</form>
<?php
}
?>
</div>
</body>
</html>
```

Since, for this example, you have chosen three types of fields, it is important to take care of them in individual ways. For this example, you want to receive a name value that will not be blank, a selected value that must not be the default, and an e-mail address that must be in the proper format. To make sure you do not have a blank field, you can validate the name value by ensuring that it does not equal a blank string. In the case of the selection, if the user has not chosen a different value than the default, the value will be a *nogo*, against which you can then validate. For the last value, the e-mail address, you use a regular expression to ensure that the e-mail address is properly formatted. By using this type of validation, you ensure that all the submitted values are in the format you need.

WORKING WITH MULTIPAGE FORMS:

Sometimes you will need to collect values from more than one page. Most developers do this for the sake of clarity. By providing forms on more than one page, you can separate blocks of information and thus create an ergonomic experience for the user. The problem, therefore, is how to get values from each page onto the next page and finally to the processing script. Being the great developer that you are, you can solve this problem and use the hidden input form type. When each page loads, you merely load the values from the previous pages into hidden form elements and submit them.

The Code:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
```

```
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<title>Sample 13.4 Page 1</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
</head>
<body>
<div style="width: 500px; text-align: left;">
<form action="sample13_4_page2.php" method="post">
<p>Page 1 Data Collection:</p>
<input type="hidden" name="submitted" value="yes" />
Your Name: <input type="text" name="yourname" maxlength="150" /><br /><br />
<input type="submit" value="Submit" style="margin-top: 10px;" />
</form>
</div>
</body>
</html>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<title>Sample 13.4 Page 2</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
</head>
<body>
<div style="width: 500px; text-align: left;">
<form action="sample13_4_page3.php" method="post">
<p>Page 2 Data Collection:</p>
Selection:
<select name="yourselection">
<option value="nogo">make a selection...</option>
<option value="1">Choice 1</option>
<option value="2">Choice 2</option>
<option value="3">Choice 3</option>
</select><br /><br />
<input type="hidden" name="yourname" value="<?php echo $_POST['yourname']; ?>" />
<input type="submit" value="Submit" style="margin-top: 10px;" />
</form>
</div>
</body>
</html>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<title>Sample 13.4 Page 3</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
</head>
```

```
<body>
<div style="width: 500px; text-align: left;">
<form action="sample13_4_page4.php" method="post">
<p>Page 3 Data Collection:</p>
Your Email: <input type="text" name="youremail" maxlength="150" /><br />
<input type="hidden" name="yourname"  value="<?php echo $_POST['yourname']; ?>" />
<input type="hidden" name="yourselection" value="<?php echo _POST['yourselection']; ?>" />
<input type="submit" value="Submit" style="margin-top: 10px;" />
</form>
</div>
</body>
</html>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<title>Sample 13.4 Page 4</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
</head>
<body>
<div style="width: 500px; text-align: left;">
<?php
//Display the results.
echo "Your Name: " . $_POST['yourname'] . "<br />";
echo "Your Selection: " . $_POST['yourselection'] . "<br />";
echo "Your Email: " . $_POST['youremail'] . "<br />";
?>
<a href="sample13_4_page1.php">Try Again</a>
</div>
</body>
</html>
```

Redisplaying Forms with Preserved Information and Error Messages:

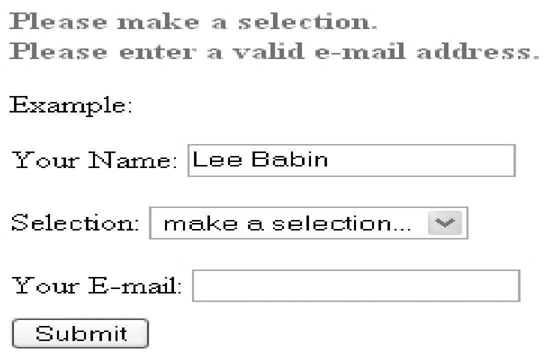
When receiving information submitted from a user, the information may not be submitted in the format you need. To ensure that users do not get frustrated, it is important to inform them of what they did wrong and clearly tell them how to fix the problem. It is also bad practice to force users to completely rewrite all the proper information they may have already submitted on the form. If users are forced to do redundant work, they may become irritated and potentially disregard your service altogether. Therefore, to keep users happy, it is important to validate properly and clearly while keeping matters as simple for them as possible.

The Code:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<title>Sample 13.5</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<style>
```

```
.error {
font-weight: bold;
color: #FF0000;
}
</style>
</head>
<body>
<div style="width: 500px; text-align: left;">
<?php
//Function to determine a valid e-mail address
function validemail($email){
return      preg_match("/^([a-zA-Z0-9])+([.a-zA-Z0-9_-])*@([a-zA-Z0-9_-])+(.[a-zA-Z0-9_-]+)+[a-zA-Z0-9_-
]$/",$email);
}
//Default to showing the form.
$goodtogo = false;
//Handle the incoming data.
if ($_POST['submitted'] == "yes"){
//Let's declare a submission value that tells you if you are fine.
$goodtogo = true;
//Validate the name.
try {
if (trim ($_POST['yourname']) == ""){
$goodtogo = false;
throw new exception ("Sorry, you must enter your name.<br />");
}
} catch (exception $e) {
?><span class="error"><?php echo $e->getmessage(); ?></span><?php
}
//Validate the select box.
try {
if ($_POST['myselection'] == "nogo"){
$goodtogo = false;
throw new exception ("Please make a selection.<br />");
}
} catch (exception $e) {
?><span class="error"><?php echo $e->getmessage(); ?></span><?php
}
//And lastly, validate for a proper e-mail addy.
try {
if (!validemail (trim ($_POST['youremail']))){
$goodtogo = false;
throw new exception ("Please enter a valid e-mail address.<br />");
}
} catch (exception $e) {
?><span class="error"><?php echo $e->getmessage(); ?></span><?php
}
//Now, if there were no errors, you can output the results.
if ($goodtogo){
echo "Your Name: " . $_POST['yourname'] . "<br />";
echo "Your Selection: " . $_POST['myselection'] . "<br />";
echo "Your E-mail Address: " . $_POST['youremail'] . "<br />";
?><br /><a href="sample13_5.php">Try Again</a><br /><?php
}
}
//Show the forms only if you do not have all the valid information.
if (!$goodtogo){
?>
<form action="sample13_5.php" method="post">
<p>Example:</p>
<input type="hidden" name="submitted" value="yes" />
Your Name: <input type="text" name="yourname" maxlength="150" value="<?php echo
$_POST['yourname']; ?>" /><br /><br />
Selection:
<select name="myselection">
<option value="nogo">make a selection...</option>
<option value="1"<?php if ($_POST['myselection'] == 1){?> selected="selected"<?php } ?>>Choice
1</option>
<option value="2"<?php if ($_POST['myselection'] == 2){?> selected="selected"<?php } ?>>Choice
2</option>
<option value="3"<?php if ($_POST['myselection'] == 3){?> selected="selected"<?php } ?>>Choice
3</option>
</select><br /><br />
Your Email: <input type="text" name="youremail" maxlength="150" value="<?php echo
$_POST['youremail']; ?>" /><br />
<input type="submit" value="Submit" style="margin-top: 10px;" />
</form>
<?php
}
?>
</div>
</body>
</html>
```

The following figure shows the potential output if you input a valid name field but leave the selection and e-mail address empty.



Please make a selection.
Please enter a valid e-mail address.

Example:

Your Name: Lee Babin

Selection: make a selection...

Your E-mail:

PREVENTING MULTIPLE SUBMISSIONS OF A FORM:

One possible occurrence that happens often is that users become impatient when waiting for your script to do what it is doing, and hence they click the submit button on a form repeatedly. This can wreak havoc on your script because, while the user may not see anything happening, your script is probably going ahead with whatever it has been programmed to do. Of particular danger are credit card number submittals. If a user continually hits the submit button on a credit card submittal form, their card may be charged multiple times if the developer has not taken the time to validate against such an eventuality.

Preventing Multiple Submissions on the Server Side:

You can deal with multiple submittal validation in essentially two ways. The first occurs on the server. *Server* side refers to a script located on the server that is receiving the data; *client* side is more browsers related.

Because the server has no actual access to the browser, validating multiple submissions can be a bit trickier. While you can accomplish this goal in a number of ways from a server-side perspective, we prefer to use a session-based method. Basically, once the submit button has been clicked; the server logs the request from the individual user. If the user attempts to resubmit a request, the script notes a request is already in motion from this user and denies the subsequent request. Once the script has finished processing, the session is unset, and you have no more worries.

For the following example, you will need a test.txt text file that you can create and place relative to the script.

The Code:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<title>Sample 13.6</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
</head>
<body>
<div style="width: 500px; text-align: left;">
<form action="sample13_6_process.php" method="post">
<p>Example:</p>
<input type="hidden" name="submitted" value="yes" />
Your Name: <input type="text" name="yourname" maxlength="150" /><br />
<input type="submit" value="Submit" style="margin-top: 10px;" />
</form>
</div>
</body>
</html>
<?php
//Start the session state.
session_start ();
//Set a session started value for this user.
if (!isset ($_SESSION['processing'])){
$_SESSION['processing'] = false;
}
```

```
//Now you ensure you haven't already started processing the request.
if ($_SESSION['processing'] == false){
//Now, you let the script know that you are processing.
$_SESSION['processing'] = true;
//Create a loop that shows the effect of some heavy processing.
for ($i = 0; $i < 2000000; $i++){
//Thinking...
}
//Every time you do this, write to a text file so you can test that
//the script isn't getting hit with multiple submissions.
if ($file = fopen ("test.txt", "w+")){
fwrite ($file, "Processing");
} else {
echo "Error opening file.";
}
//Then you start doing the calculations.
echo $_POST['yourname'];
//Then, once you have finished calculating, you can kill the session.
unset ($_SESSION['processing']);
}
?>
```

Preventing Multiple Submissions on the Client Side:

Handling multiple submittals from a client-side perspective is actually much simpler than doing it on the server side. With well-placed JavaScript, you can ensure that the browser will not let the submittal go through more than once. The problem with this method, of course, is that JavaScript is not always foolproof because of the user's ability to turn it off. That being said, however, most users will have JavaScript enabled, so this script will likely work for 90 percent of web users. The following example uses JavaScript to cut off multiple submittals from a client-side (browser) level.

The Code

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<title>Sample 13.7</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<script language="javascript" type="text/javascript">
<!--
function checkandsubmit() {
//Disable the submit button.
document.test.submitbut.disabled = true;
//Then submit the form.
document.test.submit();
}
//-->
</script>
</head>
```

```
<body>
<div style="width: 500px; text-align: left;">
<form action="sample13_6_process.php" method="post" name="test" onsubmit="return checkandsubmit ()">
<p>Example:</p>
<input type="hidden" name="submitted" value="yes" />
Your Name: <input type="text" name="yourname" maxlength="150" /><br />
<input type="submit" value="Submit" style="margin-top: 10px;"  id="submitbut" name"submitbut" />
</form>
</div>
</body>
</html>
<?php
//Create a loop that shows the effect of some heavy processing.
for ($i = 0; $i < 2000000; $i++){
//Thinking...
}
//Every time you do this, let's write to a text file so you can test that
//out script isn't getting hit with multiple submissions.
if ($file = fopen ("test.txt", "w+")){
fwrite ($file, "Processing");
} else {
echo "Error opening file.";
}
//Then you start doing the calculations.
echo $_POST['yourname'];
?>
```