**Unit VII: Working With Form                    (10)**

**7.1 Creating a user Form**

**Refer to LAB task One and LAB task TWO**

**7.2 Hidden filed for save state**

Hidden elements can be used to maintain state on the client side. To do this we essentially have to do two things.

* Make sure that all our pages have forms with hidden elements.
* Make sure that the values are passed using submit buttons (Or the JavaScript equivalent **submit()**).

In practice the latter can be a problem, because navigation by buttons can make a form look clunky. However it is relatively easy to get round this problem by using a bit of JavaScript magic! For the purpose of this page we are going to use the POST method for passing information between the various forms

**Example**

<?php  
  $num\_to\_guess = 42;  
  $message = "";  
  **if**(! isset ( $\_POST ['guess'] )) {  
    $message = "Welcome!";  
  } **else if**($\_POST ['guess'] > $num\_to\_guess) {  
    $message = $\_POST ['guess'] . " is too big!";  
  } **else if**($\_POST ['guess'] < $num\_to\_guess) {  
    $message = $\_POST ['guess'] . " is too small!";  
  } **else**{  
    $message = "Well done!";  
  }  
  $guess = ( **int**) $\_POST ['guess'];  
  $num\_tries = ( **int**) $\_POST ['num\_tries'];  
  $num\_tries ++;  
  ?>  
<**html**>  
<head>  
<title>A PHP Number Guessing Script</title>  
</head>  
<**body**>  
 <?php print $message?>  
 Guess number: <?php print $num\_tries?><br />  
<**form**method="post" action="<?php  
print $\_SERVER ['PHP\_SELF']?>">  
<input type="hidden" name="num\_tries"  
  value="<?php  
  print $num\_tries?>" /> Type your guess here: <input type="text" name="guess" value="<?php  
  print $guess?>" />  
</**form**>  
</**body**>  
</**html**>

Multi-page forms using hidden input fields

Hidden input fields are form fields that are not visible. The user can’t see or change these fields, and they are used to transmit state information between different pages. Let’s use hidden fields to transport our data across our form, to the final processing script.  
We start with the same form for step 1:

|  |
| --- |
| <form method="post" action="form2.php">      <input type="text" name="name">      <input type="text" name="email\_address">      <input type="submit" value="Go To Step 2">  </form> |

Ok, so nothing more than 2 input fields and a submit button to take us to step 2. In the following page, apart from the HTML form to gather membership data, we are going to need code to store the submitted data from step 1 in the session.

|  |
| --- |
| <form method="post" action="form3.php">  <input type="radio" group="membership\_type" value="Free">    <input type="radio" group="membership\_type" value="Normal">    <input type="radio" group="membership\_type" value="Deluxe">    <input type="checkbox" name="terms\_and\_conditions">    <input type="hidden" name="name"  value="<?php echo $\_POST['name']; ?>">    <input type="hidden" name="email\_address"   value="<?php echo $\_POST['email\_address']; ?>">    <input type="submit" value="Go To Step 3">  </form> |

Please note that although the hidden fields are not visible to the visitor, they are visible in source of the page, so you must not use them to store critical information.

Now our form contains all the data entered so far, which we are going to post to the third part of the form.

|  |
| --- |
| <form method="post" action="form\_process.php">    <input type="text" name="name\_on\_card">    <input type="text" name="credit\_card\_number">    <input type="text" name="credit\_card\_expiration\_date">    <input type="hidden" name="name"   value="<?php echo $\_POST['name']; ?>">    <input type="hidden" name="email\_address"   value="<?php echo $\_POST['email\_address'];    <input type="hidden" name="membership\_type"   value="<?php echo $\_POST['membership\_type']; ?>">    <input type="hidden" name="terms\_and\_conditions"    value="<?php echo $\_POST['terms\_and\_conditions]; ?>">    <input type="submit" value="Finish">  </form> |

Now that we have all our fields available, let’s proceed to the form processing.

|  |
| --- |
| <?php    //let's create the query  $insert\_query = "insert into subscriptions (".  "name,".  "email\_address,".  "membership\_type".  "terms\_and\_conditions,".  "name\_on\_card,".  "credit\_card\_number,".  "credit\_card\_expiration\_data".  ") values (".  "'" . $\_POST['name'] . "', ".  "'" . $\_POST['email\_address'] . "',".  "'" . $\_POST['membership\_type'] . "',".  "'" . $\_POST['terms\_and\_conditions'] . "',".  "'" . $\_POST['name\_on\_card'] . "',".  "'" . $\_POST['credit\_card\_number'] . "',".  "'" . $\_POST['credit\_card\_expiration'] ."' )";    //let's run the query  mysql\_query($insert\_query);    ?> |

And that’s it. Please note that this time, we have all the data available in the $\_POST array.

A good idea would be to use form validation for each step of the form, so you don’t end up with incomplete data in the database.

Now you know two methods of creating multipart forms, to create better user-experience for your visitors.

**7.3  Redirecting user**

On a Web site, **redirection** is a technique for moving visitors to a different Web page than the one they request, usually because the page requested is unavailable. Web users often encounter **redirection** when they visit the Web site of a company whose name has been changed or which has been acquired by another company.

You can use a simple PHP script to redirect a user from the page they entered to a different web page. One reason you may want to do this is that the page they are trying to access no longer exists. Using this method, they can be seamlessly transfered to the new page without having to click a link to continue.

* Users are redirected quickly and seamlessly
* When using the 'Back' button, the user is taken to the last viewed page, not the redirect page
* Works on all browsers

**Redirecting Example 1**

<!DOCTYPE html>

<html>

<head>

<meta http-equiv="Refresh" content="5;url=http://www.w3schools.com">

</head>

<body>

<p>Sorry! We have moved! The new URL is: <a href="http://www.w3schools.com">http://www.w3schools.com</a></p>

<p>You will be redirected to the new address in five seconds.</p>

<p>If you see this message for more than 5 seconds, please click on the link above!</p>

</body>

</html>

**7.4  Sending mail on form submission**

# Simple Form to Email PHP Contact Form

Below you can find a basic website form using only HTML (for the form) and PHP for the form processing.

Along-side the HTML form you will find a basic PHP script which will capture the form submissions and send the form contents to your email address.

The form we provide below is a "bare-bone" version. This will allow you to edit the form to fit with your own website design. You could edit the form in dreamweaver, HTML-Kit, iweb or any other HTML editor of your choice.

# Simple Form to Email PHP Contact Form

Below you can find a basic website form using only HTML (for the form) and PHP for the form processing.

Along-side the HTML form you will find a basic PHP script which will capture the form submissions and send the form contents to your email address.

The form we provide below is a "bare-bone" version. This will allow you to edit the form to fit with your own website design. You could edit the form in dreamweaver, HTML-Kit, iweb or any other HTML editor of your choice.

**What is an email header?**

An email consists of three vital components: the envelope, the header(s), and the body of the message. The envelope is something that an email user will never see since it is part of the internal process by which an email is routed. The body is the part that we always see as it is the actual content of the message contained in the email. The header(s), the third component of an email, is perhaps a little more difficult to explain, though it is arguably the most interesting part of an email.

**Header**

In an e-mail, the body (content text) is always preceded by header lines that identify particular routing information of the message, including the sender, recipient, date and subject. Some headers are mandatory, such as the FROM, TO and DATE headers. Others are optional, but very commonly used, such as SUBJECT and CC. Other headers include the sending time stamps and the receiving time stamps of all mail transfer agents that have received and sent the message. In other words, any time a message is transferred from one user to another (i.e. when it is sent or forwarded), the message is date/time stamped by a mail transfer agent (MTA) - a computer program or software agent that facilitates the transfer of email message from one computer to another. This date/time stamp, like FROM, TO, and SUBJECT, becomes one of the many headers that precede the body of an email.

To really understand what an email header is, you must see one. Here is an example of a full email header\*:

Return-Path: <example\_from@dc.edu>

X-SpamCatcher-Score: 1 [X]

Received: from [136.167.40.119] (HELO dc.edu)

by fe3.dc.edu (CommuniGate Pro SMTP 4.1.8)

with ESMTP-TLS id 61258719 for example\_to@mail.dc.edu; Mon, 23 Aug 2004 11:40:10 -0400

Message-ID: <4129F3CA.2020509@dc.edu>

Date: Mon, 23 Aug 2005 11:40:36 -0400

From: Taylor Evans <example\_from@dc.edu>

User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.0.1) Gecko/20020823 Netscape/7.0

X-Accept-Language: en-us, en

MIME-Version: 1.0

To: Jon Smith <example\_to@mail.dc.edu>

Subject: Business Development Meeting

Content-Type: text/plain; charset=us-ascii; format=flowed

Content-Transfer-Encoding: 7bit

\* email headers should always be read from bottom to top.

Fortunately, most of this information is hidden inside the email with only the most relevant or mandatory headers appearing to the user. Those headers that we most often see and recognize are bolded in the above example.

**Header Characteristics**

A single email header has some important characteristics, including perhaps the most important part of an email - this is the KEY:VALUE pairs contained in the header. Looking at the above, you can tell some of the KEY:VALUE pairs used. Here is a breakdown of the most commonly used and viewed headers, and their values:

* From: sender's name and email address (IP address here also, but hidden)
* To: recipient's name and email address
* Date: sent date/time of the email
* Subject: whatever text the sender entered in the Subject heading before sending

**Headers Provide Routing Information**

Besides the most common identifications (from, to, date, subject), email headers also provide information on the route an email takes as it is transferred from one computer to another. As mentioned earlier, mail transfer agents (MTA) facilitate email transfers. When an email is sent from one computer to another it travels through a MTA. Each time an email is sent or forwarded by the MTA, it is stamped with a date, time and recipient. This is why some emails, if they have had several destinations, may have several RECEIVED headers: there have been multiple recipients since the origination of the email. In a way it is much like the same way the post office would route a letter: every time the letter passes through a post office on its route, or if it is forwarded on, it will receive a stamp. In this case the stamp is an email header.

When viewed in their entirety, these multiple recipient headers will look like this in an email:

Received: from tom.bath.dc.uk ([138.38.32.21] ident=yalrla9a1j69szla2ydr)

by steve.wrath.dc.uk with esmtp (Exim 3.36 #2)id 19OjC3-00064B-00

for example\_to@imaps.bath.dc.uk; Sat, 07 Jun 2005 20:17:35 +0100

Received: from write.example.com ([205.206.231.26])

by tom.wrath.dc.uk with esmtp id 19OjBy-0001lb-3V

for example\_to@bath.ac.uk; Sat, 07 Jun 2005 20:17:30 +0100

Received: from master.example.com (lists.example.com [205.206.231.19])

by write.example.com (Postfix) with QMQP

id F11418F2C1; Sat, 7 Jun 2005 12:34:34 -0600 (MDT)

In the example shown above, there are three Received: stamps. Reading from the bottom upwards, you can see who sent the message first, next and last, and you can see when it was done. This is because every MTA that processed the email message added a Received: line to the email's header. These Received: lines provide information on where the message originated and what stops it made (what computers) before reaching its final destination. As the example shows, these Received: lines provide the email and IP address of each sender and recipient. They also provide the date and time of each transfer. The lines also indicate if the email address was part of an email list. It is all this information that is valued by computer programmers and IT department associates when making efforts to track and stop SPAM email message. And it is this information that arguable makes headers the most important part of an email.

PHP must be configured correctly in the **php.ini** file with the details of how your system sends email. Open php.ini file available in **/etc/** directory and find the section headed **[mail function]**.

Windows users should ensure that two directives are supplied. The first is called SMTP that defines your email server address. The second is called sendmail\_from which defines your own email address.

The configuration for Windows should look something like this:

|  |
| --- |
| [mail function]  ; For Win32 only.  SMTP = smtp.secureserver.net  ; For win32 only  sendmail\_from = webmaster@tutorialspoint.com |

Linux users simply need to let PHP know the location of their **sendmail** application. The path and any desired switches should be specified to the sendmail\_path directive.

The configuration for Linux should look something like this:

|  |
| --- |
| [mail function]  ; For Win32 only.  SMTP =  ; For win32 only  sendmail\_from =  ; For Unix only  sendmail\_path = /usr/sbin/sendmail -t -i |

Now you are ready to go:

## Sending plain text email:

PHP makes use of **mail()** function to send an email. This function requires three mandatory arguments that specify the recipient's email address, the subject of the the message and the actual message additionally there are other two optional parameters.

|  |
| --- |
| mail( to, subject, message, headers, parameters ); |

Here is the description for each parameters.

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| to | Required. Specifies the receiver / receivers of the email |
| subject | Required. Specifies the subject of the email. This parameter cannot contain any newline characters |
| message | Required. Defines the message to be sent. Each line should be separated with a LF (\n). Lines should not exceed 70 characters |
| headers | Optional. Specifies additional headers, like From, Cc, and Bcc. The additional headers should be separated with a CRLF (\r\n) |
| parameters | Optional. Specifies an additional parameter to the sendmail program |

As soon as the mail function is called PHP will attempt to send the email then it will return true if successful or false if it is failed.

Multiple recipients can be specified as the first argument to the mail() function in a comma separated list.

## Example:

**Contact.php**

<!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">

<head>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

<title>Untitled Document</title>

</head>

<body>

<form name="contactform" method="post" action="send\_form\_email.php">

<table width="450px">

<tr>

<td valign="top">

<label for="first\_name">First Name \*</label>

</td>

<td valign="top">

<input type="text" name="first\_name" maxlength="50" size="30">

</td>

</tr>

<tr>

<td valign="top"">

<label for="last\_name">Last Name \*</label>

</td>

<td valign="top">

<input type="text" name="last\_name" maxlength="50" size="30">

</td>

</tr>

<tr>

<td valign="top">

<label for="email">Email Address \*</label>

</td>

<td valign="top">

<input type="text" name="email" maxlength="80" size="30">

</td>

</tr>

<tr>

<td valign="top">

<label for="telephone">Telephone Number</label>

</td>

<td valign="top">

<input type="text" name="telephone" maxlength="30" size="30">

</td>

</tr>

<tr>

<td valign="top">

<label for="comments">Comments \*</label>

</td>

<td valign="top">

<textarea name="comments" maxlength="1000" cols="25" rows="6"></textarea>

</td>

</tr>

<tr>

<td colspan="2" style="text-align:center">

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

</td>

</tr>

</table>

</form>

</body>

</html>

**send\_form\_email.php**

<?php

if(isset($\_POST['email'])) {

// EDIT THE 2 LINES BELOW AS REQUIRED

$email\_to = "namastenabin@gmail.com";

$email\_subject = "New test mail";

function died($error) {

// your error code can go here

echo "We are very sorry, but there were error(s) found with the form you submitted. ";

echo "These errors appear below.<br /><br />";

echo $error."<br /><br />";

echo "Please go back and fix these errors.<br /><br />";

die();

}

// validation expected data exists

if(!isset($\_POST['first\_name']) ||

!isset($\_POST['last\_name']) ||

!isset($\_POST['email']) ||

!isset($\_POST['telephone']) ||

!isset($\_POST['comments'])) {

died('We are sorry, but there appears to be a problem with the form you submitted.'); }

$first\_name = $\_POST['first\_name']; // required

$last\_name = $\_POST['last\_name']; // required

$email\_from = $\_POST['email']; // required

$telephone = $\_POST['telephone']; // not required

$comments = $\_POST['comments']; // required

$error\_message = "";

$email\_exp = '/^[A-Za-z0-9.\_%-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,4}$/';

if(!preg\_match($email\_exp,$email\_from)) {

$error\_message .= 'The Email Address you entered does not appear to be valid.<br />';

}

$string\_exp = "/^[A-Za-z .'-]+$/";

if(!preg\_match($string\_exp,$first\_name)) {

$error\_message .= 'The First Name you entered does not appear to be valid.<br />';

}

if(!preg\_match($string\_exp,$last\_name)) {

$error\_message .= 'The Last Name you entered does not appear to be valid.<br />';

}

if(strlen($comments) < 2) {

$error\_message .= 'The Comments you entered do not appear to be valid.<br />';

}

if(strlen($error\_message) > 0) {

died($error\_message);

}

$email\_message = "Form details below.\n\n";

function clean\_string($string) {

$bad = array("content-type","bcc:","to:","cc:","href");

return str\_replace($bad,"",$string);

}

$email\_message .= "First Name: ".clean\_string($first\_name)."\n";

$email\_message .= "Last Name: ".clean\_string($last\_name)."\n";

$email\_message .= "Email: ".clean\_string($email\_from)."\n";

$email\_message .= "Telephone: ".clean\_string($telephone)."\n";

$email\_message .= "Comments: ".clean\_string($comments)."\n";

// create email headers

$headers = 'From: '.$email\_from."\r\n".

'Reply-To: '.$email\_from."\r\n" .

'X-Mailer: PHP/' . phpversion();

@mail($email\_to, $email\_subject, $email\_message, $headers);

?>

<!-- include your own success html here -->

Thank you for contacting us. We will be in touch with you very soon.

<?php

}

?>

## Sending attachments with email:

To send an email with mixed content requires to set **Content-type** header to **multipart/mixed**. Then text and attachment sections can be specified within **boundaries**.

A boundary is started with two hyphens followed by a unique number which can not appear in the message part of the email. A PHP function **md5()** is used to create a 32 digit hexadecimal number to create unique number. A final boundary denoting the email's final section must also end with two hyphens.

Attached files should be encoded with the **base64\_encode()** function for safer transmission and are best split into chunks with the **chunk\_split()** function. This adds **\r\n** inside the file at regular intervals, normally every 76 characters.

Following is the example which will send a file **/tmp/test.txt** as an attachment. you can code your program to receive an uploaded file and send it.

|  |
| --- |
| <html>  <head>  <title>Sending attachment using PHP</title>  </head>  <body>  <?php  $to = "xyz@somedomain.com";  $subject = "This is subject";  $message = "This is test message.";  # Open a file  $file = fopen( "/tmp/test.txt", "r" );  if( $file == false )  {  echo "Error in opening file";  exit();  }  # Read the file into a variable  $size = filesize("/tmp/test.txt");  $content = fread( $file, $size);  # encode the data for safe transit  # and insert \r\n after every 76 chars.  $encoded\_content = chunk\_split( base64\_encode($content));    # Get a random 32 bit number using time() as seed.  $num = md5( time() );  # Define the main headers.  $header = "From:xyz@somedomain.com\r\n";  $header .= "MIME-Version: 1.0\r\n";  $header .= "Content-Type: multipart/mixed; ";  $header .= "boundary=$num\r\n";  $header .= "--$num\r\n";  # Define the message section  $header .= "Content-Type: text/plain\r\n";  $header .= "Content-Transfer-Encoding:8bit\r\n\n";  $header .= "$message\r\n";  $header .= "--$num\r\n";  # Define the attachment section  $header .= "Content-Type: multipart/mixed; ";  $header .= "name=\"test.txt\"\r\n";  $header .= "Content-Transfer-Encoding:base64\r\n";  $header .= "Content-Disposition:attachment; ";  $header .= "filename=\"test.txt\"\r\n\n";  $header .= "$encoded\_content\r\n";  $header .= "--$num--";  # Send email now  $retval = mail ( $to, $subject, "", $header );  if( $retval == true )  {  echo "Message sent successfully...";  }  else  {  echo "Message could not be sent...";  }  ?>  </body>  </html> |

**7.5  Working with file uploading**

A PHP script can be used with a HTML form to allow users to upload files to the server. Initially files are uploaded into a temporary directory and then relocated to a target destination by a PHP script.

Information in the **phpinfo.php** page describes the temporary directory that is used for file uploads as **upload\_tmp\_dir** and the maximum permitted size of files that can be uploaded is stated as **upload\_max\_filesize**. These parameters are set into PHP configuration file **php.ini**

The process of uploading a file follows these steps

* The user opens the page containing a HTML form featuring a text files, a browse button and a submit button.
* The user clicks the browse button and selects a file to upload from the local PC.
* The full path to the selected file appears in the text filed then the user clicks the submit button.
* The selected file is sent to the temporary directory on the server.
* The PHP script that was specified as the form handler in the form's action attribute checks that the file has arrived and then copies the file into an intended directory.
* The PHP script confirms the success to the user.

As usual when writing files it is necessary for both temporary and final locations to have permissions set that enable file writing. If either is set to be read-only then process will fail.

An uploaded file could be a text file or image file or any document.

**Creating an upload form:**

The following HTM code below creates an uploader form. This form is having method attribute set to **post** and enctype attribute is set to **multipart/form-data**

|  |
| --- |
| <html>  <head>  <title>File Uploading Form</title>  </head>  <body>  <h3>File Upload:</h3>  Select a file to upload: <br />  <form action="/php/file\_uploader.php" method="post"  enctype="multipart/form-data">  <input type="file" name="file" size="50" />  <br />  <input type="submit" value="Upload File" />  </form>  </body>  </html> |

This will display following result:

|  |
| --- |
| **File Upload:**  Select a file to upload:      NOTE: This is just dummy form and would not work. |

**Creating an upload script:**

There is one global PHP variable called **$\_FILES**. This variable is an associate double dimension array and keeps all the information related to uploaded file. So if the value assigned to the input's name attribute in uploading form was **file**, then PHP would create following five variables:

* **$\_FILES['file']['tmp\_name']-** the uploaded file in the temporary directory on the web server.
* **$\_FILES['file']['name'] -** the actual name of the uploaded file.
* **$\_FILES['file']['size'] -** the size in bytes of the uploaded file.
* **$\_FILES['file']['type'] -** the MIME type of the uploaded file.
* **$\_FILES['file']['error'] -** the error code associated with this file upload.

The following example below attempts to copy a file uploaded by the HTML Form listed in previous section page to **/var/www/html** directory which is **document root** of your PHP server and it will display all the file's detail upon completion. Please note that if you are going to display uploaded file then don't try with binary files like images or word document.

Here is the code of **uploader.php** script which will take care of uploading a file.

|  |
| --- |
| <?php  if( $\_FILES['file']['name'] != "" )  {  copy( $\_FILES['file']['name'], "/var/www/html" ) or  die( "Could not copy file!");  }  else  {  die("No file specified!");  }  ?>  <html>  <head>  <title>Uploading Complete</title>  </head>  <body>  <h2>Uploaded File Info:</h2>  <ul>  <li>Sent file: <?php echo $\_FILES['file']['name']; ?>  <li>File size: <?php echo $\_FILES['file']['size']; ?> bytes  <li>File type: <?php echo $\_FILES['file']['type']; ?>  </ul>  </body>  </html> |

When you will upload a file using upload form and upload script, it will display following result:

|  |
| --- |
| **Uploaded File Info:**   * Sent file: **uploadedfile.txt** * File size: 2003 bytes * File type: image/jpg |

**REFER TO LAB TASK THREE**

**7.6  Working with Session**

**Refer to Lab task 2**