**Working with form**

### Creating a Simple Input Form

Given example builds a simple HTML form.

##### **A Simple HTML Form**

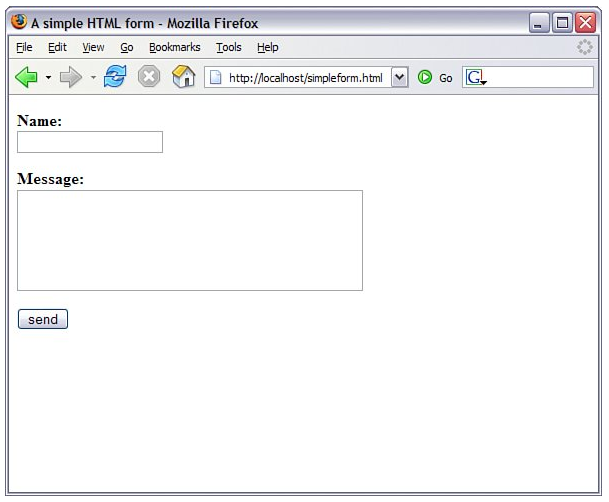
|  |
| --- |
| 1: <html>  2: <head>  3: <title>A simple HTML form</title>  4: </head>  5: <body>  6: <form action="send\_simpleform.php" method="POST">  7: <p><strong>Name:</strong><br/>  8. <input type="text" name="user"/></p>  9: <p><strong>Message:</strong><br/>  10: <textarea name="message" rows="5" cols="40"></textarea></p>  11: <p><input type="submit" value="send"/></p>  12: </form>  13: </body>  14: </html> |

Put these lines into a text file called simpleform.html and place that file in your web server document root. This listing defines a form that contains a text field with the name "user" on line 8, a text area with the name "message" on line 10, and a submit button on line 11. The FORM element's ACTION argument points to a file called send\_simpleform.php, which processes the form information. The method of this form is POST, so the variables are stored in the $\_POST superglobal.

Given example creates the code that receives our users' input.

##### **Reading Input from a Form**

|  |
| --- |
| 1: <?php  2: echo "<p>Welcome <b>".$\_POST["user"]."</b>!</p>";  3: echo "<p>Your message is:<br/><b>".$\_POST["message"]."</b></p>";  4: ?> |



Put these lines into a text file called send\_simpleform.php and place that file in your web server document root. Now access the form itself (simpleform.html) with your web browser, and you should see something like given figure

The script in send\_simpleform.php is called when the user submits the form created in simpleform.html. In the code in send\_simpleform.php, we access two variables: $\_POST["user"] and $\_POST["message"]. These are references to the variables in the $\_POST superglobal, which contain the values that the user entered in the user text field and the message text area. Forms in PHP really are as simple as that.

Enter some information in the form fields and click the send button. You should see your input echoed to the screen.

*By the Way*

*You could also use the GET method in this form (and others). POST can handle more data than GET and does not pass the data in the query string. If you use the GET method, be sure to change your superglobal to $\_GET and not $\_POST.*

### Accessing Form Input with User-Defined Arrays

The previous example showed how to gather information from HTML elements that submit a single value per element name, such as text fields, text areas, and radio buttons. This leaves us with a problem when working with elements like SELECT because it is possible for the user to choose one or more items from a multiple SELECT list. If we name the SELECT element with a plain name, like so

<select name="products" multiple>

the script that receives this data has access to only a single value corresponding to this name ($\_POST["products"]). We can change this behavior by renaming an element of this kind so that its name ends with an empty set of square brackets. We do this in given example.

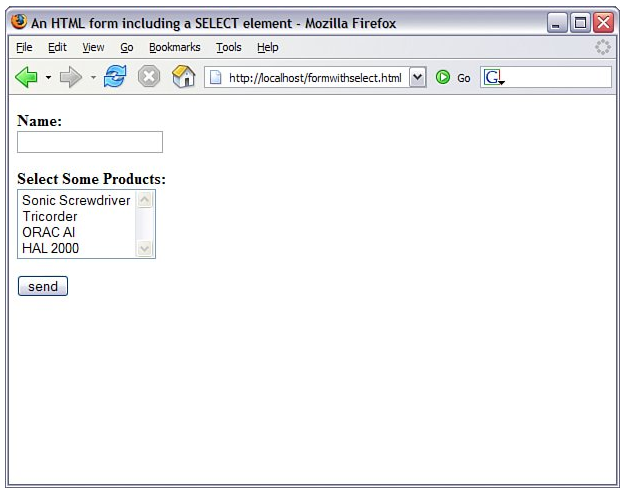
##### **An HTML Form Including a SELECT Element**

|  |
| --- |
| 1: <html>  2: <head>  3: <title>An HTML form including a SELECT element</title>  4: </head>  5: <body>  6: <form action="send\_formwithselect.php" method="POST">  7: <p><strong>Name:</strong><br/>  8: <input type="text" name="user"/>  9: <p><strong>Select Some Products:</strong><br/>  10: <select name="products[]" multiple="multiple">  11: <option value="Sonic Screwdriver">Sonic Screwdriver</option>  12: <option value="Tricoder">Tricorder</option>  13: <option value="ORAC AI">ORAC AI</option>  14: <option value="HAL 2000">HAL 2000</option>  15: </select>  16: <p><input type="submit" value="send"/></p>  17: </form>  18: </body>  19: </html> |

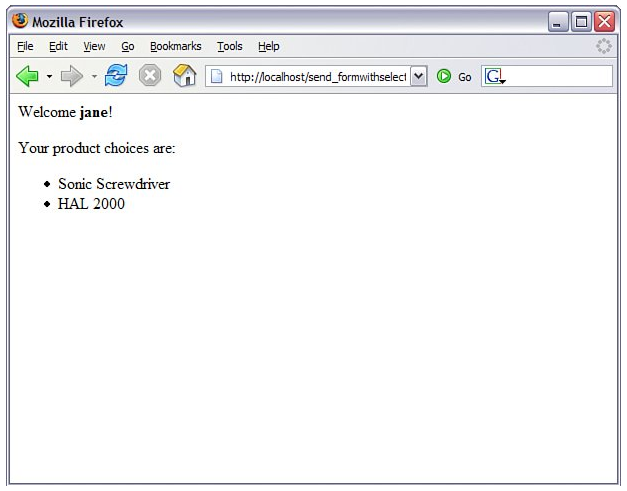
Put these lines into a text file called formwithselect.html and place that file in your web server document root. Next, in the script that processes the form input, we find that input from the "products[]" form element created on line 10 is available in an array called $\_POST["products"]. Because products[] is a SELECT element, we offer the user multiple choices using the option elements on lines 11 through 14. We demonstrate that the user's choices are made available in an array in given example.

##### **Reading Input from the Form in above example**

|  |
| --- |
| 1: <?php  2: echo "<p>Welcome <b>".$\_POST["user"]."</b>!</p>";  3: echo "<p>Your product choices are:<br/>";  4: if (!empty($\_POST["products"])) {  5: echo "<ul>";  6: foreach ($\_POST["products"] as $value) {  7: echo "<li>$value</li>";  8: }  9: echo "</ul>";  10: }  11: ?> |



Put these lines into a text file called send\_formwithselect.php and place that file in your web server document root. Now access the form with your web browser and fill out the fields. Given figure shows an example.

On line 2 of the script in send\_formwithselect.php, we access the $\_POST["user"] variable, which is derived from the user form element. On line 4, we test for the $\_POST["products"] variable. If $\_POST["products"] is present, we loop through it on line 6, and output each choice to the browser on line 7. The text within the value attribute of the selected OPTION element becomes one of the stored values in the array.

Submit the form, and you might see something like that shown in given figure.

Although the looping technique is particularly useful with the SELECT element, it can work with other types of form elements as well. For example, by giving a number of check boxes the same name, you can enable a user to choose many values within a single field name.

As long as the name you choose ends with empty square brackets, PHP compiles the user input for this field into an array. We can replace the SELECT elements from lines 1114 in formwithselect.html with a series of check boxes to achieve the same effect:

<input type="checkbox" name="products[]"

value="Sonic Screwdriver"/>Sonic Screwdriver<br/>

<input type="checkbox" name="products[]" value="Tricorder"/>Tricorder<br/>

<input type="checkbox" name="products[]" value="ORAC AI"/>ORAC AI<br/>

<input type="checkbox" name="products[]" value="HAL 2000"/>HAL 2000<br/>

The selected values will still be accessible using the $\_POST["products"] variable, just as if they came from the multiple SELECT list in send\_formwithselect.php.

### Combining HTML and PHP Code on a Single Page

In some circumstances, you might want to include the form-parsing PHP code on the same page as a hard-coded HTML form. Such a combination can be useful if you need to present the same form to the user more than once. You would have more flexibility if you were to write the entire page dynamically, of course, but you would miss out on one of the great strengths of PHP, which is that it mingles well with standard HTML. The more standard HTML you can include in your pages, the easier they are for designers and page builders to amend without asking you, the programmer, for help. For the following examples, imagine that we're creating a site that teaches basic math to preschool children and have been asked to create a script that takes a number from form input and tells the user whether it's larger or smaller than a predefined integer.

Given example creates the HTML. For this example, we need only a single text field, but even so, we'll include a little PHP.

##### **An HTML Form That Calls Itself**

|  |
| --- |
| 1: <html>  2: <head>  3: <title>An HTML form that calls itself</title>  4: </head>  5: <body>  6: <form action="<?php echo $\_SERVER["PHP\_SELF"]; ?>" method="POST">  7: <p><strong>Type your guess here:</strong>  8: <input type="text" name="guess"/></p>  9: <p><input type="submit" value="submit your guess"/></p>  10: </form>  11: </body>  12: </html> |

The action of this script is $\_SERVER["PHP\_SELF"], as seen in line 6. This global variable represents the name of the current script. In other words, the action tells the script to reload itself. The script in above example doesn't produce any output, but if you upload the script to your web server, access the page, and view the source of the page, you will notice that the form action now contains the name of the script itself.

In given example, we begin to build up the PHP element of the page.

##### **A PHP Number-Guessing Script**

|  |
| --- |
| 1: <?php  2: $num\_to\_guess = 42;  3: if (!isset($\_POST["guess"])) {  4: $message = "Welcome to the guessing machine!";  5: } else if ($\_POST["guess"] > $num\_to\_guess) {  6: $message = $\_POST["guess"]." is too big! Try a smaller number.";  7: } else if ($\_POST["guess"] < $num\_to\_guess) {  8: $message = $\_POST["guess"]." is too small! Try a larger number.";  9: } else { // must be equivalent  10: $message = "Well done!";  11: }  12: ?> |

First, we must define the number that the user guesses, and we do this in line 2 when we assign 42 to the $num\_to\_guess variable. Next, we must determine whether the form has been submitted; we can test for submission by looking for the existence of the variable $\_POST["guess"], which will be available only if your script has been submitted with a value in the guess field. If a value for $\_POST["guess"] isn't present, we can safely assume that the user arrived at the page without submitting a form. If the value is present, we can test the value it contains. The test for the presence of the $\_POST["guess"] variable takes place on line 3.

Lines 3 through 11 represent an if...else if...else control structure. Only one of these conditions will be true at any given time, depending on what (if anything) was submitted from the form. Depending on the condition, a different value will be assigned to the $message variable. That variable is then printed to the screen in line 18 of the script:

13: <html>

14: <head>

15: <title>A PHP number guessing script</title>

16: </head>

17: <body>

18: <h1><?php echo $message; ?></h1>

19: <form action="<?php echo $\_SERVER["PHP\_SELF"]; ?>" method="POST">

20: <p><strong>Type your guess here:</strong>

21: <input type="text" name="guess" /></p>

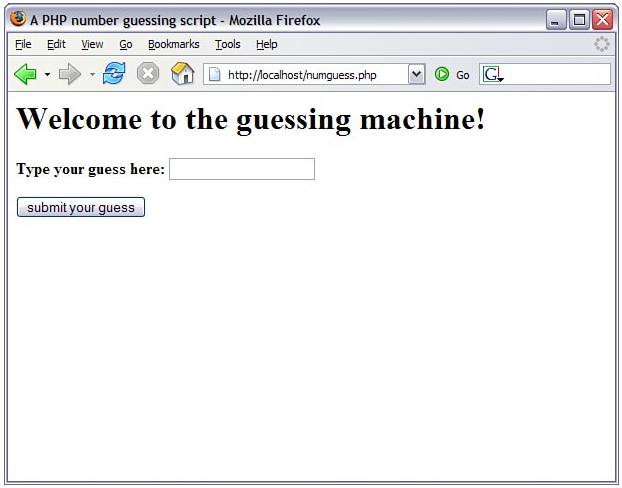
22: <p><input type="submit" value="submit your guess" /></p>

23: </form>

24: </body>

25: </html>

Put these lines into a text file called numguess.php and place this file in your web server document root. Now access the script with your web browser, and you should see something like given figure.



There are still a few more additions you could make, but you can probably see how simple it would be to hand the code to a designer for aesthetic treatment. The designer can do her part without having to disturb the programming in any way the PHP code is at the top, and the rest is 99% HTML.

### Using Hidden Fields to Save State

The script in numguess.php has no way of knowing how many guesses a user has made, but we can use a hidden field to keep track of this value. A hidden field behaves exactly the same as a text field, except that the user cannot see it unless he views the HTML source of the document that contains it.

Take the original numguess.php script and save a copy as numguess2.php. In the new version, add a line after the initial assignment of the $num\_to\_guess variable:

$num\_tries = (isset($\_POST["num\_tries"])) ? $num\_tries + 1 : 1;

This line initializes a variable called $num\_tries and assigns a value to it. If the form has not yet been submitted (if $\_POST["num\_tries"] is empty), the value of the $num\_tries variable is 1 because we will be on our first attempt at guessing the number. If the form has already been sent, the new value is the value of $\_POST["num\_tries"] plus 1.

The next change comes after the HTML level H1 heading:

<p><strong>Guess number:</strong> <?php echo $num\_tries; ?></p>

This new line simply prints the current value of $num\_tries to the screen.

Finally, before the HTML code for the form submission button, add the hidden field. This field saves the incremented value of $num\_tries:

<input type="hidden" name="num\_tries" value="<?php echo $num\_tries; ?>"/>

Given example shows the new script in its entirety.

##### **Saving State with a Hidden Field**

|  |
| --- |
| 1: <?php  2: $num\_to\_guess = 42;  3: $num\_tries = (isset($\_POST["num\_tries"])) ? $\_POST["num\_tries"] + 1 : 1;  4: if (!isset($\_POST["guess"])) {  5: $message = "Welcome to the guessing machine!";  6: } else if ($\_POST["guess"] > $num\_to\_guess) {  7: $message = $\_POST["guess"]." is too big! Try a smaller number.";  8: } else if ($\_POST["guess"] < $num\_to\_guess) {  9: $message = $\_POST["guess"]." is too small! Try a larger number.";  10: } else { // must be equivalent  11: $message = "Well done!";  12: }  13: $guess = $\_POST["guess"];  14: ?>  15: <html>  16: <head>  17: <title>Saving state with a hidden field</title>  18: </head>  19: <body>  20: <h1><?php echo $message; ?></h1>  21: <p><strong>Guess number:</strong> <?php echo $num\_tries; ?></p>  22: <form action="<?php echo $\_SERVER["PHP\_SELF"]; ?>" method="POST">  23: <p><strong>Type your guess here:</strong>  24: <input type="text" name="guess" value="<?php echo $guess; ?>" />  25: <input type="hidden" name="num\_tries" value="<?php echo $num\_tries; ?>" />  26: <p><input type="submit" value="submit your guess" /></p>  27: </form>  28: </body>  29: </html> |

Save the numguess2.php file and place it in your web server document root. Access the form a few times with your web browser and try to guess the number (pretend you don't already know it). The counter should increment by 1 each time you access the form.

|  |  |  |
| --- | --- | --- |
| Redirecting the User Our simple script still has one major drawback the form is reloaded whether or not the user guesses correctly. The fact that the HTML is hard-coded makes it difficult to avoid writing the entire page. We can, however, redirect the user to a congratulations page, thereby sidestepping the issue altogether.  When a server script communicates with a client, it must first send some headers that provide information about the document to follow. PHP usually handles this for you automatically, but you can choose to send your own header lines with PHP's header() function.  To call the header() function, you must be absolutely sure that no output has been sent to the browser. The first time content is sent to the browser, PHP sends out headers of its own, and it's too late for you to send any more. Any output from your document, even a line break or a space outside your script tags, causes headers to be sent. If you intend to use the header() function in a script, you must make certain that nothing precedes the PHP code that contains the function call. You should also check any libraries that you might be using.  Given example shows typical headers sent to the browser by PHP, beginning with line 3, in response to the request in line 1. **Typical Server Headers Sent from a PHP Script**  |  | | --- | | 1: HTTP/1.1 200 OK  2: Date: Tue, 28 Feb 2006 07:17:28 PST  3: Server: Apache/2.0.55 (Win32) PHP/5.1.2  4: X-Powered-By: PHP/5.1.2  5: Connection: close  6: Content-Type: text/html |     By sending a Location header instead of PHP's default header, you can cause the browser to be redirected to a new page, such as  header("Location: http://www.techcity.com.np");  Assuming that we've created a suitably upbeat page called congrats.html, we can amend our number-guessing script to redirect the user if she guesses correctly, as shown in given example. The only change between this and numguess2.php comes after the else clause on line 10. **Using header() to Redirect User**  |  | | --- | | 1: <?php  2: $num\_to\_guess = 42;  3: $num\_tries = (isset($\_POST["num\_tries"])) ? $\_POST["num\_tries"] + 1 : 1;  4: if (!isset($\_POST["guess"])) {  5: $message = "Welcome to the guessing machine!";  6: } elseif ($\_POST["guess"] > $num\_to\_guess) {  7: $message = $\_POST["guess"]." is too big! Try a smaller number";  8: } elseif ($\_POST["guess"] < $num\_to\_guess) {  9: $message = $\_POST["guess"]." is too small! Try a larger number";  10: } else { // must be equivalent  11: header("Location: congrats.html");  12: exit;  13: }  14: $guess = $\_POST["guess"];  15: ?>  16: <html>  17: <head>  18: <title>Using header() to Redirect User</title>  19: </head>  20: <body>  21: <h1><?php echo $message; ?></h1>  22: <p><strong>Guess number:</strong> <?php echo $num\_tries; ?></p>  23: <form action="<?php echo $\_SERVER["PHP\_SELF"]; ?>" method="POST">  24: <p><strong>Type your guess here:</strong>  25: <input type="text" name="guess" value="<?php echo $guess; ?>" />  26: <input type="hidden" name="num\_tries" value="<?php echo $num\_tries; ?>" />  27: <p><input type="submit" value="submit your guess" /></p>  28: </form>  28: </body>  30: </html> |   The else clause of our if statement on line 10 now causes the browser to send us away to a page called congrats.html. We ensure that all output from the current page is aborted with the exit statement on line 12, which immediately ends execution and output of this script. |

### Sending Mail on Form Submission

You've already seen how to take form responses and print the results to the screen, so you're only one step away from sending those responses in an email message.

Before learning about sending mail, however, read through the next section to make sure that your system is properly configured.

#### System Configuration for the mail() Function

Before you can use the mail() function to send mail, a few directives must be set up in the php.ini file so that the function works properly. Open php.ini with a text editor and look for these lines:

[mail function]

; For Win32 only.

SMTP = localhost

; For Win32 only.

sendmail\_from = me@localhost.com

; For Unix only. You may supply arguments as well (default: "sendmail -t -i").

;sendmail\_path =

If you're using Windows as your web server platform, the first two directives apply to you. For the mail() function to send mail, it must be able to access a valid outgoing mail server. If you plan to use the outgoing mail server of your ISP (in the following example, we use EarthLink), the entry in php.ini should look like this:

SMTP = mail.earthlink.net

The second configuration directive is sendmail\_from, which is the email address used in the From header of the outgoing email. It can be overwritten in the mail script itself but normally operates as the default value, as in this example:

sendmail\_from = youraddress@yourdomain.com

A good rule of thumb for Windows users is that whatever outgoing mail server you've set up in your email client on that machine, you should also use as the value of SMTP in php.ini.

If your web server is running on a Linux/UNIX platform, you use the sendmail functionality of that particular machine. In this case, only the last directive applies to you: sendmail\_path. The default is sendmail -t -i, but if sendmail is in an odd place or if you need to specify different arguments, feel free to do so, as in the following example, which does not use real values:

sendmail\_path = /opt/sendmail -odd arguments

After making any changes to php.ini on any platform, you must restart the web server process for the changes to take effect.

#### Creating the Form

In given example, you see the basic HTML for creating a simple feedback form, let's call it feedback.html. This form has an action of sendmail.php, which we will create in the next section. The fields in feedback.html are simple: Line 7 contains a name field, line 8 contains the return email address field, and line 10 contains the text area for the user's message.

##### **Creating a Simple Feedback Form**

|  |
| --- |
| 1: <html>  2: <head>  3: <title>E-Mail Form</title>  4: </head>  5: <body>  6: <form action="sendmail.php" method="POST">  7: <p><strong>Name:</strong><br> <input type="text" size="25" name="name" /></p>  8: <p><strong>E-Mail Address:</strong><br />  9: <input type="text" size="25" name="email" /></p>  10: <p><strong>Message:</strong><br />  11: <textarea name="message" cols="30" rows="5"></textarea></p>  12: <p><input type="submit" value="send" /></p>  13: </form>  14: </body>  15: </html> |

Put these lines into a text file called feedback.html and place this file in your web server document root.

In the next section, you create the script that sends this form to a recipient.

#### Creating the Script to Send the Mail

This script is only slightly different in concept from the script in send\_formwithselect.php, which simply printed form responses to the screen. In the script shown in given example, in addition to printing the responses to the screen, you send them to an email address.

##### **Sending the Simple Feedback Form**

|  |
| --- |
| 1: <html>  2: <head>  3: <title>Sending mail from the form in Listing 11.10</title>  4: </head>  5: <body>  6: <?php  7: echo "<p>Thank you, <b>".$\_POST["name"]."</b>, for your message!</p>";  8: echo "<p>Your e-mail address is: <b>".$\_POST["email"]."</b>.</p>";  9: echo "<p>Your message was:<br />";  10: echo $\_POST["message"]."</p>";  11: //start building the mail string  12: $msg = "Name: ".$\_POST["name"]."\n";  13: $msg .= "E-Mail: ".$\_POST["email"]."\n";  14: $msg .= "Message: ".$\_POST["message"]."\n";  15: //set up the mail  16: $recipient = "you@yourdomain.com";  17: $subject = "Form Submission Results";  18: $mailheaders = "From: My Web Site <defaultaddress@yourdomain.com> \n";  19: $mailheaders .= "Reply-To: ".$\_POST["email"];  20: //send the mail  21: mail($recipient, $subject, $msg, $mailheaders);  22: ?>  23: </body>  24: </html> |

The variables used in lines 79 are $\_POST["name"], $\_POST["email"], and $\_POST["message"]the names of the fields in the form, their values saved as part of the $\_POST super-global. That's all well and good for printing the information to the screen, but in this script, you also want to create a string that's sent in email. For this task, you essentially build the email by concatenating strings to form one long message string, using the newline (\n) character to add line breaks where appropriate.

Lines 12 through 14 create the $msg variable, a string containing the values typed by the user in the form fields (and some introductory text for good measure). This string will form the body of the email. Note the use of the concatenation operator (.=) when adding to the $msg variable in lines 13 and 14.

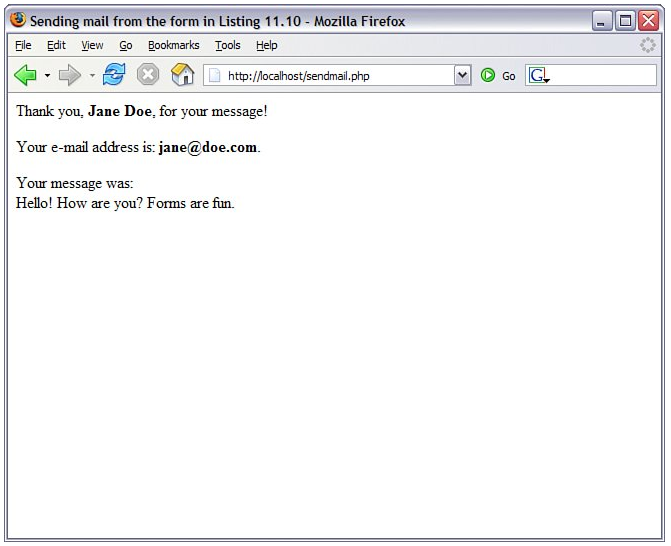
Lines 16 and 17 are hard-coded variables for the email recipient and the subject of the email message. Replace you@yourdomain.com with your own email address, obviously. If you want to change the subject, feel free to do that, too!

Lines 18 and 19 set up some mail headers, namely the From: and Reply-to: headers. You could put any value in the From: header; this is the information that displays in the From or Sender column of your email application when you receive this mail.

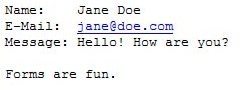
*By the Way*

*If your outbound mail server is a Windows machine, the \n newline character should be replaced with \r\n.*

*The mail() function requires four parameters: the recipient, the subject, the message, and any additional mail headers. The order of these parameters is shown in line 21, and your script is complete after you close up your PHP block and your HTML elements in lines 2224.*



Put these lines into a text file called sendmail.php and place that file in your web server document root. Use your web browser and go back to the form, enter some information, and click the submission button. You should see something like given figure in your browser.



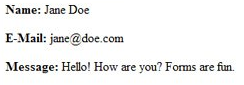
If you then check your email, you should have a message waiting for you. It might look something like given figure.

#### Formatting Your Mail with HTML

The "trick" to sending HTML-formatted email is not a trick at all. In fact, it only involves writing the actual HTML and modifying the headers sent by the mail() function. In given example, a variation of above example, changes were made in lines 12-14 and lines 18-19.

##### **Sending the Simple Feedback FormHTML Version**

|  |
| --- |
| 1: <html>  2: <head>  3: <title>Sending the Simple Feedback Form HTML Version</title>  4: </head>  5: <body>  6: <?php  7: echo "<p>Thank you, <b>".$\_POST["name"]."</b>, for your message!</p>";  8: echo "<p>Your e-mail address is: <b>".$\_POST["email"]."</b>.</p>";  9: echo "<p>Your message was:<br />";  10: echo $\_POST["message"]."</p>";  11: //start building the mail string  12: $msg = "<p><strong>Name:</strong> ".$\_POST["name"]."</p>";  13: $msg .= "<p><strong>E-Mail:</strong> ".$\_POST["email"]."</p>";  14: $msg .= "<p><strong>Message:</strong> ".$\_POST["message"]."</p>";  15: //set up the mail  16: $recipient = "you@yourdomain.com";  17: $subject = "Form Submission Results";  18: $mailheaders = "MIME-Version: 1.0\r\n";  19: $mailheaders .= "Content-type: text/html; charset=ISO-8859-1\r\n";  20: $mailheaders .= "From: My Web Site <defaultaddress@yourdomain.com> \n";  21: $mailheaders .= "Reply-To: ".$\_POST["email"];  22: //send the mail  23: mail($recipient, $subject, $msg, $mailheaders);  24: ?>  25: </body>  26: </html> |

In lines 12-14, the message string now contains HTML code. Additional headers are created in lines 18-19, which set the Mime Version header to 1.0 and the Content-type header to text/html with a character set of ISO-8859-1. When opened in an HTML-enabled mail client, the HTML in the message string will appear as intended, as shown in given figure.

### Working with File Uploads

So far, we've looked at simple form input. However, web browsers support file uploads, and so, of course, does PHP. In this section, you examine the features that PHP makes available to deal with this kind of input.

Information about the uploaded file becomes available to you in the $\_FILES super-global, which is indexed by the name of the upload field (or fields) in the form. The corresponding value for each of these keys is an associative array. These fields are described in given table, using fileupload as the name of the form field used for the upload.

| **File Upload Global Variables** | | |
| --- | --- | --- |
| **Element** | **Contains** | **Example** |
| $\_FILES["fileupload"]["name"] | Original name of uploaded file | test.gif |
| $\_FILES["fileupload"]["tmp\_name"] | Path to temporary file | /tmp/phprDfZvN |
| $\_FILES["fileupload"]["size"] | Size (in bytes) of uploaded file | 6835 |
| $\_FILES["fileupload"]["type"] | MIME type of uploaded file (where given by client) | image/gif |

Keep these elements in the back of your mind for a moment, while we create the upload form in the next section.

#### Creating the File Upload Form

First, we must create the HTML form to handle the upload. HTML forms that include file upload fields must include an ENCTYPE argument:

ENCTYPE="multipart/form-data"

PHP also works with an optional hidden field that can be inserted before the file upload field. This field must be called MAX\_FILE\_SIZE and should have a value representing the maximum size in bytes of the file that you're willing to accept. The MAX\_FILE\_SIZE field is obeyed at the browser's discretion, so you should rely on the php.ini setting, upload\_max\_filesize, to cap unreasonably large uploads. After the MAX\_FILE\_SIZE field has been entered, you're ready to add the upload field itself. This is simply an INPUT element with a TYPE argument of "file". You can give it any name you want. Given example brings all this together into an HTML upload form.

##### **A Simple File Upload Form**

|  |
| --- |
| 1: <html>  2: <head>  3: <title>A simple file upload form</title>  4: </head>  5: <body>  6: <form action="do\_upload.php" enctype="multipart/form-data" method="POST">  7: <input type="hidden" name="MAX\_FILE\_SIZE" value="51200" />  8: <p><strong>File to Upload:</strong>  9: <input type="file" name="fileupload" /></p>  10: <p><input type="submit" value="upload!" /></p>  11: </form>  12: </body>  13: </html> |

8.PNGAs you can see, file uploads are limited to 50KB on line 7, and the name of the file upload field is fileupload, as shown on line 8. Save this listing in a text file called fileupload.html and place that file in your web server document root. Use your web browser to access this form and you should see something like given figure.

This form calls the do\_upload.php script, which we will create next.

#### Creating the File Upload Script

If you remember the information regarding the $\_FILES superglobal, you have all the information you need to write a simple file upload script. This script is the back-end for the form created in given example.

##### **A File Upload Script**

|  |
| --- |
| 1: <?php  2: $file\_dir = "/path/to/upload/directory";  3: foreach($\_FILES as $file\_name => $file\_array) {  4: echo "path: ".$file\_array['"tmp\_'name"]."<br />\n";  5: echo "name: ".$file\_array[''"name"]."<br />\n";  6: echo "type: ".$file\_array["'type'"]."<br />\n";  7: echo "size: ".$file\_array[''"size"]."<br />\n";  8:  9: if (is\_uploaded\_file($file\_array['"tmp\_'name"])) {  10: move\_uploaded\_file($file\_array['"tmp\_'name"],  11: "$file\_dir/".$file\_array["name"]") or die ("Couldn't copy");  12: echo "file was moved!<br />";  13: }  14: }  15: ?> |

In above example, we first create the $file\_dir variable on line 2 to store path information. This path should be one that exists on your system, and the web server user (for example, httpd, www, nobody) must have write permissions for it.

*By the Way*

*The path used in line 2 is a Linux/UNIX path. Windows users would use escaped backslashes, such as:*

*$file\_dir = "C:\\Documents and Settings\\Owner\\Desktop\\";*

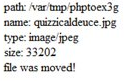
Line 3 begins a foreach statement that loops through every element in the $\_FILES array. A loop is used rather than an if statement to make our script capable of scaling to deal with multiple uploads on the same page. The foreach loop on line 3 stores the upload file's name in the $file\_name variable and the file information in the $file\_array variable. We can then output the information we have about the upload.

Before moving the uploaded file from its temporary position to the location specified in line 2, first check that it exists. We do so on line 9, using the is\_uploaded\_file() function. This function accepts a path to an uploaded file and returns true only if the file in question is a valid upload file. This function therefore enhances the security of your scripts.

Assuming that all is well, the file is copied from its temporary home to a new directory on lines 10 and 11. We use another function, move\_uploaded\_file(), for this purpose. This function copies a file from one place to another, first performing the same security checks as those performed by is\_uploaded\_file(). The move\_uploaded\_file() function requires a path to the source file and a path to the destination. It returns true if the move is successful and false if the file isn't a valid upload file or if the file couldn't be found.

*Watch Out!*

*Beware of the names of uploaded files. Operating systems such as Mac OS and Windows are pretty relaxed when it comes to file naming, so expect uploaded files to come complete with spaces, quotation marks, and all manner of other unexpected characters. Therefore, it's a good idea to filter filenames.*

Put these lines into a text file called do\_upload.php and place that file in your web server document root. Use your web browser to go back to the form and then try to upload a file. If successful, you should see something like given figure in your browser.