

Handling User Input in PHP

Input to PHP scripts is typically passed to the program using web forms. Forms include text fields, radio buttons, check boxes, popup boxes, scroll tables, and the like. Thus retrieving input from the user is typically a two-step process: you must create an HTML document that provides forms to allow users to pass information to the server, and your PHP code must have a means for parsing the input data and determining the action to take. We will briefly discuss the two topics in the following sections.

HTML Forms

Forms are designated within an HTML document by the fill-out form tag:

```
<FORM METHOD = "POST" ACTION = "http://form.url.com/servlet/serletName">
... Contents of the form ...
</FORM>
```

The URL given after ACTION is the URL of your program. The METHOD is the means of transferring data from the form to the program. In this example, we have used the "POST" method, which is the recommended method. There is another method called "GET", but there are common problems associated with this method. Both will be discussed in the next section.

Within the form you may have anything except another form. The tags used to create user interface objects are INPUT, SELECT, and TEXTAREA.

The INPUT tag specifies a simple input interface:

```
<INPUT TYPE="text" NAME="thisinput" VALUE="default" SIZE=10 MAXLENGTH=20>
<INPUT TYPE="checkbox" NAME="thisbox" VALUE="on" CHECKED>
<INPUT TYPE="radio" NAME="radio1" VALUE="1">
<INPUT TYPE="submit" VALUE="done">
<INPUT TYPE="radio" NAME="radio1" VALUE="2" CHECKED>
<INPUT TYPE="hidden" NAME="notvisible" VALUE="5">
```

Which would produce the following:

☒ ☐ ☒

The different attributes are mostly self-explanatory. The TYPE is the variety of input object that you are presenting. Valid types include "text", "password", "checkbox", "radio", "submit", "reset", and "hidden". Every input but "submit" and "reset" has a NAME which will be associated with the value returned in the input to the program. This will not be visible to the user (unless they read the HTML source). The other fields will be explained with the types:

"text" - refers to a simple text entry field. The VALUE refers to the default text within the text field, the SIZE represents the visual length of the field, and the MAXLENGTH indicates the maximum number of characters the textfield will allow. There are defaults to all of these (nothing, 20, unlimited).

"password" - the same as a normal text entry field, but characters entered are obscured.

"checkbox" - refers to a toggle button that is independently either on or off. The VALUE refers to the string sent to the server when the button is checked (unchecked boxes are disregarded). The default value is "on".

"radio" - refers to a toggle button that may be grouped with other toggle buttons such that only one in the group can be on. It's essentially the same as the checkbox, but any radio button with the same NAME attribute will be grouped with this one.

"submit" and "reset" - these are the pushbuttons on the bottom of most forms you'll see that submit the form or clear it. These are not required to have a NAME, and the VALUE refers to the label on the button. The default names are "Submit Query" and "Reset" respectively.

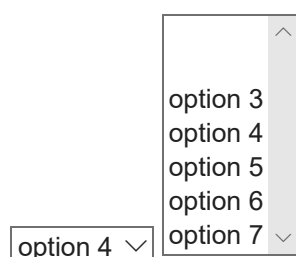
"hidden" - this input is invisible as far as the user interface is concerned (though don't be fooled into thinking this is some kind of security feature -- it's easy to find "hidden" fields by perusing a document source or examining the URL for a GET method). It simply creates an attribute/value binding without need for user action that gets passed transparently along when the form is submitted.

The second type of interface is the SELECT interface, which includes popup menus and scrolling tables. Here are examples of both:

```
<SELECT NAME="menu">
<OPTION>option 1
<OPTION>option 2
<OPTION>option 3
<OPTION SELECTED>option 4
<OPTION>option 5
<OPTION>option 6
<OPTION>option 7
</SELECT>
```

```
<SELECT NAME="scroller" MULTIPLE SIZE=7>
<OPTION SELECTED>option 1
<OPTION SELECTED>option 2
<OPTION>option 3
<OPTION>option 4
<OPTION>option 5
<OPTION>option 6
<OPTION>option 7
</SELECT>
```

Which will give us:



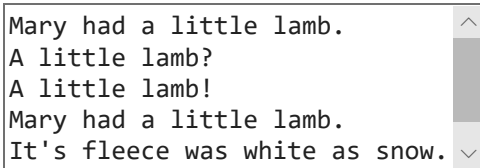
The image shows a web form with two input elements. On the left is a dropdown menu (popup menu) with a small downward arrow on the right. It currently displays "option 4". To its right is a vertical scrolling list (scroller) with a small upward arrow on the right. It contains a list of options: "option 3", "option 4", "option 5", "option 6", and "option 7". The "option 4" in the scroller is highlighted, indicating it is the currently selected option.

The SIZE attribute determines whether it is a menu or a scrolled list. If it is 1 or it is absent, the default is a popup menu. If it is greater than 1, then you will see a scrolled list with SIZE elements. The MULTIPLE option, which forces the select to be a scrolled list, signifies that a more than one value may be selected (by default only one value can be selected in a scrolled list).

OPTION is more or less self-explanatory -- it gives the names and values of each field in the menu or scrolled table, and you can specify which are SELECTED by default.

The final type of interface is the TEXTAREA interface:

```
<TEXTAREA NAME="area" ROWS=5 COLS=30>
Mary had a little lamb.
A little lamb?
A little lamb!
Mary had a little lamb.
It's fleece was white as snow.
</TEXTAREA>
```



As usual, the NAME is the symbolic reference to which the input will be bound when submitted to the program. The ROWS and COLS values are the visible size of the field. Any number of characters can be entered into a text area.

The default text of the text area is entered between the tags. Whitespace is supposedly respected (as between <PRE> HTML tags), including the newline after the first tag and before the last tag.

Server-Side Input Handling -- PHP

PHP is capable of handling both GET and POST requests. The parsing of the input is done for you by PHP, so you are separated from the actual format of the input data completely.

PHP scripts are typically a combination of HTML and PHP code, enclosed within <?php ?> tags, and are executed in response to HTTP requests from a client. Let's take a look at a very simple PHP program, the traditional HelloWorld:

```
<?php

print "<html>";
print "<head>";
$title = "Hello World";
print "<title>$title</title>";
print "</head>";
print "<body bgcolor=white>";
print "<h1>$title</h1>";

$params = $_GET["param"];
if($params) {
    print "Thanks for the lovely param='$params' binding.";
}

print "</body>";
print "</html>";

?>
```

We'll discuss points in this code again in the section on PHP output, but for now, we will focus on the input side. The variable \$_GET is an associative array containing the parameters from a client request. The values of the parameters passed from the HTML FORM can be retrieved by

accessing the array element corresponding to the parameter name. It is through this mechanisms that you can retrieve any of the values entered or implicit in the form. As might be inferred from the example above, PHP returns null if the parameter for whose name you request does not have a value. Unchecked buttons' bindings are not passed in a GET or POST message. You can check for null to determine when buttons are off. PHP also contains functions to check for array keys, such as [isset](#).

The variable `$_POST` contains parameters from a POST message. `$_REQUEST` contains all of the contents from the `$_GET`, `$_POST`, and `$_COOKIE` arrays. Be sure to check the PHP documentation for caveats and limitations when using these three arguments. For example, `$_GET` values may be at most 100 characters long.

Returning Output to the User

In your project, you are going to be concerned with returning HTML documents to the user. The documents will be dynamically created based on the output of the query. You can format it however you like, using ordinary HTML formatting routines and tags. A nice feature of PHP is the ability to include both standard HTML content and PHP code intermixed. This allows us to separate out the static presentation content from the content dynamically generated by your PHP code. For example, we could rewrite the HelloWorld example above like the following:

```
<html>
<head>
<?php $title = "Hello World" ?>
<title><?php print "$title"; ?></title>
</head>

<body bgcolor=white>
<h1><?php print "$title"; ?></h1>

<?php
$param = $_GET["param"];
if($param) {
    print "Thanks for the lovely param='$param' binding.";
}
?>

</body>
</html>
```

Notice how we the variable `$title` is still available in later `<?php ?>` code blocks.

You can provide provide separate HTML forms and PHP code for processing. Alternatively (or additionally), you can integrate FORMs into your PHP code by creating a FORM on the fly in your program, which will be invoked when a GET or POST is invoked by the client. For example, you may wish to include an input box on the output page so the user may submit another request, similar to how web search engines include the query box on the results page.

Handling Special Characters

Special characters like `&`, `<`, and `>`, need to be escaped as `&`, `<`, and `>`, respectively in HTML text (see [W3School's HTML Tutorial](#) for more). Moreover, special characters appearing in URL's need to be escaped, differently than when they appear in HTML text. For example, if you link on text with special characters and want to embed them into extended URLs as parameter values, you need to escape them: convert space to `+` or `%20`, convert `&` to `%26`, convert `=` to `%3D`,

convert % to %25, etc. (In general, any special character can be escaped by a percent sign followed by the character's hexadecimal ASCII value.) Important: Do NOT escape the & that actually separates parameters! Be careful not to confuse the escape strings for HTML text with those for URL's.

In PHP, you do not (and should not) do these conversions by hand. PHP contains many built in functions for HTML and URL parsing, encoding, and decoding. Browse the PHP documentation for more information.