# Programming via PHP More about forms

## 10.1. Transmission methods

HTTP, the transmission protocol used for the Web, actually defines two methods for sending form information to a Web server, called POST and GET. In Web forms, we've consistently used the POST method by including `method="post"` in the `form` tag. This is the recommended way for transmitting form data.

The distinction between the two is this: With the GET transmission method, the information from the form is encoded into the URL that is sent to the Web server. This technique has the shortcoming that the URL will become unmanageably long for moderately long forms. The HTTP designers thus quickly found that a better design would be to have Web browsers send the form data after sending the URL to the Web server.

The GET method, though, is still sometimes useful, particularly when you want to be able to access a PHP script through the use of a Web link rather than through completing a form. It's easiest to describe this through an example based on our forum site: When we list the name of each post's user, we might want to allow the user to click on the name to view the full information about that person. To do this, we'd want to generate the following HTML.

```
<p>By: <a href="user.php?userid=sherlock">Sherlock Holmes</a></p>
```

The GET method uses a question mark to separate the name of the script from the form data; following the question mark are field names and their associated values, separated by an equals sign. If there are multiple fields, each field/value pair is separated by an ampersand ('`&`').

Thus, when the user clicks on the link, the browser will send the URL to the Web server, which will invoke the `user.php` script as if the user had filled out a form typing`sherlock` into the `userid` field. The `user.php` script that we saw earlier will work as it has already been written.

(In fact, the values sent via the GET and POST methods are accessible separately by PHP. But in invoking `import_request_variables`, we've consistently passed `"pg"` as the first parameter. The *p* says to import the POST values, and the *g* says to import the GET values. So for our purposes, both sets of values are imported together.)

## 10.2. Input elements

We've already seen the *input* element, and in particular we've used the *text* and *submit* values for its *type* attribute, in order to create text fields and buttons. But we often want to create other types of input elements. In this section we'll look at some of the other possible values for *type*.

### 10.2.1. Hidden fields

One important value for *type* is *hidden*. Given this, a Web browser will not display anything about that input element on the screen. This may sound useless at first, since the user won't be able to interact with it, but it is actually one of the more useful options. It is used in situations where the HTML composer wants to send information into a script without showing it to the user.

The *hidden* type is really only useful when it is associated with a *value* attribute in addition to the *name* attribute. Whatever is in this *value* attribute will be sent to the PHP script without giving the user a chance to see or modify it.

Of course, the *hidden* type shouldn't be used when you have information that the user shouldn't ever see. While the element won't be rendered inside an HTML browser, the user who chooses to read the raw HTML source code will be able to see everything that appears.

### 10.2.2. Password fields

Another useful type is *password*. This works exactly like a text field, but it directly the Web browser to not display whatever the user types in the text field. Most Web browsers choose to display asterisks ('*') for each character the user types.

The *password* field doesn't provide any security beyond preventing getting the information by somebody looking over the user's shoulder. Of course, somebody looking over the user's keyboard might be able to figure out the information by watching the user's fingers.

### 10.2.3. Checkboxes

A checkbox is a box that can be checked or unchecked to represent a yes/no value. The checkbox's value is sent to the PHP script only when it is checked. If the checkbox is not checked, then the browser won't send any information about it, and so the `import_request_variables` function won't create a variable.

This brings up a question: How in a PHP script can we identify whether a variable exists or not? The answer lies in the **`isset`** function, which returns a Boolean value. Thus, you can write **`if`**`(`**`isset`**`($form_box))`... to test whether the checkbox named *`box`* was checked.

An *`input`* element representing a checkbox should always contain a *`value`* attribute so that the Web browser will know what value to send when the checkbox is checked. This *`value`* attribute does not affect whether the checkbox is checked: That is done by setting the *`checked`* attribute to *`checked`*; checkboxes ar unchecked when no *`checked`* attribute is mentioned.

Another issue that comes with checkboxes is that people normally expect that you can click anywhere on the label associated with the checkbox to toggle its value. However, the HTML we have seen thus far provides no way of indicating what the label is, so the Web browser cannot provide this user interface.

HTML provides a way of indicating the label through its *`label`* element. To use it, you enclose the *`input`* element along with its associated label within the same tag. (For cases where this is impossible because the label appears separately from the *`input`* element, you can use the *`for`* attribute whose value matches the *`name`* attribute of the matching *`input`* element.)

```
<label><input type="checkbox" name="box" value="on" checked="checked" />
  Do you understand checkboxes?</label>
```

Not all Web browsers honor the *`label`* tag, but it doesn't hurt to include it for those Web browsers that do honor it.

☑ Do you understand checkboxes?

Incidentally, you can use the *`label`* tag for other elements, such as those with a *`type`* of *`text`* or *`password`*, but the case for doing this is less compelling than for checkboxes or radio buttons.

Not all Web browsers honor the *`label`* tag, but it doesn't hurt to include it for those Web browsers that do honor it.

## 10.2.4. Radio buttons

Radio buttons present several options to the user, from which the user may select one. Most interfaces display a circle next to each option, filling in the circle that is currently selected.

Creating a set of radio buttons in HTML is a matter of creating several *input* elements all with the *type* attribute of *radio* and the identical *name* attribute.

```
<label><input type="radio" name="station" value="fm899" />
  FM 89.9 KQED</label>
<br /><label><input type="radio" name="station" value="fm917" checked="checke
d" />
  FM 91.7 KUAR</label>
```

○ FM 89.9 KQED
● FM 91.7 KUAR

## 10.3. Other form elements

### 10.3.1. Text areas

HTML specifies some form elements that are created through tags other than the *input* tag. One in the text area, typically a large text field with many rows: The way to create one of these is through a *textarea* element.

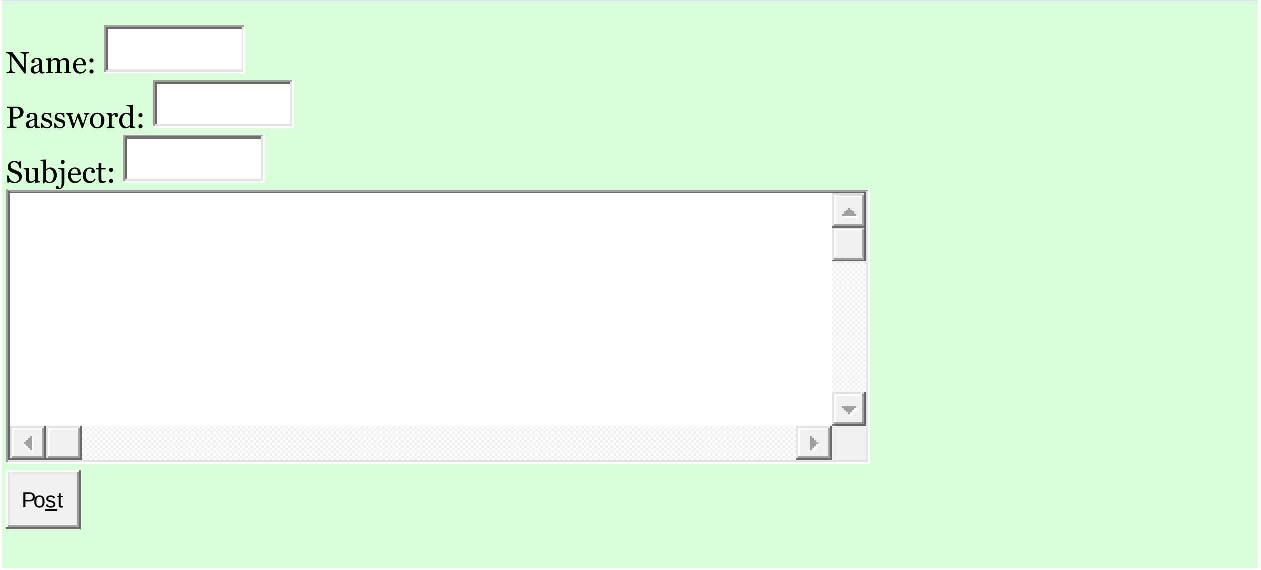
The *textarea* element has three official attributes that are typically used. The *name* attribute works just as for the *input* element: It identifies the name associated with the text area's contents when the form is submitted to the Web server. There are also the *rows* and the *cols* attributes, which provide guidance to the Web browser about how tall and how wide to make the text area; but the browser should provide scrollbars when the text becomes taller or wider than the area that is displayed on the screen.

Unlike the *input* element, which should not contain any text within it, the *textarea* element can. Whatever appears within the element should appear as the initial value within the textarea (as you would do using the *value* attribute for an *input* element corresponding to a text field).

In fact, we should have used the *textarea* element in the Web form for allowing a user to post information to our forum Web site. The following allows users to use multiple words.

```
<form method="post" action="post.php">
<p>Name: <input type="text" name="user" />
<br />Password: <input type="password" name="passwd" />
```

```
<br />Subject: <input type="text" name="subj" />
<br /><textarea name="body" rows="8" cols="80">Type your post here.</textarea
>
<br /><input type="submit" value="Post" />
</p></form>
```



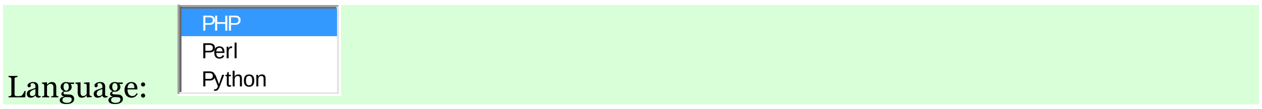
## 10.3.2. Lists

A final thing you might want is to present a list of options from which the user might choose. This can be done using the `select` element, within which is nested an`option` elements for each option to appear. The `select` element has a `name` attribute giving the name for the browser to use when identifying the selection when the form is submitted. The `option` element has a `value` attribute for specifying what value to send when that option is selected; and it has a `selected` attribute which allows the HTML to specify which option is initially selected.

```
Language: <select name="language">
<option value="php" selected="selected">PHP</option>
<option value="perl">Perl</option>
<option value="python">Python</option>
</select>
```

HTML does not specify whether the browser should do this using a drop-down menu or a list, but most use drop-down menus.

There are two options available for configuring the `select` element's appearance: The `size` attribute configures how many rows to use if it is displayed as a list box, and the `multiple` attribute, when set to `multiple`, tells the browser to allow the user to select multiple options. Most browsers wil display the `select` element as a list box when either is indicated. The below `select` element is defined identically to the above example, except that `size="3"` has been added into the `select` tag.



**Source: http://www.toves.org/books/php/ch10-moreforms/index.html**