XHTML Forms

Web forms, much like the analogous paper forms, allow the user to provide input. This input is typically sent to a server for processing. Forms can be used to submit data (e.g., placing an order for a product) or retrieve data (e.g., using a search engine). There are two parts to a form: the user interface, and a script to process the input and ultimately do something meaningful with it. This document explains how to create the form's user interface. The mechanics for processing form data – specifically with PHP – will be covered later in the course.

Form syntax

A form has three main components: the `form` element, form widgets (e.g., text boxes and menus), and a submit button. The `form` element has two required attributes – `method` and `action`.

```
<form method="post" action="script.url">
  ...
</form>
```

All form widgets have a `name` attribute that uniquely identifies them. The data is sent to the server as name-value pairs, where the name is the input element's name and value is that input element's value. We will discuss later how this value can be used by the server-side script.

Submission method

The `method` attribute specifies how data should be sent to the server. The `get` method encodes the form data into the URL. Suppose a form has two input elements with the names `name1` and `name2` with respective values `value1` and `value2`. The `get` method would send a URL of the form `script.url?name1=value1&name2=value2`. The `post` method will not include the form data encoded into the URL. The `post` method is considered more secure because the `get` method allows a hacker to bypass the form by calling the script directly and passing an arbitrarily-encoded URL string, which could contain invalid names and values. *NOTE*: If the form values contain non-ASCII characters or the form content of the URL exceeds 100 characters, the `post` method *must* be used.

Submission action

The `action` attribute specifies the script that will process the data. The value for this attribute is typically a CGI script or some other type of Web-based script, such as a PHP script. The `get` method is typically used when the script does not modify any state information on the server side, such as when a database query is performed. The `post` method is typically used if the script does modify any state information, such as making an update to a database.

Selection widgets

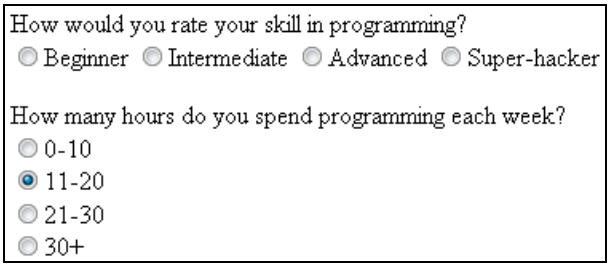
Selection widgets allow a user to select one or more items from a constrained set of choices. Selection widgets should always be preferred over text input widgets when the number of constrained choices is a manageable number, because they prevent erroneous input. If the number of constrained choices is small enough so that they can all be visually displayed, (e.g., a person's gender), radio buttons or check boxes should be used. If the number of constrained choices is large enough that it is infeasible to display them all (e.g., the states in the United States), a menu is a good choice.

Radio buttons

A radio button is a form widget that allows the user to choose *only one* of a predefined set of items. When the user selects a radio button, any previously selected radio button in the same group is deselected. To create a radio button, use the `input` element with `radio` as the `type` attribute's value, specify a name using the `name` attribute, and provide a value using the `value` attribute. All radio buttons that have the same value for the `name` attribute are considered a *group*. The value provided to the `name` attribute will be the name used in the name-value pair that gets passed to the server-side script. To make one of the radio buttons be the default selection, set the `checked` attribute for that radio button to the value "`checked`".

```
<p>How would you rate your skill in programming?<br />
  <input type="radio" name="skill" value="beg" />Beginner
  <input type="radio" name="skill" value="int" />Intermediate
  <input type="radio" name="skill" value="adv" />Advanced
  <input type="radio" name="skill" value="sup" />Super-hacker</p>
<p>How many hours do you spend programming each week?<br />
  <input type="radio" name="hours" value="beg" />0-10<br />
  <input type="radio" name="hours" value="int" checked="checked" />11-20<br />
  <input type="radio" name="hours" value="adv" />21-30<br />
  <input type="radio" name="hours" value="sup" />30+</p>
```

This is how the markup may appear in the browser.



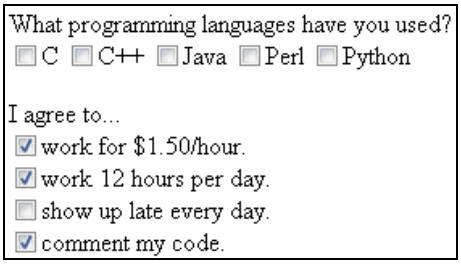
Note that the labels displayed next to the radio buttons are not the values of the `value` attributes. The `value` attribute's content is what will be sent to the server; the label/description is provided within the page text. Also note that the first group of radio buttons does not have a radio button selected. To ensure that a selection is made, use the `checked` attribute as shown in the example. If no radio button in a group is selected, that group's name-value pair will *not* be sent to the server, meaning that the input element will be undefined within the script that processes the form data.

Checkboxes

A checkbox is a form widget that allows the user to make *multiple* selections from a number of items. To create a checkbox, use the `input` element, specify `checkbox` as the `type`, specify a name using the `name` attribute, and provide a value using the `value` attribute. As with radio buttons, all checkboxes that have the same value for the `name` attribute are considered a *group*. You may pre-select one or more checkboxes by setting the `checked` attribute for that checkbox to the value "`checked`".

```
<p>What programming languages have you used?<br />
  <input type="checkbox" name="proglang" value="c" />C
  <input type="checkbox" name="proglang" value="cplusplus" />C++
  <input type="checkbox" name="proglang" value="java" />Java
  <input type="checkbox" name="proglang" value="perl" />Perl
  <input type="checkbox" name="proglang" value="perl" />Python</p>
<p>I agree to...<br />
  <input type="checkbox" name="cheaplabor" value="yes" checked="checked" />work for $1.50/hour.<br />
  <input type="checkbox" name="longdays" value="yes" checked="checked" />work 12 hours per day.<br />
  <input type="checkbox" name="late" value="yes" />show up late every day.<br />
  <input type="checkbox" name="usecomments" value="yes" checked="checked" />comment my code.<br /></p>
```

Here's how the markup may appear in the browser.



Suppose the checkboxes corresponding to "C", "Java", and "Perl" were selected in the above example. Three name-value pairs would be sent to the server: `proglang=c`, `proglang=java`, and `proglang=perl`.

Menus

A menu is a form widget that allows the user to select one (or possibly multiple) of several predefined values. Menus are useful in situations where there is not enough screen "real estate" to display all the values or where the readability of the page could be impaired by displaying too many values. Suppose you want the user to select their state of residence. One method is to make numerous radio buttons; the alternative is to use a menu.

The example output on the left demonstrates how the menu hides all but the first choice, whereas all of the choices are displayed when using radio buttons. The example output on the left also shows how the menu allows a user to immediately see the currently selected value, whereas radio buttons would force a user to scan potentially all of the radio buttons to determine the currently selected value. One might argue that the menu therefore improves page readability as well as conserving screen real estate. The example output on the right shows the expansion of the menu widget and the addition of scroll bars to further minimize the real estate occupied by the menu items.

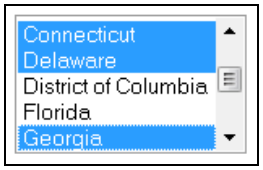
To create a menu, use the `select` element to specify a name using the `name` attribute. The optional attribute `size` specifies the desired height, in lines, of the menu and the optional attribute `multiple` allows the user to select more than one menu option (usually with the Control key for PCs and Command key for Macintoshes). The menu items themselves are created using the `option` element, which has a required `value` attribute. The text for the menu item is specified between the begin- and end-tags for the `option` element. To have a menu item selected by default, use the `selected` attribute.

```
<select name="state">
  <option value="al">Alabama</option>
  <option value="ak">Alaska</option>
  <option value="as">American Samoa</option>
  <option value="az">Arizona</option>
  <option value="ar">Arkansas</option>
  <option value="ca">California</option>
  ...
</select>
```

The following markup demonstrates the `select` element's `size` and `multiple` attributes, as well as the `selected` attribute for the `option` element.

```
<select name="state" size="5" multiple="multiple">
  <option value="al">Alabama</option>
  <option value="ak">Alaska</option>
  <option value="as">American Samoa</option>
  <option value="az">Arizona</option>
  <option value="ar">Arkansas</option>
  <option value="ca" selected="selected">California</option>
  ...
</select>
```

Here is how the markup may appear in the browser.



The example output on the left shows a menu with five options displayed simultaneously, with California already selected as the default choice. The example output on the right shows a menu with multiple (non-contiguous) choices selected.

Most Web browsers support grouping menu items into categories. To create a group menu, use the `optgroup` element, specifying the name of the category with the label attribute. Next, make the desired menu choices (`option` elements) children of the `optgroup` element.

```
<select name="state" size="10">
  <optgroup label="Pacific">
     <option value="ak">Alaska</option>
     <option value="ca">California</option>
     <option value="hi">Hawaii</option>
     <option value="or">Oregon</option>
     <option value="wa">Washington</option>
  </optgroup>
  <optgroup label="West">
     <option value="az">Arizona</option>
     <option value="co">Colorado</option>
  …
</select>
```

Here is how the output may appear in the browser.

Text input widgets

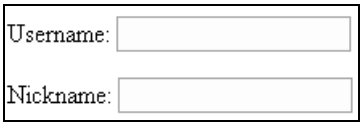
Text input widgets are used to input one or more strings of text. They can have either a specialized purpose, such as submitting a password, or a generic purpose, such as entering a free-form line of text.

Text boxes

Text boxes contain one line of free-form text, and are typically used when the input is not derived from a constrained set of values. For example, text boxes are good choices for names or addresses, because it is impossible to predict in advance the name or address that could be entered by a user. To create a text box, use the `input` element, specify `text` as the `type` attribute's value, and name the text box using the `name` attribute.

```
<p>Username: <input type="text" name="username" /></p>
<p>Nickname: <input type="text" name="nickname" /></p>
```

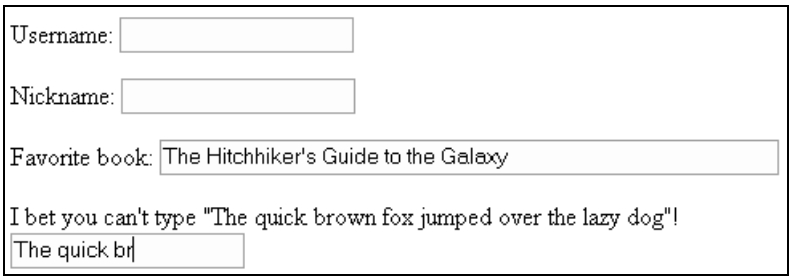
This is how the markup may appear in the browser.



Three optional elements control how the text box appears. The `value` attribute specifies the default text that should be shown when the text box is displayed. This text, unless modified by the user, will be sent to the server. The `size` attribute specifies the width of the text box measured in characters. The `maxlength` attribute specifies the maximum number of characters that can be entered in the text box.

```
<p>Username: <input type="text" name="username" /></p>
<p>Nickname: <input type="text" name="nickname" /></p>
<p>Favorite book: <input type="text" name="favbook" size="60"
  value="The Hitchhiker's Guide to the Galaxy" /></p>
<p>I bet you can't type &quot;The quick brown fox jumped over the lazy dog&quot;!<br />
<input type="text" name="pangram" maxlength="12" /></p>
```

This is how the markup may appear in the browser. *NOTE*: The text in the last text box was entered to demonstrate that no more than twelve characters are permitted.



A few things to note about text boxes:

- The default width is twenty characters (e.g., `size="20"`).
- The default value for a text box is the empty string (e.g., `value=""`). Unlike a radio button group – which if no radio button is selected, sends nothing to the server – the text box will send the empty string to the server.

Password boxes

Password boxes can be used to visually mask confidential data, such as passwords, from casual observers. Password boxes are identical to text boxes, except that whatever is typed into a password box is masked by bullets or asterisks. To create a password box, use the `input` element, specify `password` as the `type` attribute's value, and name the password box using the `name` attribute.

```
<p>Username: <input type="text" name="uname" /></p>
<p>Password: <input type="password" name="passwd" /></p>
```

Here is how the markup may be displayed in the browser.

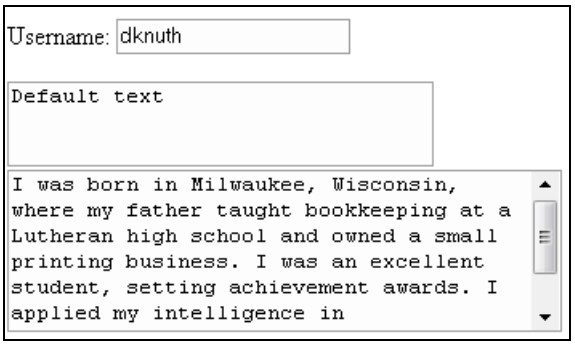
As with text boxes, password boxes also support the `size`, `maxlength`, and `value` attributes; however, having a predefined value for a password box defeats the purpose of having a password at all. It is also important to note that password boxes do not provide any encryption, so it is possible for the password to be intercepted when the form data is sent to the server unless a secure connection is used (i.e., HTTPS). The primary purpose of the password box is to *visually* protect the text string.

Text areas

Text areas allow the user to provide more than one line of textual input. Text areas can be as wide as the page itself, and the browser will add scroll bars if the text provided by the user exceeds the size of the text area. To create a text area, use the `textarea` element, specify the number of rows and columns using the `rows` and `cols` attributes, and name the text area using the `name` attribute. Default text can be specified by placing it between the `textarea` begin- and end-tags.

```
<p>Username: <input type="text" name="username" /></p>
<textarea name="comments" rows="2" cols="30">Default text</textarea>
<textarea name="comments" rows="5" cols="40"></textarea>
```

This is how the markup may appear in the browser. *NOTE*: The text in the second text area was added to demonstrate that scroll bars are added if the input text exceeds the size of the text area.



Hidden fields

A hidden field is a form component that allows name/value pairs to be specified without any visual representation. The hidden name/value pairs will be sent to the server for processing just like the pairs corresponding to the other form widgets. To create a hidden field, use the `input` element, specify `hidden` as the `type` attribute's value, and provide values for both the `name` and `value` attributes.

```
<input type="hidden" name="promotion_code" value="x3g9kf43" />
```

Hidden fields can be specified anywhere within the `form` begin- and end-tags.

Hidden fields are useful for providing context. For example, the hidden field in the above example would allow a script to identify any promotion that is associated with the Web page, perhaps for data mining purposes, without confusing the user by providing unnecessary information.

Submitting and resetting forms

Form data is submitted to the server when the user clicks on a submit button widget. *NOTE*: There are other methods for submitting form data without using a button widget; however, they are beyond the scope of this document. To create a submit button, use the `input` element, specify `submit` as the `type` attribute's value, and provide a label for the button widget via the `value` attribute.

```
<input type="submit" value="Let's Go!" />
```

This is how the markup may appear in the browser.

The `name` attribute is optional for the submit button. If present, then the browser will send a name-value pair to the server. For example, suppose you have a Web page with two separate forms corresponding to creating an account and signing in

using an existing account. The first form might have fields for address information, desired username, and password. The second form will likely only have two text boxes: username and password. The first form's submit button may be marked up as `<input type="submit" name="newacct" value="Create Account" />`. The second form's submit button may be marked up as `<input type="submit" name="signin" value="Sign In" />`. The script on the server can simply check for the existence of `newacct` or `signin` to determine the next course of action.

A typical companion to the submit button widget is the reset button widget, which resets a form to its original state (i.e., default values). To create a reset button, use the `input` element, specify `reset` as the `type` attribute's value, and provide a label for the button widget via the `value` attribute.

```
<input type="reset" value="Start Over" />
```

This is how the markup may appear in the browser.

To create a submit or reset button with an image, use the `button` element with an `img` child element. The `type`, `name`, and `value` attributes for the button element should either be `submit` or `reset` depending on the desired action.

```
<button type="submit" name="submit" value="submit">
  <img src="letsgo.gif" alt="Submit form data" />
</button>
```

This is how the markup may appear in the browser.

Organizing form content

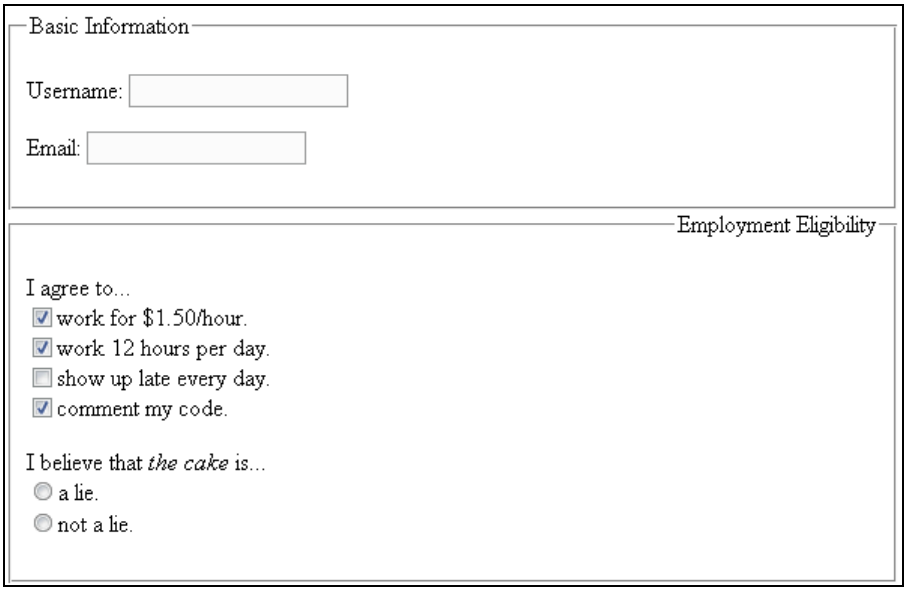
A form typically contains several types of elements for gathering information. As the form becomes more complex, it helps the user if these elements are organized into visually distinct groups. Legends, labels, and tab order can help organize form widgets so that the form is easier to understand for the user.

Legends

A legend visually groups widgets by placing a labeled border around them. To create a legend, use the `fieldset` element. The `fieldset` element contains two items: the legend and the form widgets that are to be grouped together. To create a legend, begin a `legend` element, specify the label text, then end the `legend` element. The legend element also accepts an optional value – `left`, `right`, or `center` – for the `align` attribute. The default alignment is `left`.

```
<fieldset>
  <legend>Basic Information</legend>
  <p>Username: <input type="text" name="username" /></p>
  <p>Email: <input type="text" name="email" /></p>
</fieldset>
<fieldset>
  <legend align="right">Employment Eligibility</legend>
  <p>I agree to...<br />
    <input type="checkbox" name="cheaplabor" value="yes" checked="checked" />work for $1.50/hour.<br />
    <input type="checkbox" name="longdays" value="yes" checked="checked" />work 12 hours per day.<br />
    <input type="checkbox" name="late" value="yes" />show up late every day.<br />
    <input type="checkbox" name="usecomments" value="yes" checked="checked" />comment my code.
  </p>
    <p>I believe that <i>the cake</i> is...<br  />
    <input type="radio" name="cake" checked="checked" />a lie.<br />
    <input type="radio" name="cake" />not a lie.
  </p>
</fieldset>
```

Here is how the markup may appear in the browser:



Labels

The XHTML `label` element allows an informative label to be presented with a widget. The `label` element has two advantages: (1) the focus will be given to the associated form widget when the user clicks on the label, and (2) the label/widget association can be used for client-side scripting purposes.

To make use of the `label` element, first assign a unique value to the `id` attribute for the widget to be labeled. Then provide this unique `id` to the `label` element's `for` attribute and provide text for the label:

```
<p>
  <label for="uname">Username:</label>
  <input type="text" name="username" id="uname" />
</p>
```

Labels typically do not change how the markup appears in the browser. The only noticeable difference is that the mouse cursor may appear as an *arrow* over an XHTML label instead of an *i-beam*.

A few things to note about labels:

- The XHTML `label` element is optional.
- Technically the `for` attribute is optional for the `label` element. If omitted, however, the label will not be associated with any widget and will be useless.
- The content of the label is not restricted to text; an image could be used as a label. Here's an example:

```
<p>
  <label for="uname"><img src="username_label.jpg" alt="Username" /></label>
  <input type="text" name="username" id="uname" />
</p>
```

Tab order

As with links, the tab order can be specified for the fields of a form. The default tab order depends on the order in which the form elements exist in the XHTML markup. Specifying a tab order allows the user to complete fields in a particular group before going on to the next group. To specify a tab order, use the `tabindex` attribute of any form element. For some reason Mozilla Firefox and Microsoft Windows Internet Explorer do not work properly with a `tabindex` of `0` (zero), so you should start your tab indices at `1` (one). Here's an example where controlling tab order would be useful:

```
<table>
  <tr>
    <td>Username:</td>
    <td><input type="text" name="username" tabindex="1" /></td>
    <td>Favorite programming language:</td>
    <td><input type="text" name="favlang" tabindex="3" /></td>
  </tr>
  <tr>
    <td>Password:</td>
    <td><input type="password" name="passwd" tabindex="2" /></td>
    <td>Favorite computer scientist:</td>
    <td><input type="text" name="favcompsci" tabindex="4" /></td>
  </tr>
</table>
```

Here's how the markup may appear in the browser:

Username:		Favorite programming language:	
Password:		Favorite computer scientist:	

The default tab order is "Username", "Favorite programming language", "Password", and "Favorite computer scientist" based on the order in which the form elements were given in the markup. However, username and password fields are typically grouped together. Therefore, the *preferred* order (as given in the markup above) is "Username", "Password", "Favorite programming language", and "Favorite computer scientist".

Recall the following about tab order:

- The value for `taborder` attribute should be between 0 (first) and 32767 (last).
- When the user is presented with the Web page, the first time the Tab key is pressed, the browser's URL text field will be given focus. Any subsequent presses of the Tab key will give the element with `taborder="0"` focus, then `taborder="1"`, and so on.
- For *selection* widgets, pressing the Tab key only gives the next form element the input focus; to manipulate the element, further keystrokes are necessary. For example, checkboxes and radio buttons can be "activated" by pressing the Return key. Menus can be manipulated by using the arrow keys, where pressing Enter sets the selected value.

Keyboard shortcuts

As with links, keyboard shortcuts can be specified to give focus to and activate fields of a form. To specify an access key, use the `accesskey` attribute of any form element.

```
<p>Username: <input type="text" name="username" accesskey="u" /></p>
<p>How would you rate your skill in programming?<br />
  <input type="radio" name="skill" value="beg" accesskey="g" />Beginner
  <input type="radio" name="skill" value="int" accesskey="i" />Intermediate
  <input type="radio" name="skill" value="adv" accesskey="a" />Advanced
  <input type="radio" name="skill" value="sup" accesskey="s" />Super-hacker
</p>
```

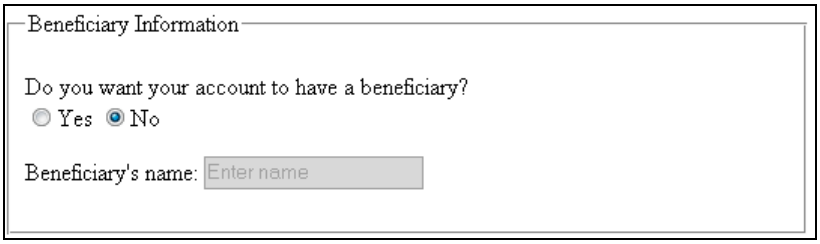
For Microsoft Windows browsers, keyboard shortcuts must be modified with the "Shift+Alt" key combination. For example, "Shift+Alt+A" will select the "Advanced" radio button in the markup example given above. For Macintosh browsers, keyboard shortcuts must be modified with the Control key.

Disabling elements

Situations may arise where you want form widgets to be displayed in the browser, but you do not want their values to be modified. To disable a form element, specify a value of `disabled` for the `disabled` attribute. The browser will "grey out" the element so that the user will not be able to interact with it (the grey out effect is browser-dependent). For example, a user could provide optional beneficiary information for a retirement account.

```
<fieldset>
  <legend>Beneficiary Information</legend>
  <p>Do you want your account to have a beneficiary?<br />
    <input type="radio" name="wantsbenef" value="true" />Yes
    <input type="radio" name="wantsbenef" value="false" checked="checked" />No
  </p>
  <p>Beneficiary's name:
    <input type="text" name="benefname" disabled="disabled" value="Enter name" />
  </p>
</fieldset>
```

Here's how the markup may appear in the browser:



The above example shows how the default value for the "Beneficiary's name" field is visible; however, the "greyed out" effect indicates that the field is disabled. *NOTE*: JavaScript would be required to enable the text field if the user changed the radio button state to "Yes", and likewise to disable the text field if the user changed the radio button state to "No".