**HTML Forms**

**Introduction**
Forms are used to submit data (on the client side) to a processing agent (on the server side). Form data is typically associated with one or more controls, such as edit controls, checkboxes, radio buttons, and so on. A typical scenario occurs when a user or client fills out an HTML form and then presses a submit button to submit all the data entered to a web server.

An example of this occurs during web forum registration, such as CSNS. After you fill out all your profile information in CSNS, you can hit the Update or Submit button to send the data to Dr. Sun’s web server, [http://sun.calstatela.edu](http://sun.calstatela.edu).

**Where to Find Information**
Information about HTML forms can be found at the following locations:

- HTML 4 Section 17: [http://www.w3.org/TR/html4/interact/forms.html](http://www.w3.org/TR/html4/interact/forms.html)
- HTML 5 Section 4.10: [http://dev.w3.org/html5/spec/Overview.html#forms](http://dev.w3.org/html5/spec/Overview.html#forms)
The FORM Element

Requirements
The FORM element is an HTML element container that holds any flow content element other than another FORM element (in other words, FORM elements may not be nested). FORM elements also require both start and end tags. Self-closing FORM tags are prohibited.

Controls*
A form is nothing without controls. A control is a special type of window** that is associated with a piece of data, called a value, that you would like to send (back) to a web server. A control is also associated with a name or identifier that gives both JavaScript and the web server the ability to process the data that comes from one or more controls. In other words, without a control name or identifier, how is JavaScript or a web server supposed to know that a string such as “7alcjzld” sent to it from a form is a password? It can’t, and therefore controls must always come in name-value pairs. When submitted, the controls that these name-value pairs come from are called “successful controls.”
On a funny note, the HTML 4 standard uses the word “control” while the HTML 5 standard prefers the term “form-associated element.” They both mean the same thing; they are elements, with name and value attributes, that can partake in form submission.

** In web browsers, controls are not actually windows. HTML controls are actually called windowless or lightweight controls since the implementation of these controls is completely determined by the user agent rather than by the operating system. More on this when you take CS202!

In HTML 4 there are many different types of controls:

1.) Buttons
2.) Checkboxes
3.) Radio Buttons
4.) Menus
5.) Text Input
6.) File Select
7.) Object Controls
8.) Hidden Controls

HTML 5 adds many more new controls such as date/time and color controls; however, currently none of these newer controls are supported by any modern browser yet.

The INPUT Element
The INPUT element is used to create several different kinds of basic text and mouse input-type controls. More complicated controls such as the menu control need their own special element since they are not basic text and mouse input-type controls. The INPUT element does not require an end tag, and you may also use self-closing tags if you wish.

Push Buttons
To create a push button, use the attribute/value pair type=“BUTTON” and be sure to give your push button a unique name/id and value. The value is the actual button text and the name is used by JavaScript and the web server to access data related to the state of the button.

HTML Code

```
<form>
  <input name="OK" type="button" value="Ok">
</form>
```

Push buttons are also affected by cascading style sheets. You can create some beautiful buttons by overriding the default grayish button styles. However, when drastically changing the default style to a new one, be sure to override the appropriate pseudo-class selectors (focus, hover, and active) to make the buttons look right while dragging the mouse around while the button is in the active state. If you’re not careful in overriding the appropriate pseudo-class selectors, your button might not behave like a regular button, which might annoy your visitors.
CSS Code

```css
.Pretty {
  border-style: outset;
  border-width: 2px;
  border-color: blue;
  background-color: #DDEEFF;
  width: 79px;
  height: 29px;
  font-family: 'Times New Roman';
  font-style: italic;
  font-weight: bold;
  color: red;
}
.Pretty:focus {
  border-style: outset;
}
.Pretty:active {
  border-style: outset;
}
.Pretty:active:hover {
  border-style: inset;
}
```

HTML Code

```html
<form>
  <input class="Pretty" name="OK" type="button" value="Ok"><p>
  <input class="Pretty" name="CANCEL" type="button" value="Cancel"><p>
  <input class="Pretty" name="ABORT" type="button" value="Abort"><p>
</form>
```
Checkboxes
A checkbox is a type of control that is in either one of two states: checked and unchecked. These controls serve as on-and-off switches for which the user can toggle back and forth between the two states. It is important to note that unlike buttons, checkboxes do not use the VALUE attribute to associate the checkbox control with a text label. Instead, use the LABEL element immediately before or after the INPUT element to show the text.

The LABEL element requires both a start tag and an end tag. Self-closing tags cannot be used. The content of the LABEL element is the phrasing content you want associated with your checkbox control. Without the LABEL element (if ordinary phrasing content was used instead), only checking on the checkbox would check or uncheck the control; but with the LABEL element beside the checkbox control, clicking on the label text will also check or uncheck the checkbox control. Use the FOR attribute of the LABEL element to link the label to the appropriate control.

To create a checkbox, use the attribute/value pair type="CHECKBOX" and be sure to give your checkbox a unique identifier instead of a name, since labels only work with identifiers. Then add the LABEL element associated with the checkbox immediately before or after the INPUT element.

HTML Code

```html
<form>
<input type="checkbox" id="CHOICE_01" checked="checked">
<label for="CHOICE_01">Check me!!!(</label>
<br/>
<label for="CHOICE_02">Check me!!!(</label>
<input type="checkbox" id="CHOICE_02" checked="checked">
</form>
```

Checkboxes are not really affected by cascading style sheets to the extent that buttons are. Custom checkboxes actually requires some crafty CSS and JavaScript coding for which might not work on all browsers. In fact, the CSS 2.1 standard states, “Notably for HTML, user agents may render borders for certain user interface elements (e.g., buttons, menus, etc.) differently than for ordinary elements.”
Other than that, there is really nothing in the CSS standard that states how CSS styles should affect individual controls such as checkboxes. Therefore, avoid applying CSS styles to checkboxes. As an example, here is some HTML/CSS code and the results in both Firefox 3.5 and Internet Explorer 8. As you can see, putting a background color around the checkbox doesn’t really produce what you would think would be the desired effect (a red checkbox).

CSS Code

```css
input {
    background-color: red;
}
```

HTML Code

```html
<form>
    <input type="checkbox" id="iCHOICE_01" checked="checked">
    <label for="iCHOICE_01">Check me!!!</label>
    <br />
    <label for="iCHOICE_02">Check me!!!</label>
    <input type="checkbox" id="iCHOICE_02" checked="checked">
</form>
```

Radio Buttons

A radio buttons is a type of control that is in either one of two states: checked and unchecked. Like checkboxes, these controls serve as on-and-off switches for which the user can toggle back and forth between the two states. Radio buttons are very similar to checkboxes, with one difference: radio buttons are divided into groups for which only one item per group can be checked. Radio buttons should also specify labels, just like checkboxes do, so that clicking on the text also changes the state of the control.

Radio buttons are great for questionnaires where each question only has one possible answer. If there is more than one answer per question, checkboxes are the choice instead. Using a radio button, you can prevent a person from selecting more than one button by dividing your radio buttons into a named
group. To define a set of radio button, we need to define at least three INPUT attributes in each radio button. A fourth attribute, CHECKED, is optional.

1.) TYPE: The type will always be RADIO.
2.) NAME: The name of your radio button group.
3.) ID: A unique identifier for each radio button.
4.) CHECKED: Equal to “checked” if radio button is to be initially checked.

Directly below are two questions, each divided into two groups, Q1 and Q2. Each group has five unique radio buttons, given identifiers Q1_A1, Q1_A2, etc. If you try it out, you will notice that only one radio button per group can be checked.

```
<html>
<body>
<form>
<p>1.) What is your instructor's name?</p>
<input name="Q1" id="Q1_A1" type="radio" />
<label for="Q1_A1">Steven</label><br />
<input name="Q1" id="Q1_A2" type="radio" />
<label for="Q1_A2">Erin</label><br />
<input name="Q1" id="Q1_A3" type="radio" />
<label for="Q1_A3">Richard</label><br />
<input name="Q1" id="Q1_A4" type="radio" />
<label for="Q1_A4">Eric</label><br />
<input name="Q1" id="Q1_A5" type="radio" checked="checked" />
<label for="Q1_A5">None of the above</label>
<p>2.) What color is the sky?</p>
<input name="Q2" id="Q2_A1" type="radio" />
<label for="Q2_A1">Green</label><br />
<input name="Q2" id="Q2_A2" type="radio" />
<label for="Q2_A2">Blue</label><br />
<input name="Q2" id="Q2_A3" type="radio" />
<label for="Q2_A3">Orange</label><br />
<input name="Q2" id="Q2_A4" type="radio" />
<label for="Q2_A4">Yellow</label><br />
<input name="Q2" id="Q2_A5" type="radio" checked="checked" />
<label for="Q2_A5">None of the above</label>
</form>
</body>
</html>
```
Of course, as with checkboxes, do not use CSS to try to modify the style of a radio button. It just won’t look right, nor will it be consistent from browser to browser. For example, the following CSS code added to the above example yields the following results in Firefox 3.5 and Internet Explorer 8.

```
input[type=radio] {
  background-color:#DDEEFF;
}
```
Reset Buttons

A reset button is just like a regular push button. However, a push button does one extra thing: it resets all controls within the same form to their original states or values. For example, in our radio button example, let’s say we wanted to revert our selections back to their original states, to “None of the above.” Just add the following code to the FRAME section of your previous radio button example to add a reset button and see what happens. Notice that no matter what you choose, once you hit the reset button, both radio button groups go back to “None of the above.”

To specify a reset button in HTML, set the TYPE attribute of the INPUT element to “reset.”
Also note that like push buttons, you can use CSS to modify the default style of the submit button. The behavior is pretty similar no matter which browser you use in this case.

**Submit Buttons**

A submit button is just like a regular push button. However, a submit button does one extra thing: it triggers a form submission. When a form is submitted, an action is executed, which is specified using the ACTION attribute of the FORM element. The simplest type of action is to specify a URL to go to, which is shown in the following example. Here, using the same radio button quiz example from before, but with a submit button and an action to “thanks.html,” clicking on the submit button will take you to the “thank you for taking the quiz” web page.

To specify a submit button in HTML, set the TYPE attribute of the INPUT element to “submit.” Also, don’t forget to set the ACTION attribute of the FORM element.
2.) What color is the sky?

Also note that like push buttons, you can use CSS to modify the default style of the submit button. The behavior is pretty similar no matter which browser you use in this case.

**Single-Line Input Controls**

The profile form from CSNS is filled with a bunch of single-line input controls. We use these types of controls in cases where we only need or are expecting a short line of text. In Microsoft Windows these types of controls are also called single-line edit controls.

To define a single-line input control, set the TYPE attribute of the INPUT element to “text.” If you want to have some initial text populated in the control, you may add it using the VALUE attribute of the INPUT element as well.

Single-line input controls may utilize CSS to create pretty input controls.
Text Controls

Password Controls
The profile form from CSNS also contains one password control. A password control is the same thing as a single-line input control, but with one difference: the password control only displays bullets instead of the characters that you actually type.

To define a password control, set the TYPE attribute of the INPUT element to “password.” If you want to have a default password populated in the control, you may add it using the VALUE attribute of the INPUT element as well.
Please note that while a password control hides the password text visually in the control, the data is transferred out of the form in plain text, for everyone to see. A password control is mainly meant to prevent nearby “prying eyes” from looking at your password.

![Password Controls](image)

**File Select Controls**

On CSNS, there is a file select control on the page where you upload your files. A file select control is actually two controls put together side-by-side: an edit control and a browse button. To define a file select control, set the TYPE attribute of the INPUT element to “file.” If you want to have a default filename populated in the edit control, you may add it using the VALUE attribute of the INPUT element as well.
As is done on CSNS, do not try to apply any CSS styles to the file select control. While there is a way to get that browse button to look like the upload and done buttons, but it is very tricky and requires an unfun HTML/CSS hack.

**Image Buttons**

An image button is just a custom image submit button. To define an image button control, set the TYPE attribute of the INPUT element to “image” and set the image using the SRC attribute of the INPUT element.
Hidden Controls

Hidden controls are controls that are typically used by JavaScript to store data before it is shipped off to the web server. For this course they aren’t that important, so don’t worry about hidden controls for now. You will use them in CS320 quite a bit however.

Advanced Controls

The Button Element

The BUTTON element is very similar to the INPUT element version, but the BUTTON element may contain phrasing content such as text and images that can be put inside a button.
The SELECT Element
The SELECT element defines what is known to most Microsoft Windows users as an extended combo box, or drop down list box. Each list item is defined by an OPTION element, and if desired, you can group your OPTION elements into groups using the OPTGROUP element. To set the default selection, you can
use the SELECTED attribute of the OPTION element. Only OPTION elements can be selected; OPTGROUPs cannot be selected.

HTML Code

```html
<form action="thanks.html">
  <select name="RATINGS1" title="Ratings1">
    <option>1</option>
    <option>2</option>
    <option>3</option>
    <option>4</option>
    <option selected="selected">5</option>
    <option>6</option>
    <option>7</option>
    <option>8</option>
    <option>9</option>
    <option>10</option>
  </select>
  <select name="RATINGS2" title="Ratings2">
    <optgroup label="Sucked really bad.">
      <option>1</option>
      <option>2</option>
      <option>3</option>
    </optgroup>
    <optgroup label="It was painful; but I lived.">
      <option>4</option>
      <option selected="selected">5</option>
      <option>6</option>
    </optgroup>
    <optgroup label="Not bad at all.">
      <option>7</option>
      <option>8</option>
      <option>9</option>
    </optgroup>
    <optgroup label="Banzai!!!">
      <option>10</option>
    </optgroup>
  </select>
  <input type="submit" value="Rate Me"> &nbsp; <input type="reset">
</form>
```
The TEXTAREA Element
The TEXTAREA element defines a multi-line edit control. The content of this element is the initial text of the multi-line edit control. Be sure to give the control a name and specify the number of expected rows and columns of visible text. You can also use CSS to set the dimensions of a TEXTAREA element and to make it look pretty.

HTML Code
```
<form action="thanks.html">
  <textarea name="COMMENTS" rows="10" cols="80">Initial text.</textarea>
  <br />
  <input type="submit" value="Submit Comment">&nbsp;<input type="reset">
</form>
```
GET Versus POST

There are two ways to send data to a web server through the HTTP protocol using forms. One is called GET and the other is called POST. To specify a particular method, use the METHOD attribute of the FORM element. The two supported values for this attribute are GET and POST.

To test out POST, let’s send the above TEXTAREA example to my web server using the POST method. Here is the form code that we are going to run.

HTML Code
<form action="http://www.steven-emory.com" method="post">
<textarea name="COMMENTS" rows="10" cols="80">Initial text.</textarea>
<br />
<input type="submit" value="Submit Comment"/>
<input type="reset">
</form>
Wow, this movie was like the worst of all time! It was so bad I almost puked my guts out and lost my popcorn on my girlfriend! Oh wait, what girlfriend? You see... that movie was so bad it made me so delirious I almost forgot I almost lost my lunch on someone else's girlfriend he he ha!
After pressing Submit Comment, the form navigates to my homepage. Pay specific attention to the URL.

The data the server got was:

```
-- post
192.168.1.1 - - [] "POST / HTTP/1.1" 200 511
192.168.1.1 - - [] "GET /stylesheets/common.css HTTP/1.1" 200 37
192.168.1.1 - - [] "GET /stylesheets/default.css HTTP/1.1" 200 42
192.168.1.1 - - [] "GET /favicon.ico HTTP/1.1" 404 209
```

Where did the message in the TEXTAREA go? Well, it was sent within the body of the POST message sent to the server. Now let's try using GET instead of POST. Here is the code:

```
HTML Code
<form action="http://www.steven-emory.com" method="get">
<textarea name="COMMENTS" rows="10" cols="80">Initial text.</textarea>
<br />
<input type="submit" value="Submit Comment"> &nbsp; <input type="reset">
</form>
```
Wow, this movie was like the worst of all time! It was so bad I almost puked my guts out and lost my popcorn on my girlfriend! Oh wait, what girlfriend? You see... that movie was so bad it made me so delirious I almost forgot I almost lost my lunch on someone else's girlfriend he ha ha!

Forums

This website is for me and my CS120 students.

© 2009 Steven Emory

Contact Me
After submission, notice anything about the URL in the address bar now? Has a bunch of junk right?
Using GET, the data from the TEXTAREA still gets sent to the web server, but in a slightly different way (though the URL). And if we look at the data the web server got from us:

192.168.1.1 - - [] "GET
/?COMMENTS=Wow%2C+this+movie+was+like+the+worst+of+all+time%21+It+was+so+bad+I+almost+puked+my+guts+out+and+lost+my+popcorn+on+my+girlfriend%21+Oh+wait%2C+what+girlfriend%3F+You+see...+that+movie+was+so+bad+it+made+me+so+delirious+I+almost+forgot+I+almost+lost+my+lunch+on+someone+else%27s+girlfriend+ha+ha+ha%21 HTTP/1.1" 200 511

It is actually quite different than the data we got before. I will explain the difference in class, but in general this stuff is better covered in CS320. All that is really important is that you know there are two different methods of sending data to a web server through a form: one is used to post data to the server (making changes) while the other is used to retrieve/get/query data from the server (without making changes).