

Web Authoring: Forms

Academic Computing Services
A Division of Information Services

www.ku.edu/acs

Abstract: Learn to collect information from visitors to your Web site by providing text boxes, checkboxes, menus, and other input controls with HTML forms.

Contents

Introduction	3
Objectives	3
Prerequisites	3
Related Training Available from ACS.....	3
HTML Forms	3
Controls.....	3
Control Types.....	4
The Back End.....	5
The <code>form</code> Element.....	6
Attributes for <code>form</code>	6
The <code>input</code> Element.....	7
Attributes for <code>input</code>	7
Control Types Created with <code>input</code>	9
The <code>select</code> , <code>optgroup</code> , and <code>option</code> Elements.....	16
The <code>select</code> Element.....	16
The <code>optgroup</code> Element.....	17

ACS Computer Training

Web Authoring: Forms

The <code>option</code> Element.....	17
Exercises.....	18
The <code>textarea</code> Element.....	20
Attributes for <code>textarea</code>	20
Exercise	21
Advanced Control Types.....	22
The <code>button</code> Element.....	22
File Select Controls	24
Password Entry Fields	25
Hidden Controls	26
Object Controls	27
Labels.....	27
The <code>label</code> Element.....	27
Exercises.....	28
Adding Structure to Forms: the <code>fieldset</code> and <code>legend</code> Elements	28
Keyboard Access to Form Controls	29
Form Submission	30
For More Information.....	30
Online.....	31
Getting Additional Help	32

Introduction

Online forms enable you to gather information from visitors to your Web site.

Objectives

The goal of this workshop is to introduce participants to HTML form syntax. After today's workshop, participants will be able to:

- Produce HTML documents with forms.
- Understand the importance of the scripts that go with forms to process the data forms collect.

Prerequisites

Web Authoring: Intermediate

Related Training Available from ACS

All workshops offered by Academic Computing Services (ACS), a division of Information Services, are free to KU students, staff, faculty, and [approved affiliates](#). The general public is also welcome to most workshops, but some ACS workshops require a [registration fee](#) for them.

To learn more about or register for workshops, receive automatic announcements of upcoming workshops, and track workshops you've registered for and have attended, visit the ACS Web site at www.ku.edu/acs/train. You can also check our online schedule at www.ku.edu/acs/schedule for a list of class offerings and their availability. For further workshop related questions, please email training@ku.edu.

HTML Forms

HTML documents can contain forms that users can “complete” and submit to the server. This provides a means for information (besides simple requests) to be directed from the client to the server, making for two-way information flow between them. (This is in contrast to ordinary HTTP client-server interactions, in which the information flow consists almost exclusively of data, i.e., HTML content, being sent in one direction: from the server to the client.)

Forms consist of special elements called controls (checkboxes, radio buttons, menus, etc.) and labels for those controls, as well as normal content and markup. Users complete forms by modifying the controls. When the completed form is submitted, the browser sends the collected data to a specified HTTP server.

Controls

Each form control has a name and a value. A control's name is specified by the control element's `name` attribute, and is used to identify the value that control represents.

The value of a control is first set to a specified initial value. Users modify control values by interacting with the controls. Resetting a form returns all controls to their initial

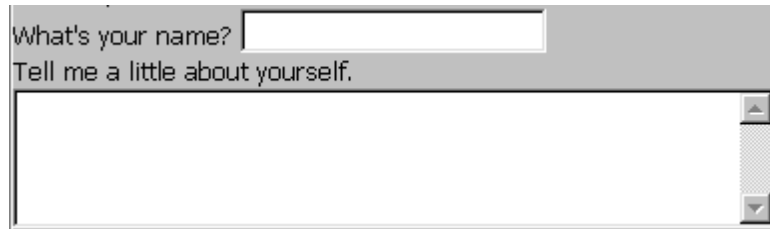
values. Submitting a form pairs controls' names with their current values and causes these pairs to be sent to the server.

Control Types

See <http://www.ku.edu/acs/training/classes/wa-forms/controls.html> for examples of form control types. Forms can contain any of the following types of controls:

text input

There are two types of text input controls: single-line and multi-line. The `input` element creates a single-line input control, while the `textarea` element creates a multi-line input control.

A screenshot of a web form. The top part contains the text "What's your name?" followed by a single-line text input field. Below that is the text "Tell me a little about yourself." followed by a multi-line text area with a vertical scrollbar on the right side.

Text input controls

buttons

There are three types of buttons: submit buttons, reset buttons, and push buttons. A submit button submits the form. A reset button resets all form controls to their initial values. Push buttons have no default behavior; they are used to trigger associated client-side scripts. *Client-side scripts are beyond the scope of this class.*

Buttons are created with the `button` element or the `input` element.

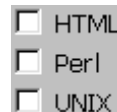


Button controls

checkboxes

Checkboxes are on/off switches that can be toggled by the user. A checkbox is "on" when the control element's `selected` attribute is set.

Multiple checkboxes in a form can share the same control name, providing a means for users to select several values for the same property. The `input` element is used to create a checkbox control.



Checkbox controls

radio buttons

Radio buttons are like checkboxes except that when several share the same control name, they are mutually exclusive: when one is switched "on", all others

with the same name are switched “off”. The `input` element is used to create a radio button control.



Radio button controls

menus

Menus offer users options from which to choose. The `select` element creates a menu, in combination with the `optgroup` and `option` elements.



Menu controls

file select

This control type allows users to select files so that their contents may be submitted with a form. The `input` element is used to create a file select control.



A file select control

hidden controls

You can create controls that are not rendered but whose values are submitted with the form. This control type is generally used to store information between client/server exchanges that would otherwise be lost due to the stateless nature of HTTP. The `input` element is used to create a hidden control.

object controls

You can insert generic objects in forms such that associated values are submitted along with other controls. Object controls are created with the `object` element.

The use of objects in forms is beyond the scope of this class.

The Back End

It's important to understand that HTML forms provide only the front-end user interface for the collection and submission of data, which does nothing by itself. In order to function, *a form must be combined with a script* that receives and processes the information entered into the form by the user. This script is written in a programming language, *not* HTML.

Because this class covers only the HTML used to produce the forms themselves, we will rely in our exercises on a very basic prepared script that simply lists the values submitted to it by any form.

To learn to write your own scripts for processing form data, consider attending the ACS class *Web Authoring: CGI Scripts*.

The *form* Element

The `form` element acts as a container for controls. It specifies:

- The layout of the form (given by the contents of the element).
- The program that will handle the completed and submitted form (the `action` attribute).
- The method by which user data will be sent to the server (the `method` attribute).
- A character encoding that must be accepted by the server in order to handle this form (the `accept-charset` attribute).

A form can contain text and markup (paragraphs, lists, etc.) in addition to form controls.

Attributes for `form`

`action` = *URL*

The `action` attribute specifies the script that will process the form. (See **The Back End** on page 5.) This attribute is required.

`method` = `get` or `post`

The `method` attribute specifies which HTTP method will be used to submit the form data to the server. The default value is “`get`”.

`enctype` = *MIME type*

The `enctype` attribute specifies the content type used to submit the form to the server (when the value of `method` is “`post`”). The default value is “`application/x-www-form-urlencoded`”. The value “`multipart/form-data`” should be used in combination with `input` elements whose type attributes are set to “`file`”. *The default value is acceptable in most other cases.*

`accept-charset` = *character set list*

The `accept-charset` attribute specifies the list of character encodings for input data that must be accepted by the server processing this form. The value is a space- and/or comma-delimited list of character set values. The server must interpret this list as an exclusive-or list, i.e., the server must be able to accept any single character encoding per entity received.

The default value for this attribute is the reserved string “`UNKNOWN`”. Browsers may interpret this value as the character encoding that was used to transmit the document containing this `form` element. *The default value is acceptable in most cases.*

`accept` = *MIME type list*

The `accept` attribute specifies a comma-separated list of content types that a server processing this form will handle correctly. Browsers may use this information to filter out non-conforming files when prompting a user to select files to be sent to the server (cf. the `input` element when `type` = “`file`”). *The default value is acceptable in most other cases.*

The following common attributes can also be used with `form` elements:

- `id`, `class` (identifiers)
- `lang` (language information), `dir` (text direction)
- `style` (inline style information)
- `title` (element title)
- `target` (target frame information)
- `onsubmit`, `onreset`, `onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup` (events)

The *input* Element

The `input` element defines a form control.

Attributes for `input`

`type` = `text` or `submit` or `checkbox` or `radio` or `reset` or `image` or `button` or `file` or `password` or `hidden`

The `type` attribute specifies the type of control to create. See **Control Types Created with `input`** on page 9 for information about the possible values for this attribute. The default value is “text”.

`name` = *text*

The `name` attribute assigns the control name. This attribute is required for all types but “submit” and “reset”.

`value` = *text*

The `value` attribute specifies the initial value of the control. It is required for types “checkbox” and “radio”.

`size` = *integer*

The `size` attribute tells the browser the initial width of the control. The width is given in pixels except when the `type` attribute has the value “text” or “password”. In that case, its value refers to the number of characters.

`maxlength` = *integer*

When the `type` attribute has the value “text” or “password”, the `maxlength` attribute specifies the maximum number of characters the user may enter. This number may exceed the specified size, in which case the browser should offer a scrolling mechanism. The default value for this attribute is an unlimited number.

`checked`

When the `type` attribute has the value “radio” or “checkbox”, this Boolean attribute specifies that the button is on.

`src` = *URL*

When the `type` attribute has the value “image”, this attribute specifies the location of the image to be used to decorate the graphical submit button.

`accept`

The `accept` attribute is the same for `input` as it is for `form` (see page 6).

`readonly`

When set, this Boolean attribute prohibits changes to the control. The `readonly` attribute specifies whether the control may be modified by the user.

Note: The only way to modify dynamically the value of the `readonly` attribute is through a client-side script. *Client-side scripts are beyond the scope of this class.*

`disabled`

When set, this Boolean attribute disables the control for user input.

Note: The only way to modify dynamically the value of the `disabled` attribute is through a client-side script. *Client-side scripts are beyond the scope of this class.*

`tabindex = integer`

The `tabindex` attribute specifies the position of the current element in the tabbing order for the current document. This value must be a number between 0 and 32767.

With tabbing navigation, your Web pages can be keyboard navigable in much the same way as many dialog boxes are in the Windows and Mac OS operating systems. The tabbing order defines the order in which elements will receive focus when navigated by the user via the keyboard.

Note: Tabbing navigation is a recent addition to HTML, and may not yet be supported by all browsers. Be sure to test your pages thoroughly before relying on its implementation.

`accesskey = character`

The `accesskey` attribute assigns an access key to the element. An access key is a single character from the document character set. Pressing the assigned access key gives focus to the element.

As with tabbing navigation, access keys provide keyboard navigation of your Web pages. The invocation of access keys depends on the underlying system: one may have to press an “alt” or “⌘” key in addition to the access key, for example.

Note: Access keys are a recent addition to HTML, and may not yet be supported by all browsers. Be sure to test your pages thoroughly before relying on their implementation.

The following common attributes can also be used with `input` elements:

- `id`, `class` (identifiers)

- `lang` (language information), `dir` (text direction)
- `title` (element title)
- `style` (inline style information)
- `alt` (alternate text)
- `usemap` (client-side image maps)
- `onfocus`, `onblur`, `onselect`, `onchange`, `onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup` (events)

Control Types Created with `input`

The control type defined by the `input` element depends on the value of the `type` attribute:

text

Creates a single-line text input control.

submit

Creates a submit button.

checkbox

Creates a checkbox.

radio

Creates a radio button.

reset

Creates a reset button.

image

Creates a graphical submit button. The value of the `src` attribute specifies the URL of the image that will decorate the button. Mouse click coordinates within the image are added to the input value when the form is submitted, accommodating image maps. For accessibility reasons, you should provide alternate text for the image via the `alt` attribute.

button

Creates a push button.

file

Creates a file select control.

password

Like “text”, but the input text is rendered in such a way as to hide the characters.

hidden

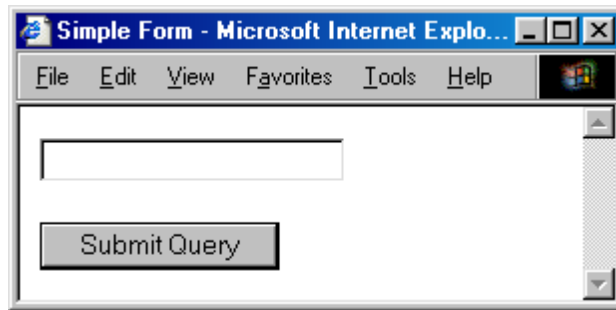
Creates a hidden control.

Exercises

The exercises that follow demonstrate the use of most of these control types.

Text Input

As a first exercise, create a page with a simple form consisting of a text input for collecting the user's name. This form will also have a submit button, as all forms must for the information entered to be collected. Both controls are `input` elements; the text input has `type="text"`, the submit button `type="submit"`.



A simple form with a text input control and a submit button

Here is the HTML source for this example:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">

<TITLE>Simple Form</TITLE>

<FORM action=
"http://www.ku.edu/cgiwrap/acs/training/classes/wa-forms/forms.pl"
method="post">

<P>
<INPUT type="text" name="name">

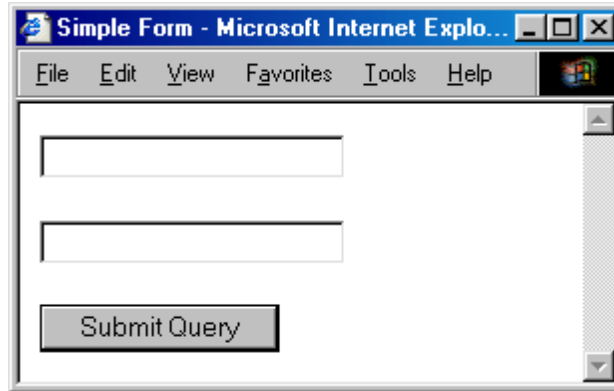
<P>
<INPUT type="submit">
</FORM>
```

Note that the URL given in the `action` attribute of the `form` element represents the generic, prepared script referred to on page 5. It will be used in all exercises in this class, since creating the required form processing script is beyond the scope of this class. To learn to write your own scripts for processing form data, consider attending the ACS class *Web Authoring: CGI Scripts*.

Enter this in a text editor and save it as **form.html** as directed by the instructor. When finished, open the file in your Web browser to test it.

Adding a Second Text Input

Add a second text input for collecting the user's email address.



The form with a second text input control

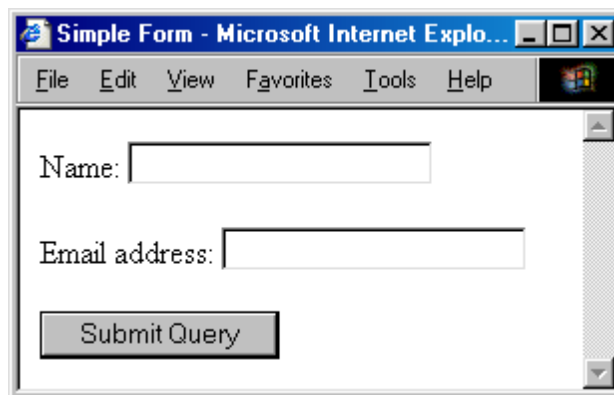
Here is a portion of the source, with the new text highlighted:

```
<P>  
<INPUT type="text" name="name">  
  
<P>  
<INPUT type="text" name="address">  
  
<P>  
<INPUT type="submit">
```

Notice that the name attribute values are different for the two text inputs. Add the second control to **form.html**, save, and test in your Web browser.

Distinguishing the Controls

There's nothing to tell the user what is to go into the text fields, so add some text to label the controls.



The form with text labeling the text input controls

```
<P>  
Name: <INPUT type="text" name="name">  
  
<P>  
Email address: <INPUT type="text" name="address">
```

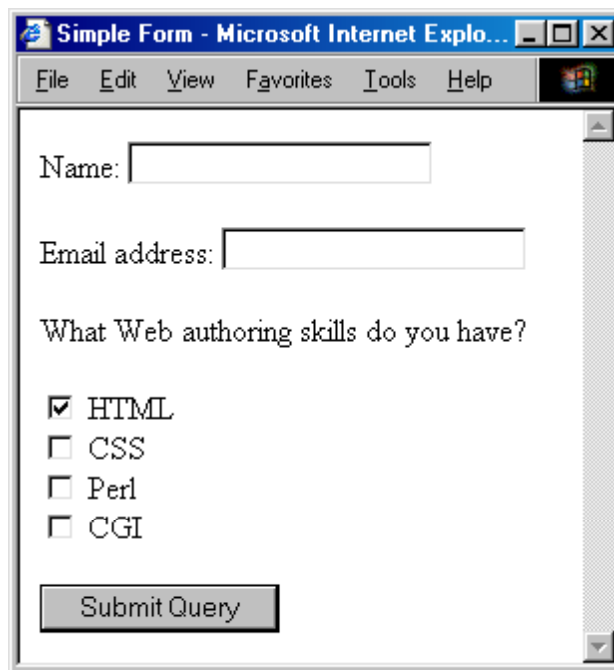
Add the highlighted text to **form.html**, save, and reload the page in your Web browser.

The text to the left of each text input control serves to label that control. Later, we'll see how to formally associate labels with controls, adding structure and eliminating reliance on visual layout (in **Labels** on page 27).

Checkboxes

An input with `type="checkbox"` represents a checkbox on the page. All checkbox controls in a group share the same name. The `value` attribute determines the value submitted by the selected control. The additional attribute `checked` can be used to initially select a checkbox.

Add a set of checkboxes to your form:

A screenshot of a Microsoft Internet Explorer browser window titled "Simple Form - Microsoft Internet Expl...". The browser's menu bar includes "File", "Edit", "View", "Favorites", "Tools", and "Help". The form content includes a "Name:" label followed by a text input field, an "Email address:" label followed by a text input field, and a question "What Web authoring skills do you have?". Below the question are four checkboxes: "HTML" (checked), "CSS", "Perl", and "CGI". At the bottom of the form is a "Submit Query" button.

The form with a set of checkboxes added

```
<P>  
Email address: <INPUT type="text" name="address">  
  
<P>  
What Web authoring skills do you have?
```

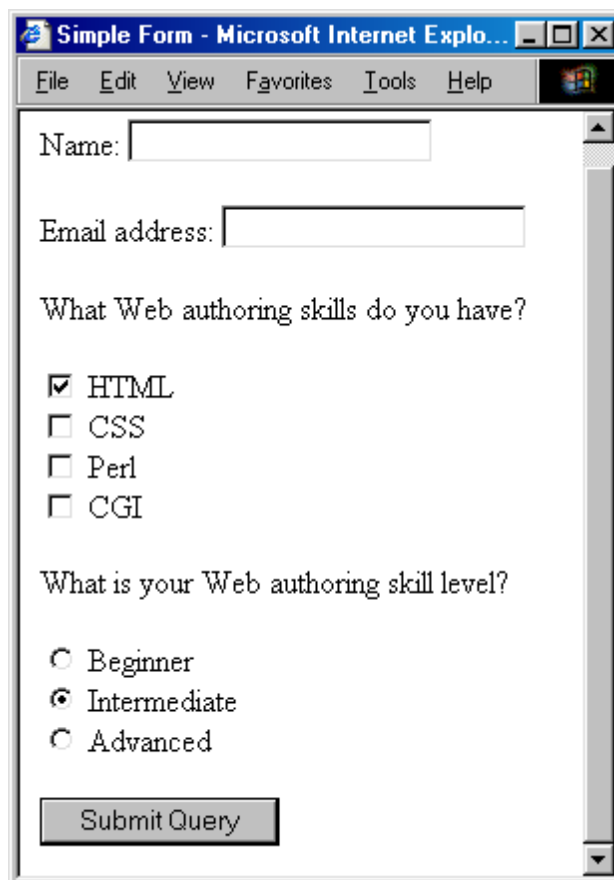
```
<P>  
<INPUT type="checkbox" name="skills" value="HTML" checked> HTML<BR>  
<INPUT type="checkbox" name="skills" value="CSS"> CSS<BR>  
<INPUT type="checkbox" name="skills" value="Perl"> Perl<BR>  
<INPUT type="checkbox" name="skills" value="CGI"> CGI  
  
<P>  
<INPUT type="submit">
```

Add the highlighted text to **form.html**, save, and reload the page in your Web browser.

Radio Buttons

An `input` with `type="radio"` represents a radio button on the page. Radio buttons are just like checkboxes, except that only one radio button in a group can be selected at one time. Clicking one radio button in a group turns off whichever radio button was selected before.

Add a set of radio buttons to your form:

A screenshot of a Microsoft Internet Explorer browser window titled "Simple Form - Microsoft Internet Explo...". The browser's menu bar includes "File", "Edit", "View", "Favorites", "Tools", and "Help". The form content includes a "Name:" text input field, an "Email address:" text input field, and a question: "What Web authoring skills do you have?". Below this question are four checkboxes: "HTML" (checked), "CSS", "Perl", and "CGI". Another question follows: "What is your Web authoring skill level?". Below this question are three radio buttons: "Beginner", "Intermediate" (selected), and "Advanced". At the bottom of the form is a "Submit Query" button.

The form with a set of radio buttons added

```
<INPUT type="checkbox" name="skills" value="CGI"> CGI
```

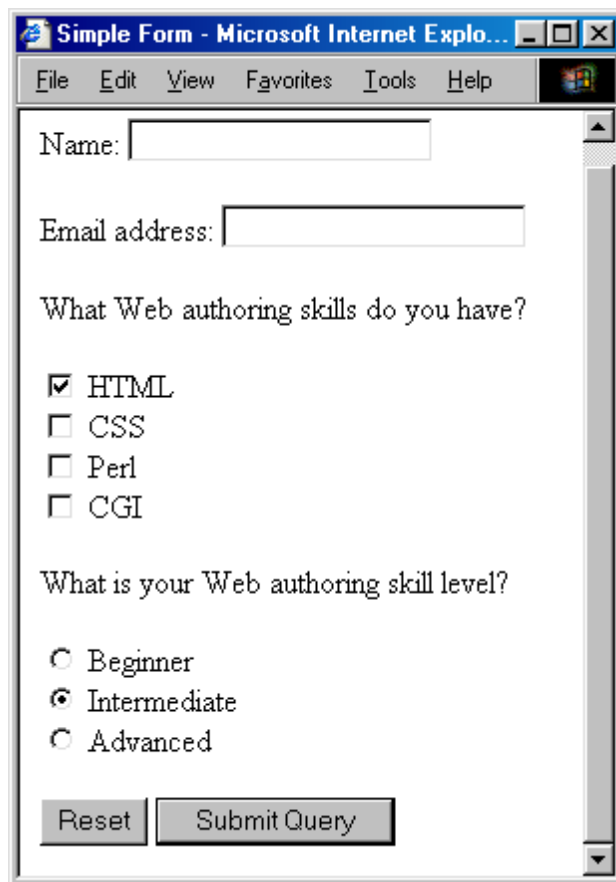
```
<P>  
What is your Web authoring skill level?  
  
<P>  
<INPUT type="radio" name="level" value="beginner"> Beginner<BR>  
<INPUT type="radio" name="level" value="intermediate" checked>  
Intermediate<BR>  
<INPUT type="radio" name="level" value="advanced"> Advanced  
  
<P>  
<INPUT type="submit">
```

Add the highlighted text to **form.html**, save, and reload the page in your Web browser.

Reset Button

An input with `type="reset"` represents a reset button, which resets all controls in the form to their initial (often cleared) values. This can be useful when the form is typically filled-in repeatedly by the same user with significantly different information from one use to the next.

Add a reset button to your form:



The form with a reset button added

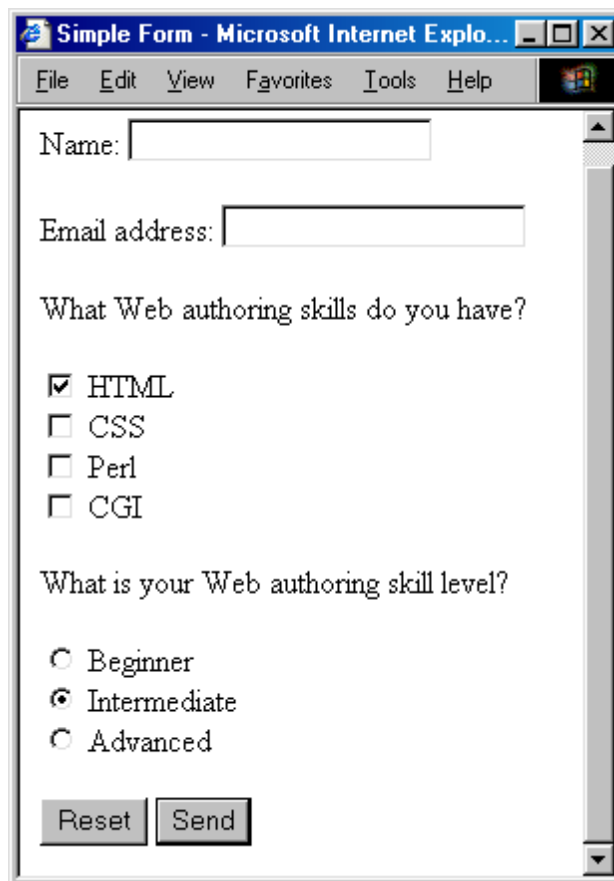
```
<P>  
<INPUT type="reset"> <INPUT type="submit">  
</FORM>
```

Add the highlighted text to **form.html**, save, and reload the page in your Web browser.

Button Values

For both submit and reset buttons, the value attribute can optionally be used to set the text that appears on the button's face to something other than the default.

Change the submit button to read "Send":



The screenshot shows a web browser window with a form titled "Simple Form". The form contains the following elements:

- A text input field labeled "Name:".
- A text input field labeled "Email address:".
- A question: "What Web authoring skills do you have?"
- Four checkboxes: "HTML" (checked), "CSS", "Perl", and "CGI".
- A question: "What is your Web authoring skill level?"
- Three radio buttons: "Beginner", "Intermediate" (selected), and "Advanced".
- Two buttons at the bottom: "Reset" and "Send".

The form with the submit button set to read "Send"

```
<P>  
<INPUT type="reset"> <INPUT type="submit" value="Send">  
</FORM>
```

Add the highlighted text to **form.html**, save, and reload the page in your Web browser.

The select, optgroup, and option Elements

The `select` element creates a pop-up menu or scrolled list box. Each choice offered by the menu or list box is represented by an `option` element. A `select` element must contain at least one `option` element.

The `optgroup` element allows you to group choices logically. Option groups are typically displayed through a hierarchical menu. In HTML 4.0x, all `optgroup` elements must be specified directly within a `select` element (i.e., groups may not be nested).

Note: `optgroup` is a recent addition to HTML, and may not yet be supported by all browsers. Be sure to test your pages thoroughly before relying on its implementation.

The `select` Element

Attributes for SELECT

`name` = *text*

The `name` attribute assigns the control name.

`size` = *integer*

If a `select` element is presented as a scrolled list box, the `size` attribute specifies the number of rows in the list that should be visible at the same time.

`multiple`

If set, this Boolean attribute allows multiple selections. If not set, the `select` element only permits single selections. The default value is single selection.

`disabled`

The `disabled` attribute is the same for `select` as it is for `input` (see page 8).

`tabindex`

The `tabindex` attribute is the same for `select` as it is for `input` (see page 8).

The following common attributes can also be used with `select` elements:

- `id`, `class` (identifiers)
- `lang` (language information), `dir` (text direction)
- `title` (element title)
- `style` (inline style information)
- `onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup` (events)

The `optgroup` Element

Attributes for `optgroup`

`label` = *text*

The `label` attribute specifies the label for the option group.

`disabled`

The `disabled` attribute is the same for `optgroup` as it is for `input` (see page 8).

The following common attributes can also be used with `optgroup` elements:

- `id`, `class` (identifiers)
- `lang` (language information), `dir` (text direction)
- `title` (element title)
- `style` (inline style information)
- `onfocus`, `onblur`, `onchange`, `onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup` (events)

The `option` Element

Attributes for `option`

`selected`

When set, this Boolean attribute specifies that this option is pre-selected.

Note: It is considered an error if more than one `option` element has the `selected` attribute set and the `select` element does not have the `multiple` attribute set.

`value` = *text*

The `value` attribute specifies the initial value of the control. If this attribute is not set, the initial value is set to the contents of the `option` element.

`label` = *text*

The `label` attribute allows you to specify a shorter label for an option than the content of the `option` element.

`disabled`

The `disabled` attribute is the same for `option` as it is for `input` (see page 8).

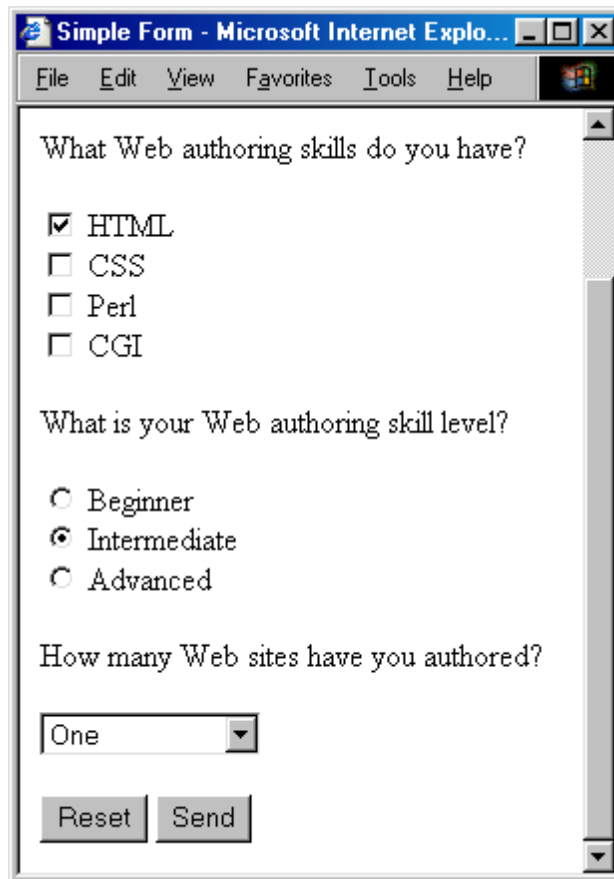
The following common attributes can also be used with `option` elements:

- `id`, `class` (identifiers)
- `lang` (language information), `dir` (text direction)

- title (element title)
- style (inline style information)
- onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup (events)

Exercises

Add a pop-up menu to your form:



The form with a pop-up menu added

```
<INPUT type="radio" name="level" value="advanced"> Advanced  
<P>  
How many Web sites have you authored?  
<P>  
<SELECT name="sites">  
<OPTION value="0">None  
<OPTION value="1" selected>One  
<OPTION value="2">Two  
<OPTION value="3+">Three or More  
</SELECT>
```

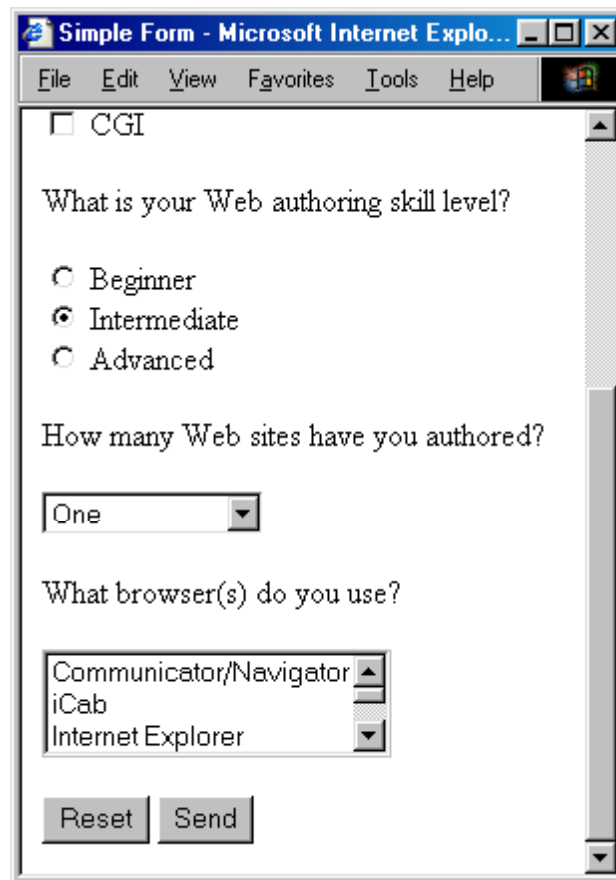
ACS Computer Training

Web Authoring: Forms

```
<P>  
<INPUT type="reset"> <INPUT type="submit" value="Send">
```

Add the highlighted text to **form.html**, save, and reload the page in your Web browser.

Next, add a scrolled list box to your form, using a `select` element with its `size` attribute set to a value greater than one. Allow multiple selections with the `multiple` attribute.



The form with a scrolled list box added

```
<OPTION value="3+">Three or More  
</SELECT>  
  
<P>  
What browser(s) do you use?  
  
<P>  
<SELECT name="browsers" size="3" multiple>  
<OPTION>Communicator/Navigator  
<OPTION>iCab  
<OPTION>Internet Explorer  
<OPTION>Lynx  
<OPTION>OmniWeb  
<OPTION>Opera
```

```
<OPTION>Other  
</SELECT>  
  
<P>  
<INPUT type="reset"> <INPUT type="submit" value="Send">
```

Add the highlighted text to **form.html**, save, and reload the page in your Web browser.

Notice that in the absence of a `value` attribute, the content of the selected `option` element becomes the value submitted by the `select` control.

The *textarea* Element

The `textarea` element creates a multi-line text input control. The contents of this element are its initial value.

Attributes for `textarea`

`name` = *text*

The `name` attribute assigns the control a name.

`rows` = *integer*

The `rows` attribute specifies the number of visible text lines. Users can enter more lines than this; the browser provides a scrolling mechanism to accommodate. This attribute is required.

`cols` = *integer*

The `cols` attribute specifies the visible width in average character widths. Users can enter longer lines than this; the browser provides a scrolling mechanism or wraps lines to accommodate. This attribute is required.

`readonly`

The `readonly` attribute is the same for `textarea` as it is for `input` (see page 8).

`disabled`

The `disabled` attribute is the same for `textarea` as it is for `input` (see page 8).

`tabindex`

The `tabindex` attribute is the same for `textarea` as it is for `input` (see page 8).

`accesskey`

The `accesskey` attribute is the same for `textarea` as it is for `input` (see page 8).

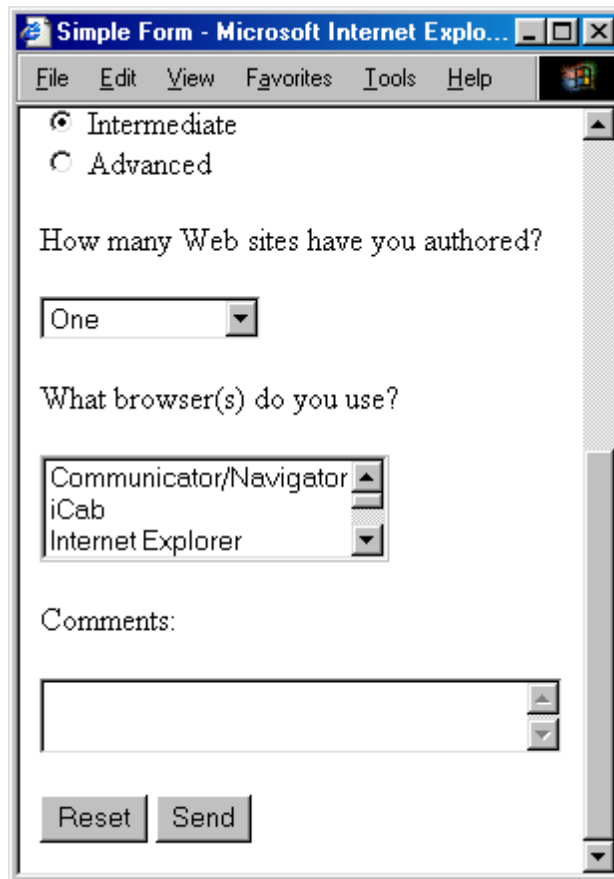
The following common attributes can also be used with `textarea` elements:

- `id`, `class` (identifiers)

- lang (language information), dir (text direction)
- title (element title)
- style (inline style information)
- onfocus, onblur, onselect, onchange, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup (events)

Exercise

Add a textarea element to your form:

A screenshot of a Microsoft Internet Explorer browser window titled "Simple Form - Microsoft Internet Explo...". The browser's menu bar includes "File", "Edit", "View", "Favorites", "Tools", and "Help". The page content features a form with the following elements: a radio button group with "Intermediate" selected and "Advanced" unselected; a text input field containing "One" with a dropdown arrow; a text input field containing "Communicator/Navigator", "iCab", and "Internet Explorer" with up and down arrows; a "Comments:" label above a new, empty text area with vertical scroll bars; and "Reset" and "Send" buttons at the bottom.

The form with a textarea control added

```
<OPTION>Other  
</SELECT>  
  
<P>  
Comments:  
  
<P>  
<TEXTAREA rows="2" cols="30" name="comments"></TEXTAREA>  
  
<P>
```

```
<INPUT type="reset"> <INPUT type="submit" value="Send">
```

Add the highlighted text to **form.html**, save, and reload the page in your Web browser.

Advanced Control Types

The preceding sections covered the basic control types most commonly used. Other form controls available in HTML are discussed below.

The `button` Element

Buttons created with the `button` element function just like buttons created with the `input` element, but they offer richer rendering possibilities: the `button` element may have content, including images and marked up text.

Note: It is illegal to associate an image map with an `img` that appears as the contents of a `button` element.

Attributes for `button`

`name = text`

The `name` attribute assigns the control name.

`value = text`

The `value` attribute assigns the initial value to the button.

`type = submit or reset or button`

The `type` attribute declares the type of the button. “Submit” creates a submit button, “reset” creates a reset button, and “button” creates a push button. The default value is “submit”.

`disabled`

The `disabled` attribute is the same for `button` as it is for `input` (see page 8).

`accesskey`

The `accesskey` attribute is the same for `button` as it is for `input` (see page 8).

`tabindex`

The `tabindex` attribute is the same for `button` as it is for `input` (see page 8).

The following common attributes can also be used with `button` elements:

- `id`, `class` (identifiers)
- `lang` (language information), `dir` (text direction)
- `title` (element title)
- `style` (inline style information)

- onfocus, onblur, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup (events)

Exercises

Replace the submit button with a graphical submit button, using an `img` element contained in a `button` element with `type="submit"`:



The screenshot shows a web browser window with the following form elements:

- Question: "How many Web sites have you authored?"
- Dropdown menu: "One"
- Question: "What browser(s) do you use?"
- Dropdown menu: "Communicator/Navigator", "iCab", "Internet Explorer"
- Text input field: "Comments:"
- Buttons: "Reset" and a graphical submit button featuring a globe image.

The form with a graphical submit button created with a button element of `type="submit"`

```
<INPUT type="reset">  
<BUTTON type="submit"><IMG  
src="http://www.ku.edu/acs/training/classes/wa-forms/globe.gif"  
alt="Submit">  
</BUTTON>  
</FORM>
```

Replace “`<INPUT type="submit" value="Send">`” with the highlighted text in **form.html**, save, and reload the page in your Web browser.

You can choose from <http://www.ku.edu/acs/training/classes/wa-forms/aquasubmit.gif>, <http://www.ku.edu/acs/training/classes/wa-forms/globe.gif>, or

<http://www.ku.edu/acs/training/classes/wa-forms/ku.gif> for the image source URL, or use any other that you know.

Another way to create a graphical submit button is with an `input` element of `type="image"`, as in the following example. With this method, mouse click coordinates within the image are added to the input value when the form is submitted, accommodating image maps. (Image maps are discussed in the class *Web Authoring: Tables, Frames, and Image Maps*.)

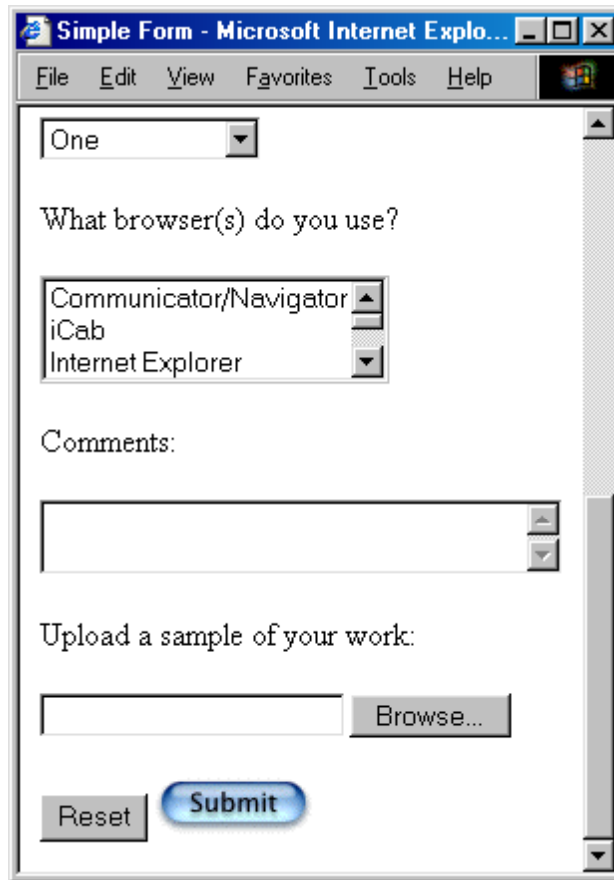


The form with a graphical submit button created with an `input` element of `type="image"`

```
<INPUT type="reset"> <INPUT type="image"  
src="http://www.ku.edu/acs/training/classes/wa-forms/aquasubmit.gif"  
alt="Submit">  
</FORM>
```

File Select Controls

A form can provide the means for a user to attach a file to the form submission, with an `input` of `type="file"`. Note that this type of input requires special handling on the part of the back-end script.



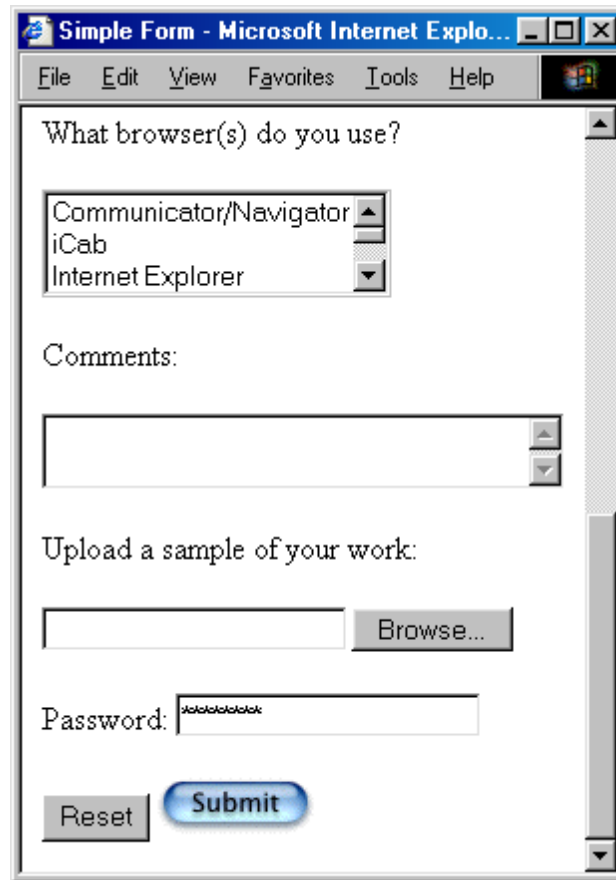
The form with a file select control added

```
<TEXTAREA rows="2" cols="30" name="comments"></TEXTAREA>
<P>
Upload a sample of your work:
<P>
<INPUT type="file" name="file">
<P>
<INPUT type="reset"> <INPUT type="image"
src="http://www.ku.edu/acs/training/classes/wa-forms/aquasubmit.gif"
alt="Submit">
```

Note that the entire control—including the text entry field for manually typing the file name and the Browse button for locating it through a standard open file dialog box—is generated by the browser in its rendering of the `input` element.

Password Entry Fields

An `input` with `type="password"` functions just like an ordinary text input, except the text entered is masked on-screen, typically by bullets or asterisks.



The form with a password entry field added

```
<INPUT type="file" name="file">  
<P>  
Password: <INPUT type="password" name="password">  
<P>  
<INPUT type="reset"> <INPUT type="image"  
src="http://www.ku.edu/acs/training/classes/wa-forms/aquasubmit.gif"  
alt="Submit">
```

Note that this does not encrypt or otherwise secure the transmission over the Internet of the information entered in any way.

Hidden Controls

An input element of type="hidden" can be used to embed information in the form data submission that does not appear in the form that the user views and completes. This technique is often useful for carrying data through a succession of script-generated forms.

```
Password: <INPUT type="password" name="password">  
<P>
```

```
<INPUT type="hidden" name="form" value="My form">  
  
<INPUT type="reset"> <INPUT type="image"  
src="http://www.ku.edu/acs/training/classes/wa-forms/aquasubmit.gif"  
alt="Submit">
```

Object Controls

HTML also supports the use of arbitrary objects, such as Java applets, as controls which set form values, using `input` elements of `type="object"`. Object controls are beyond the scope of this class.

Labels

Some form controls automatically have labels associated with them (e.g., buttons), while most do not (e.g., text fields, checkboxes, radio buttons, and menus).

For those controls that have implicit labels, browsers use the value of the `value` attribute as the label.

The `label` element is used to specify labels for controls that do not have implicit labels.

Note: `label` is a recent addition to HTML, and may not yet be supported by all browsers. Be sure to test your pages thoroughly before relying on its implementation.

The `label` Element

The `label` element can be used to attach information to controls. Each `label` element is associated with exactly one form control.

The `for` attribute associates a label with another control explicitly: the value of the `for` attribute must be the same as the value of the `id` attribute of the associated control element. (More than one `label` can be associated with the same control by creating multiple references via the `for` attribute.)

To associate a label with another control implicitly, the control element must be within the contents of the `label` element. In this case, the `label` may only contain one control element. The label itself may be positioned before or after the associated control.

Note that this technique cannot be used when a table is being used for layout, with the label in one cell and its associated control in another cell.

When a `label` element receives focus, it passes the focus on to its associated control.

Labels may be rendered by browsers in a number of ways (e.g., visually, read by speech synthesizers, etc.).

Attributes for `label`

`for` = *id reference*

The `for` attribute explicitly associates the label being defined with another control. When present, the value of this attribute must be the same as the value of

the `id` attribute of some other control in the same document. When absent, the label being defined is associated with the element's contents.

`accesskey`

The `accesskey` attribute is the same for `label` as it is for `input` (see page 8).

The following common attributes can also be used with `label` elements:

- `id`, `class` (identifiers)
- `lang` (language information), `dir` (text direction)
- `title` (element title)
- `style` (inline style information)
- `onfocus`, `onblur`, `onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup` (events)

Exercises

Add `label` tags to your form to formally define “Name: ” as the label for the name input, using implicit association:

```
<FORM
action="http://www.ku.edu/cgiwrap/acs/training/classes/wa-forms/forms.pl"
method="post">

<P>
<LABEL>Name: <INPUT type="text" name="name"></LABEL>

<P>
Email address: <INPUT type="text" name="address">
```

Next, define “Email address: ” as the label for the address input, using explicit association:

```
<LABEL>Name: <INPUT type="text" name="name"></LABEL>

<P>
<LABEL for="address">Email address: </LABEL><INPUT type="text"
name="address" id="address">

<P>
What Web authoring skills do you have?
```

In practice, you would extend one or the other of these techniques to all of your control labels.

Adding Structure to Forms: the `fieldset` and `legend` Elements

The `fieldset` element enables you to group thematically related controls and labels. Grouping controls makes it easier for users to understand their purpose while simultaneously facilitating tabbing navigation for visual browsers and speech navigation

for speech-oriented browsers. The proper use of this element makes documents more accessible.

The `legend` element enables you to assign a caption to a `fieldset`. The `legend` improves accessibility when the `fieldset` is rendered non-visually.

Note: `fieldset` and `legend` are recent additions to HTML, and may not yet be supported by all browsers. Be sure to test your pages thoroughly before relying on their implementation.

The fieldset Element

Attributes for fieldset

The following common attributes can be used with `fieldset` elements:

- `id`, `class` (identifiers)
- `lang` (language information), `dir` (text direction)
- `title` (element title)
- `style` (inline style information)
- `onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup` (events)

The legend Element

Attributes for legend

`accesskey`

The `accesskey` attribute is the same for `legend` as it is for `input` (see page 8).

The following common attributes can be used with `legend` elements:

- `id`, `class` (identifiers)
- `lang` (language information), `dir` (text direction)
- `title` (element title)
- `style` (inline style information)
- `onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup` (events)

Keyboard Access to Form Controls

Many of the control element descriptions above include the attributes `tabindex` and `accesskey`. As described for `input` on page 8, these are used to provide ways to select, or give *focus* to, form controls using the keyboard, thereby improving the accessibility of the form. The `tabindex` attribute designates a control's place in the tabbing order, used

when stepping through the controls by successively pressing the tab key. The `accesskey` attribute defines a keyboard shortcut for accessing a control.

Tabbing navigation and access keys are recent additions to HTML, and may not yet be supported in all browsers; furthermore, when they are supported, the actual key sequences used to activate these features may depend on the configuration of the browser. Be sure to test your pages thoroughly before relying on the implementation of these features.

Form Submission

When a form is submitted, the data collected are submitted to the server in a specific way, according to the attributes of the `form` element. The author of a form-processing script must be aware of the particulars of the manner in which the data will be received as program input in order to effectively process it.

When a form is submitted, the browser encodes the data and includes them in the HTTP message it sends to the server.

The default encoding is known as *application/x-www-form-urlencoded*, and corresponds to the way text is encoded in URLs, i.e., standard alphanumeric characters appear as-is, space characters are replaced by “+”, and other reserved characters and non-alphanumeric characters are replaced by percent signs followed by two-digit hexadecimal representations of their ASCII codes. (This encoding type is described in the technical standards document RFC1738, “Uniform Resource Locators”, section 2.2, available at <http://www.ietf.org/rfc/rfc1738.txt>.)

This encoding is applied to a listing of control names and values, ordered as they appear in the document, in which names are separated from values by “=” and name/value pairs are separated from each other by “&”. The control name is simply repeated for multiple control values, as might come from checkbox or menu controls.

Another encoding type, *multipart/form-data*, can be specified with the `enctype` attribute of the `form` element. It is often used for forms containing file upload controls, and includes all form data in a standard multipart MIME data stream like those used for email attachments. Information about this encoding can be found in the technical standards document RFC2045, “Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies”, available at <http://www.ietf.org/rfc/rfc2045.txt>.

Once encoded, the form data is submitted by the browser to the server as indicated in the `form` element’s `action` attribute, according to the specified `method`.

If the `method` is “get”, an ordinary request is sent to the server for the URL specified by `action`, with a question mark and then the encoded form data set appended to that URL.

If the `method` is “post”, as in the examples above, the browser sends the form data in an attachment to its HTTP message to the server.

These HTTP messages and the processing of encoded form data sets are further discussed in the class *Web Authoring: CGI Scripts*.

For More Information

Here is an additional source of information about HTML forms:

Online

- Forms section of the HTML 4.01 Specification
- <http://www.w3.org/TR/html4/interact/forms.html>

Getting Additional Help

ACS provides consulting and Q&A help in a variety of ways:

785/864-0200

question@ku.edu

www.ku.edu/acs/help

Last Update: 03/07/2003