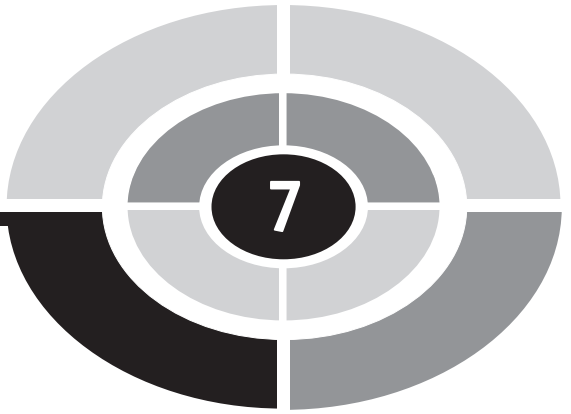

CHAPTER

Forms and Event Handling

It seems that no matter what web site you visit these days, you are asked to fill out a form—be it an order form, subscription form, membership form, financial form, survey, and the list goes on. Although forms may seem invasive, prying into our private affairs, forms are the only practical way to collect information that is necessary to conduct business on the Internet.

Forms are created using HTML form elements such as buttons and check boxes. Forms used by commercial web sites also interact by using JavaScript. A JavaScript is used for a variety of purposes, including data validation and for dynamically interacting with elements of a form.

In this chapter, you'll learn how to add another dimension to your HTML forms by writing JavaScripts that make an HTML form come alive.

Building Blocks of a Form

As you probably remember from when you learned HTML, a *form* is a section of an HTML document that contains *elements* such as radio buttons, text boxes, check boxes, and option lists. HTML form elements are also known as *controls*. Elements are used as an efficient way for a user to enter information into a form.

Forms are used for all kinds of purposes. In a business, forms are used to gather order information from a customer. Forms are also used for online surveys. Teachers use forms for online tests. Information entered into a form is sent to the web server for processing when the user clicks a submit button.

The program that processes the form is called a *Common Gateway Interface (CGI)* program. CGI programs are written in one of a number of programming languages, including JSP, PHP, Perl, and ASP. CGI programs typically interact with non-web applications such as databases and other systems that are necessary to process the form. Once processing is completed, the CGI program usually creates another web page dynamically and sends the web page to the browser that sent the form.

Elements and JavaScript

Each element has one or more attributes, which is information associated with the element. For example, the `value` attribute is used to define a default value, not the user-entered value. A good example would be the `name` attribute, since this attribute is used to reference the element. You'll learn about the different kinds of attributes that are available for each element throughout this chapter as each element is discussed.

Many applications require that some information contained on a form be verified using a validation process. Two common ways to validate information on a form are by using CGI programs and JavaScripts. A CGI program validates information after the form is submitted. A JavaScript can validate information whenever one of several events occurs while the form is displayed on the screen. You'll learn about these events in the "Responding to Form Events" section of this chapter.

Validation should occur on both the client (via JavaScript) and the server (via a CGI program). The client-side validation provides immediate feedback and reduces load on the server. It's good practice to validate again on the server because you don't always know that the JavaScript executed properly on the browser. You could make an exception to this if you require that JavaScript be enabled in order to use a web site (but that's more of a business decision).

In addition to validating information, JavaScripts can dynamically change a form while the form is displayed on the screen. For example, a JavaScript can activate or deactivate elements based on a value the user enters into another element. You can

also set the default value of elements based on a value entered by a user into another element.

A JavaScript can interact with elements of a form in many ways. You'll learn about them in this chapter. However, you won't learn about creating a form here; instead, you'll see examples of forms that are used to illustrate JavaScripts. Pick up a copy of *HTML: The Complete Reference, Third Edition* by Thomas A. Powell or *How to Do Everything with HTML* by James H. Pence (both books published by McGraw-Hill/Osborne) if you need to brush up on how to create forms.

Responding to Form Events

A JavaScript executes in response to an event that occurs while a form is displayed on the screen. An event is something the user does to the form, such as clicking a button, selecting a radio button, or moving the cursor away from an element on the form. The browser also fires events when the page finishes loading from the server. You can execute a script each time one of the form events listed in Table 7-1 occurs.

An event is associated with an element of a form as a attribute defined within the opening tag of the element. You assign this attribute the name of the JavaScript function that you want executed when the event occurs.

Let's say that your form has an input element in which the user enters his or her first and last names and e-mail address (Figure 7-1). You want a JavaScript to validate the e-mail address by checking whether the address includes an @ sign when the user moves the cursor away from the input element. You do this by using the `onblur` event attribute in the opening `<INPUT>` element tag and assigning the name of the JavaScript function to the `onblur` event attribute. The `onblur` event occurs when the cursor moves away from the element, which is called *losing focus*. The following example illustrates how this is done.

```
<!DOCTYPE html PUBLIC
      "-//W3C//DTD XHTML 1.0 Transitional//EN">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
   <title>onblur event</title>
      <script language="Javascript" type="text/javascript">
         <!--
         function ValidateEmail(EmailAddress)
         {
            var Location = EmailAddress.indexOf('@')
            if ( Location == -1)
            {
              alert
```

```
                ('You entered an inaccurate email address.')
            }
        }
        -->
    </script>

</head>
   <body>
      <FORM action="http://www.jimkeogh.com" method="post">
         <P>
         First Name: <INPUT type="text" name="Fname"/><BR>
         Last Name: <INPUT type="text" name="Lname"/><BR>
         Email: <INPUT type="text" name="Email"
             onblur="ValidateEmail (this.value)"/><BR>
         <INPUT name="Submit" value="Submit" type="submit"/>
         <INPUT name="Reset" value="Reset" type="reset"/>
         </P>
     </FORM>
   </body>
</html>
```

Event	Description
onload	Executes when the browser finishes loading a window or all frames within a frameset
onunload	Executes when the browser removes a document from a window or frame
onclick	Executes when the mouse button is clicked over an element
ondblclick	Executes when the mouse button is double-clicked over an element
onmousedown	Executes when the mouse button is clicked while the mouse cursor is over an element
onmouseup	Executes when the mouse button is released while the mouse cursor is over an element
onmouseover	Executes when the mouse cursor moves onto an element
onmousemove	Executes when the mouse cursor is moved while over an element
onmouseout	Executes when the mouse cursor is moved away from an element
onfocus	Executes when an element receives focus
onblur	Executes when an element loses focus
onkeypress	Executes when a key is pressed and released
onkeydown	Executes when a key is held down

Table 7-1 Form Events

Event	Description
`onkeyup`	Executes when a key is released
`onsubmit`	Executes when a form is submitted
`onreset`	Executes when a form is reset
`onselect`	Executes when text is selected in a text field
`onchange`	Executes when an element loses input focus and the value of the element has changed since gaining focus

Table 7-1 Form Events *(continued)*

You'll see a form displayed when you call this web page from your browser. The form has five elements: The first two elements are input elements for the first and last names. The third element is also an input element, where the user enters an e-mail address. The last two elements are buttons—the Submit button that submits the form to the web server, and the Reset button that clears data from the form. Notice that each element has a `name` attribute, which is assigned a unique name. The `name` attribute can be referred to in a JavaScript, although we don't refer to the `name` attribute in this example.

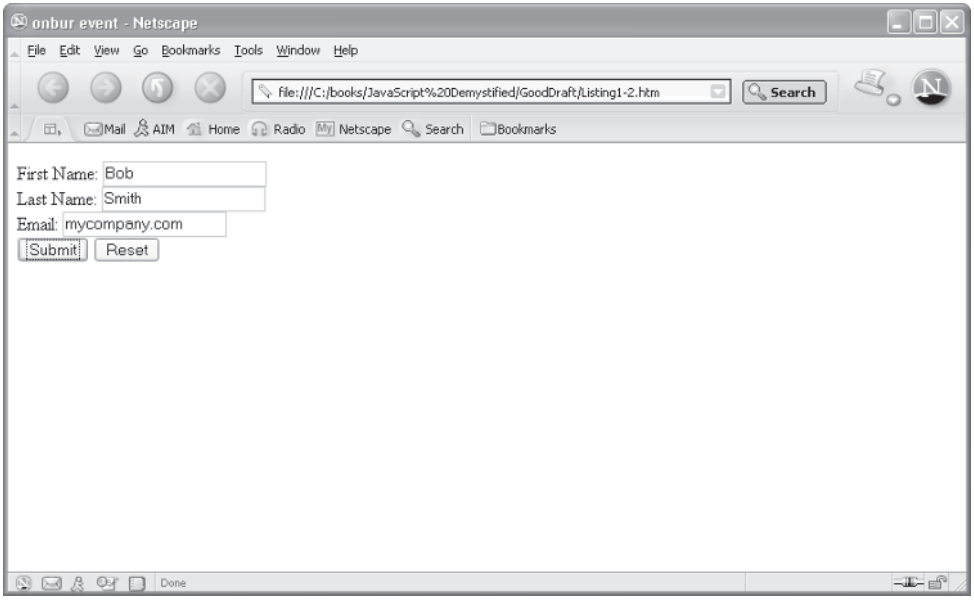

Figure 7-1 The form prompts the user to enter his or her first and last names and an e-mail address.

Take a look at the Email element and you'll notice that we've included the onblur event in the INPUT open tag. We also assigned to it the name of the function that we want called whenever the user moves the cursor from the Email input element. This is called ValidateEmail() and is defined in the JavaScript located in the <head> section of this web page script. The ValidateEmail() function is passed one parameter, which is the value of the Email input element. This parameter might look a little strange, but you've seen something like this used in previous chapters. You'll recall that whenever you want to write something on the screen, you execute the following statement:

```
document.write('Display this text.')
```

Here, the name of an object is document, and write() is the name of the method that is associated with the document object. This is basically the same thing as the parameter that is being passed to the ValidateEmail() function.

In the onblur event code, the name of the object is called this—that is, this refers to the current object, which is the Email input element. It is like saying, "The color of this car is blue." It is assumed that everyone knows which car you're talking about, because it is the only car that you're looking at. Therefore, use the word *this* whenever you want to refer to the name of the current object.

Notice that value is the attribute associated with the this object (the Email input element). Whenever you use the name of an attribute such as value, you are telling the browser to use the value of the attribute. In this case, we're telling the browser to use the value of the value attribute, which is the information the user enters into the Email input element.

Suppose the user enters *jkeogh@mcgrawhill.com* into the Email input element on this form. In this case, the this.value is the same as jkeogh@mcgrawhill .com, because the e-mail address is the value assigned to the value attribute by the browser when the user enters the address into the Email input element on the form.

Let's take a look at the ValidateEmail() function definition in the JavaScript within the <head> portion of the web page. The e-mail address passed to the ValidateEmail() function is assigned to EmailAddress. The first statement within the function declares a variable called Location and initializes the variable with the index of the @ symbol within EmailAddress.

You'll recall from Chapter 6 that the indexOf() function finds the position of a character within a string of characters. The indexOf() function returns a –1 if the string doesn't contain the character. The value of the Location variable will either be –1, if the @ symbol isn't in the EmailAddress, or an index value, which means there is a good chance that the e-mail address is in the proper format. (We won't know whether it is a valid e-mail address until we try sending an e-mail to that address.)

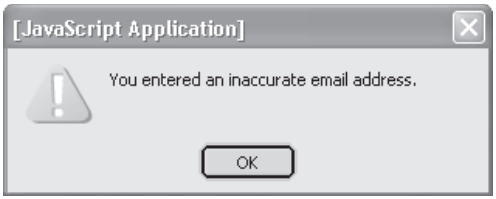

Figure 7-2 A warning message is displayed if the @ symbol was not entered in the e-mail address.

We're only interested if the value of the `Location` variable is –1. Therefore, we use an if statement (see Chapter 3) to determine whether the @ symbol *wasn't* entered by the person. If the @ symbol was not entered, an alert dialog box is displayed with a message warning that the e-mail address is invalid (Figure 7-2).

Form Objects and Elements

When you look at a form within a web page, you probably don't necessarily think about how the form relates to everything else that you're seeing. However, relationships on a web page are very important when you are a JavaScript programmer, because you need to know them to access them.

Everything that you see on a web site is considered an object. The first object you see is the window, which is referred to in a JavaScript as `window`. A window contains an HTML document referred to as `document`. You've referenced the document throughout this book whenever you called the `document.write()` function. A document can have one or more forms, and a form can have one or more elements.

Form objects are stored in an array (see Chapter 4) called *forms* and appear in the order in which the forms appear in the document. You can reference each form by referencing the form's index. Suppose you wanted to reference the third form. You'd write this:

```
window.document.forms[2]
```

You're telling the browser to go to the `window` object and then within the `window` object go to the `document` object and then reference the form that is assigned to the 2 index value of the forms array. (Remember that the index 2 is referencing the third form, because the first form is index 0.)

TIP *Although this is a good syntax to reference the window object, this is not required. You can use this instead:*

```
document.forms[2] (might be worth mentioning...)
```

Forms are assigned to elements of the forms index in the order that each form appears in the document. You can reference a form using its index instead of using the name of the form. Remember that the name of the form is the value that is assigned to the form's `name` attribute. Here's how to reference a form by using the name of the form. In this example, we're referencing the order form:

```
window.document.forms.order
```

TIP *Referencing by name is better practice than referencing by index because the display and ordering of elements changes all the time, and it requires less maintenance if you reference by name. Also, referencing by name makes your code easier for humans to understand and maintain.*

The following example shows how to access an attribute of a form. We defined the `display()` function in the JavaScript within the `<head>` tag. This function receives the value of the form's `Reset` element and displays it in an alert dialog box (Figure 7-3). The function is called in response to an `onclick` event that occurs when the user clicks the Reset button.

```
<!DOCTYPE html PUBLIC
     "-//W3C//DTD XHTML 1.0 Transitional//EN">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
   <title>Accessing form attributes</title>
      <script language="Javascript" type="text/javascript">
         <!--
         function display()
         {
          alert
              ('Value: ' + document.forms.order.Reset.value)
         }
         -->
      </script>
</head>
   <body>
      <FORM action=
       "http://www.jimkeogh.com" method="post" name="order">
         <P>
         First Name: <INPUT type="text" name="Fname"/><BR>
         Last Name: <INPUT type="text" name="Lname"/><BR>
```

```
          Email: <INPUT type="text" name="Email"/><BR>
          <INPUT name="Submit" value="Submit" type="submit"/>
          <INPUT name="Reset" value="Reset"
            type="reset" onclick="display()"/>
          </P>
      </FORM>
   </body>
</html>
```

Elements on a form are stored in an array called *elements* in the order in which the elements appear on the form. Here's how you access an element by using the element's index within the elements array:

```
window.document.forms.order.elements[2]
```

This tells the browser to go to the `window` object and within the `window` object go to the `document` object. Within the `document` object go to the `forms` and access the form named `order`. And within the `order` form access the element that has index 2, which is the third element.

Time-Saving Shortcut

Here's a trick JavaScript pros use to reduce the amount of typing they have to do when referencing attributes of elements. Let's say that you want to access the `value` attribute of the `email` element. You'd write the following:

```
window.document.forms.order.email.value
```

Suppose you want to access several attributes of the `email` element. Instead of writing the full path, you can use a with statement to save keystrokes when writing your JavaScript. Here's the shortcut:

```
with(window.document.forms.order.email)
{
   alert('Email: ' + value)
}
```

Figure 7-3 The alert dialog box displays an attribute of an element when the Reset button is clicked.

In this example, the full path is written once at the top of the with statement and is then automatically applied to each attribute within the with statement. You can use this same technique to create elements of a form, like so:

```
with(window.document.forms.order)
{
   alert('Email: ' + mail.value)
}
```

You can write other statements in the with statement to reference other elements of the order form without having to write the complete path.

Changing Attribute Values Dynamically

You can spice up any form by changing the attributes of the form element dynamically. Let's say that your user/customer wants to modify an existing order. Your application displays the order form, and then prompts the customer to make changes. You could highlight those changes by altering the color, style, or font of the element after the customer makes the change. This gives the customer a visible way of telling what information has changed.

You can change an attribute of an element by assigning a new value to the attribute from within a JavaScript function. The function is then called when an appropriate event occurs. In the next example, we'll display the form you saw in a previous example that enables the user to enter a first and last name and an e-mail address. This example displays default values for these elements just as if existing contact information were recalled from a file. Whenever the user changes the default value, we'll display the new value in blue instead of black and change the background color from white to silver.

Here's how this is done. First, we define a function in the `<head>` tag called `Highlight()`. This function receives one parameter, which is the name of the element that calls the function. The name is compared with the names of each element on the form. When a match occurs, statements within the if statement change the text color and background color style attributes of the element by assigning a new value to the style of the element (Figure 7-4).

Notice that the `Fname` element, `Lname` element, and `Email` element trap the `onchange` event. The `onchange` event occurs when the cursor is moved away from the element (that is, it loses input focus) and the value of the element has changed since the last time the cursor was placed on the element (that is, it gained focused). The `onchange` event happens when the user changes the element and then moves on to another element. When the `onchange` event occurs, the

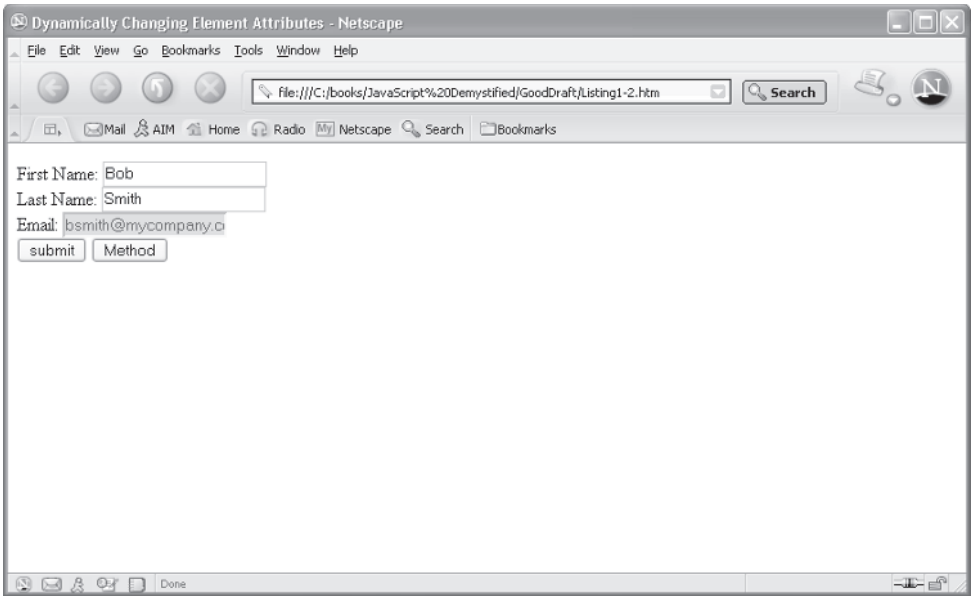

Figure 7-4 The color and background color of an element is changed after a user changes the value of the element.

`Highlight()` function is called and is passed the name of the element. This would also be rather maintenance-intensive. Instead, you can pass in the element itself so the function is more generic.

```
<!DOCTYPE html PUBLIC
         "-//W3C//DTD XHTML 1.0 Transitional//EN">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
   <title>Dynamically Changing Element Attributes</title>
      <script language="Javascript" type="text/javascript">
         <!--
         function Highlight(Element)
         {
            Element.style.color = 'blue'
            Element.style.backgroundColor = 'silver'
         }
         -->
        </script>
   </head>
   <body>
      <FORM name="Contact"
```

```
        action="http://www.jimkeogh.com" method="post">
      <P>
        First Name: <INPUT value="Bob" type="text"
          name="Fname" onchange="Highlight(this)"/><BR>
        Last Name: <INPUT value="Smith" type="text"
          name="Lname" onchange="Highlight(this)"/><BR>
        Email: <INPUT value="bsmith@mcgrawhill.com" type="text"
           name="Email" onchange="Highlight(this)"/><BR>
        <INPUT name="Submit" value="submit" type="submit"/>
        <INPUT name="Reset" value="Method" type="reset"/>
      </P>
      </FORM>
   </body>
</html>
```

Changing Elements Based on a Value Selected by the User

Another way you can jazz up your form is to fill in information automatically based on information already entered into the form. You do this by assigning a new value to the `value` attribute of an element after the user changes another element on the form.

Here's how this works. Suppose you want to fill in the e-mail address on the form automatically, based on a user's first and last names as entered in the form. In this example, the e-mail address will consist of the first initial of the user's first name and the full last name, as entered by the user. So Mary Jones's e-mail address would look like this: mjones@mycompany.com.

The next example traps the `onchange` event for both the `Fname` and `Lname` elements and calls the `SetEmail()` function, which is defined in the `<head>` tag section of the document. The `SetEmail()` function determines whether a first and last name were entered into the form by examining the `length` attribute of the string, which you learned about in Chapter 6. If the `length` is greater than zero, we assume that the user entered a first name or last name. Both names must be entered; otherwise, the function doesn't set the e-mail address because the e-mail address requires both the first and last names.

However, if both names exist, the function copies the first letter of the first name using the `charAt()` function. As you'll recall from Chapter 6, each character of a string is assigned as an element of an array. The first element has an index of 0. The `charAt()` function is told to return the character at index 0, which is the first letter of the value of the first name.

The domain name is then concatenated to the value of the last name, and the value of the last name is concatenated to the first letter of the first name to form

the e-mail address. The e-mail address is then assigned to the value of the `Email` element (Figure 7-5).

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
   <title>Dynamically Change Attribute Value</title>
      <script language="Javascript" type="text/javascript">
         <!--
         function SetEmail()
         {
            with (document.forms.Contact)
            {
               if (Fname.value.length >0
                 && Lname.value.length >0)
               {
                  Email.value =
                    Fname.value.charAt(0) +
                     Lname.value + '@mycompany.com'
               }
            }
         }
         -->
      </script>
</head>
   <body>
      <FORM name=
               "Contact" action="http://www.jimkeogh.com"
                  method="post">
         <P>
         First Name: <INPUT type="text" name="Fname"
                onchange="SetEmail()"/><BR>
         Last Name: <INPUT type="text" name="Lname"
             onchange="SetEmail()"/><BR>
         Email: <INPUT type="text" name="Email"><BR>
         <INPUT name="Submit" value="Submit" type="submit"/>
         <INPUT name="Reset" value="Reset" type="reset">
        </P>
     </FORM>
   </body>
</html>
```

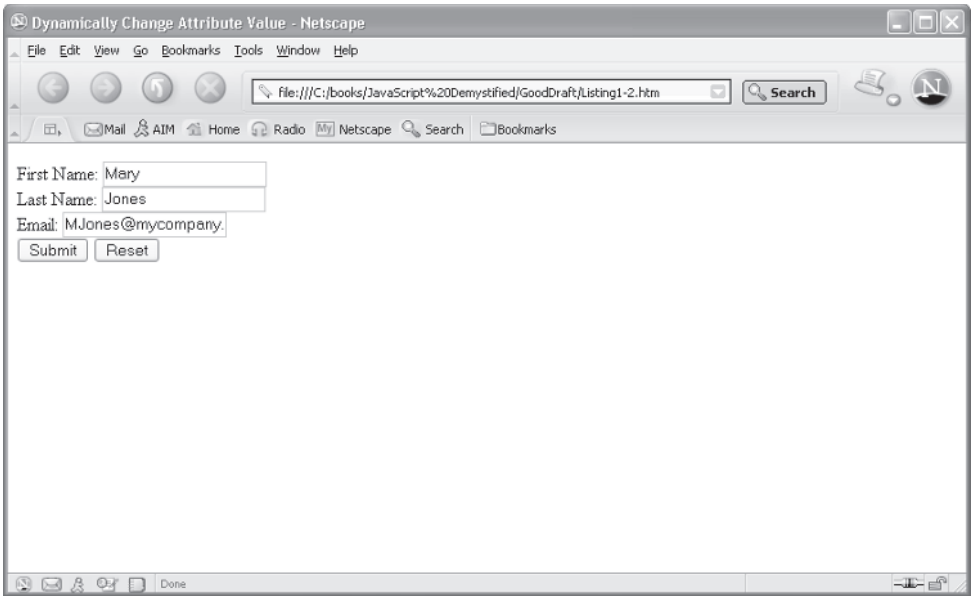

Figure 7-5 The JavaScript automatically fills in the e-mail address when the user enters a first and last name.

Changing an Option List Dynamically

As you'll recall, an option list presents a user with two or more items from which to choose. Items that appear on the option list are typically set when the option list is created. However, you can change the content of an option list on the fly by using a JavaScript function.

Let's say that you want to give the user the option of selecting either a car or a motorcycle, but not both. One way to do this is to display two radio buttons called Cars and Motorcycles. When one radio button is selected, the other radio button is automatically deselected. In other words, when the Cars radio button is selected, the Motorcycles radio button will be deselected because the two radio buttons are part of the same form and have the same value for the `name` attribute.

To wow the user, you can change items in an option list to reflect whatever radio button the user selects. That is, the option list shows cars when the Cars radio button is selected and the same option list shows motorcycles when the Motorcycles radio button is selected. You can dynamically change items in an option list by calling a JavaScript function whenever the radio button selection changes. The function then resets items on the option list.

The following example shows how this works. Take a look at the form and you'll notice an option list that contains two models of motorcycles. Beneath the

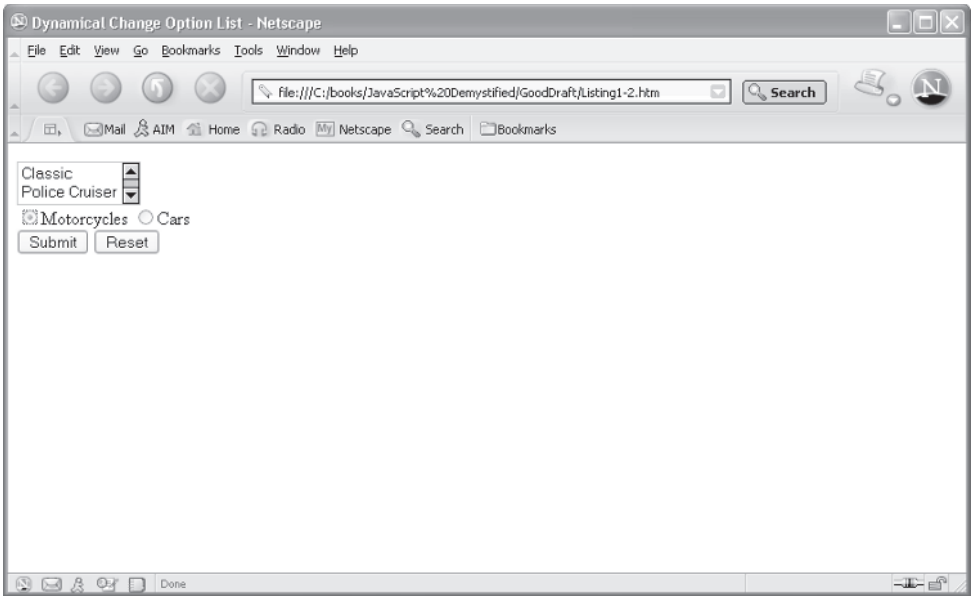

Figure 7-6 Items on the option list change based on the radio button selected by the user on the form.

option list are two radio buttons: Motorcycles and Cars. The Motorcycles radio button is selected by default. Each radio button responds to the `onclick` event by calling the `ResetOptionList()` function, passing it the value of the radio button.

You'll notice that the `ResetOptionList()` function is defined in the `<head>` tag section of the page. The value of the radio button selected is assigned to the `ElementValue` parameter of the `ResetOptionList()` function. Based on this value, the `ResetOptionList()` function resets the text and the value of items on the option list to reflect the radio button that the user selected (Figure 7-6). Notice that each item on the option list has a unique value; this enables the CGI application to determine which option was selected.

```
<!DOCTYPE html PUBLIC
    "-//W3C//DTD XHTML 1.0 Transitional//EN">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
   <title>Dynamically Change Option List</title>
      <script language="Javascript" type="text/javascript">
         <!--
         function ResetOptionList(ElementValue)
         {
            with (document.forms.Contact)
```

```
            {
               if (ElementValue == 1)
               {
                 OptionList [0].text = "Classic"
                 OptionList [0].value = 1
                 OptionList [1].text = "Police Cruiser"
                 OptionList [1].value = 2
               }
               if (ElementValue == 2)
               {
                 OptionList [0].text = "Ford"
                 OptionList [0].value = 1
                 OptionList [1].text = "Chevy"
                 OptionList [1].value = 2
               }
            }
         }
         -->
      </script>
</head>
   <body>
      <FORM name="Contact"
            action="http://www.jimkeogh.com" method="post">
         <P>
          <select name="OptionList" size="2">
              <option Value=1>Classic
              <option Value=2>Police Cruiser
              </select>
          <BR>
          <INPUT TYPE="radio"
               name="vehicles" checked="true"
               value=1 onclick ="
                ResetOptionList(this.value)">Motorcycles
          <INPUT TYPE="radio"
               name="vehicles" Value=2 onclick="
                ResetOptionList(this.value)">Cars
          <BR>
          <INPUT name="Submit" value="Submit" type="submit"/>
          <INPUT name="Reset" value="Reset" type="reset">
        </P>
      </FORM>
   </body>
</html>
```

Evaluating Check Box Selections

A check box is a common element found on many forms and is used to enable a user to select one or more items from a set of known items. You can write a Java Script function that evaluates whether or not a check box was selected and then processes the result according to the needs of your application.

You'll see how this is done in the next example, where the user is prompted to select his or her level of education using check boxes. Each check box item displays a level of education, and this information is processed when the user clicks the Process button at the bottom of the form (Figure 7-7).

The Process button traps the `onclick` event and calls the JavaScript `Education()` function, which is defined in the `<head>` tag section of this page. The `Education()` function evaluates each check box to determine whether the item is checked and then displays the user's education in an alert dialog box (Figure 7-8).

The `Education()` function begins by declaring a string and initializing it with the first part of the text that will appear in the alert dialog box. It then evaluates the checked attribute of each check box. If the checked attribute is true, the level of education is concatenated to the string. You'll notice that the += operator is used. As you'll recall from Chapter 2, this operator concatenates the value to the

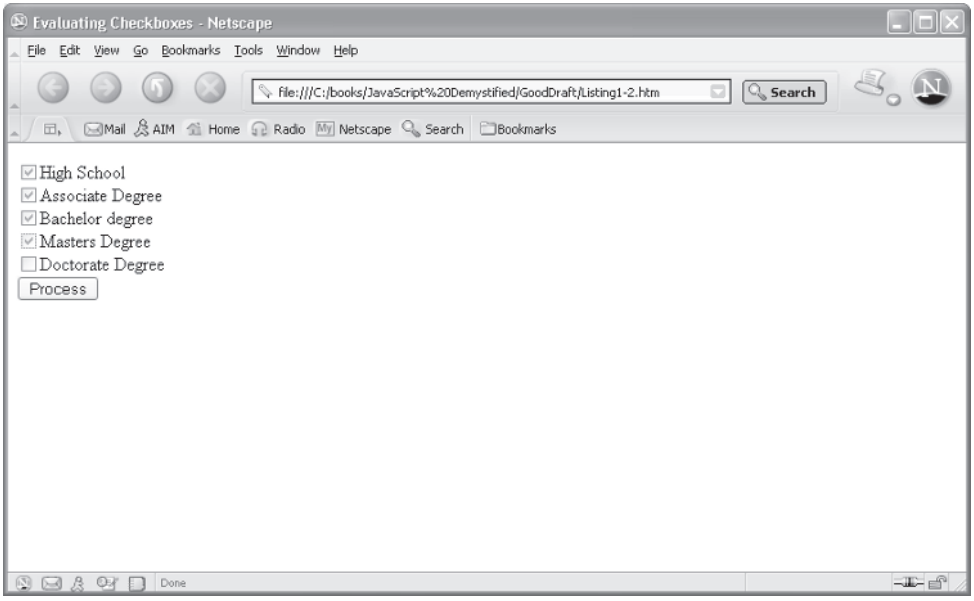

Figure 7-7 A JavaScript function can evaluate choices made using a check box or other elements on a form.

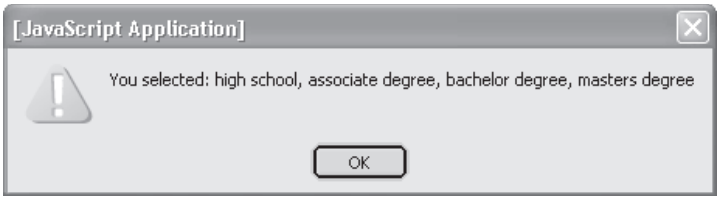

Figure 7-8 The Education() function displays check box selections.

right (level of education) of the operator to the value to the left of the operator (value of the selection variable) and then assigns the concatenated strings to the `selection` variable.

```
<!DOCTYPE html PUBLIC
        "-//W3C//DTD XHTML 1.0 Transitional//EN">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
   <title>Evaluating Checkboxes</title>
      <script language="Javascript" type="text/javascript">
         <!--
         function Education()
         {
            var selection = "You selected: "
            with (document.forms.Contact)
            {
               if (HS.checked == true)
               {
                 selection += "high school"
               }
               if (AD.checked == true)
               {
                 selection += ", associate degree"
               }
               if (BD.checked == true)
               {
                 selection += ", bachelor degree"
               }
               if (MD.checked == true)
               {
                 selection += ", masters degree"
               }
               if (DD.checked == true)
               {
```

```
                selection += ", doctorate degree "
              }
            }
            alert(selection)
          }
          -->
      </script>
</head>
   <body>
      <FORM name="Contact"
          action="http://www.jimkeogh.com" method="post">
          <P>
          <INPUT TYPE="checkbox"
             name="HS" value="HS">High School
          <BR>
          <INPUT TYPE="checkbox"
              name="AD" value="AD">Associate Degree
          <BR>
          <INPUT TYPE="checkbox"
             name="BD" value="BD">Bachelor degree
          <BR>
          <INPUT TYPE="checkbox"
             name="MD" value="MD">Masters Degree
          <BR>
          <INPUT TYPE="checkbox"
             name="DD" value="DD">Doctorate Degree
          <BR>
          <INPUT name="Process" value="Process"
               type=reset onclick ="Education()"  >
        </P>
      </FORM>
   </body>
</html>
```

Manipulating Elements Before the Form Is Submitted

You can manipulate elements on a form after the user clicks the Submit button and before the form is actually submitted to the CGI application. This is handy if you need to validate information on the form or want to amend information to the form that the user didn't enter.

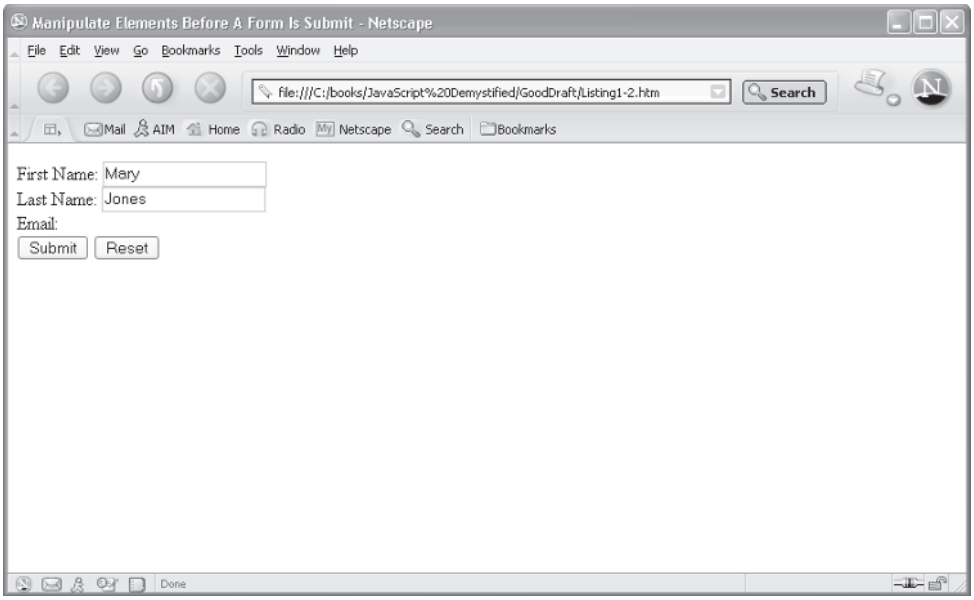

Figure 7-9 The SetEmail() function creates the e-mail address and assigns it to the Email element before the form is submitted for process.

You do this by assigning a JavaScript function to the `onsubmit` event. You'll see how this is done in the next example, where the e-mail address is automatically entered into the form after the user submits the form for processing.

This form is similar to other forms you've seen in this chapter, except the Email element is a hidden element (Figure 7-9). You probably remember from the time you learned HTML that a *hidden element* is like any other element on a form, except the element doesn't appear on the screen. A hidden element has a name and value that is sent to the CGI program along with other elements of the form for processing.

When the Submit button is clicked, the `SetEmail()` function is called. The `SetEmail()` function creates an e-mail address using the user's first and last names. The function then assigns the e-mail address to the value of the Email element, and the form is submitted to the CGI program.

```
<!DOCTYPE html PUBLIC
        "-//W3C//DTD XHTML 1.0 Transitional//EN">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
   <title>Manipulate Elements Before A Form Is Submitted</title>
      <script language="Javascript" type="text/javascript">
```

```
         <!--
         function SetEmail()
         {
            with (document.forms.Contact)
            {
               if (Fname.value.length >0 &&
                        Lname.value.length >0)
               {
                  Email.value = Fname.value.charAt(0)
                     + Lname.value + '@mycompany.com'
               }
            }
         }
         -->
      </script>
</head>
   <body>
      <FORM name="Contact"
          action="http://www.jimkeogh.com" method="post">
         <P>
         First Name: <INPUT type="text" name="Fname"/> <BR>
         Last Name: <INPUT type="text" name="Lname"/><BR>
         Email: <INPUT type="hidden" name="Email"/><BR>
         <INPUT name="Submit" value="Submit"
               type="submit" onsubmit="SetEmail()"/>
         <INPUT name="Reset" value="Reset" type="reset">
        </P>
     </FORM>
   </body>
</html>
```

Using Intrinsic JavaScript Functions

JavaScript has a special set of functions called *intrinsic* functions that mimic actions of the Submit button and Reset button of a form. You don't define an intrinsic function, because JavaScript defines the function for you. However, you can call an intrinsic function in the same way you would if you had defined the function.

An intrinsic function is often used to replace the Submit button and the Reset button with your own graphical images, which are displayed on a form in place of these buttons. This is illustrated in the next example. Two `<img>` (image) tags are used: one to display mysubmit.gif and the other to display myreset.gif. Notice that

each of these traps the `onclick` event and calls the appropriate intrinsic function. This has the same effect as inserting the Submit and Reset buttons on the form and then clicking them.

You can do this as follows:

```
<input type="image" src="mysubmit.gif"/>
```

The intrinsic functions would usually be called from the JavaScript function.

```
<!DOCTYPE html PUBLIC
            "-//W3C//DTD XHTML 1.0 Transitional//EN">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
   <title>Using Intrinsic JavaScript Functions</title>
</head>
   <body>
      <FORM name="Contact"
              action="http://www.jimkeogh.com" method="post">
         <P>
         First Name: <INPUT type="text" name="Fname"/> <BR>
         Last Name: <INPUT type="text" name="Lname"/><BR>
         Email: <INPUT type="text" name="Email"/><BR>
         <img src="mysubmit.gif"
         onclick="javascript:document.forms.Contact.submit()"/>
         <img src="myreset.gif"
         onclick="javascript:document.forms.Contact.reset()"/>
        </P>
     </FORM>
   </body>
</html>
```

Changing Labels Dynamically

You can avoid cluttering a form with elements by relabeling an element when its purpose has already been served. Think of this a reusing an element. You can re-label an element and change any of its attributes by using a JavaScript function.

Let's see how this in done. The next example is similar to the example used earlier in the chapter for changing an option list dynamically. Here, it displays an option list that contains either motorcycles or cars, depending on the category that the user selects. In the earlier example, radio buttons were used. The appropriate option list was displayed depending on which radio button the user selected. In this example, the user clicks a button to change the option list.

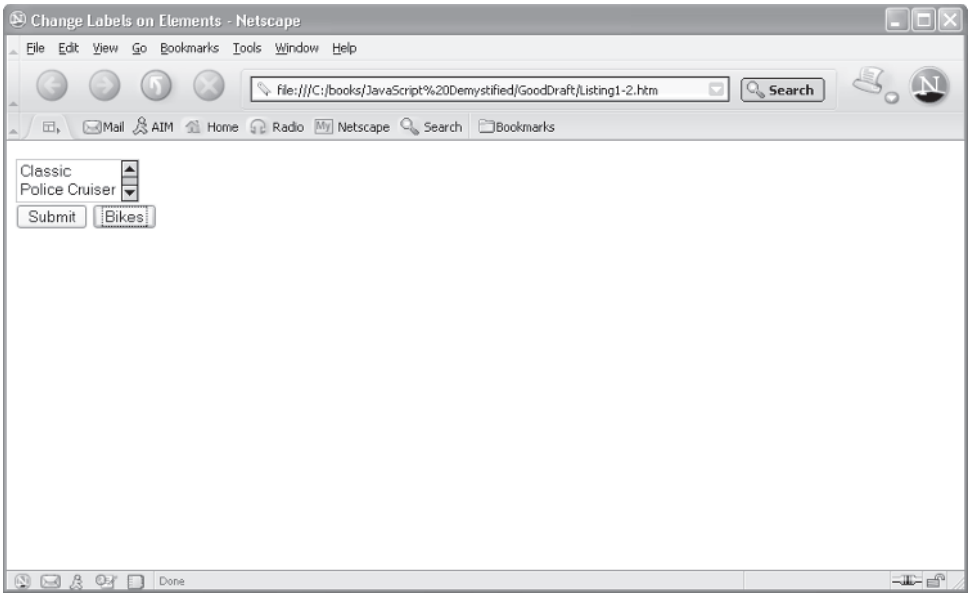

Figure 7-10 You can use a JavaScript function to change a label on an element such as a button while the form is being used.

The option list consists of motorcycles, and the button is labeled *Cars* when the form is displayed. The user changes the option list to show cars by clicking the Cars button. This causes the button to be relabeled as *Bikes*. When the Bikes button is clicked, the option list shows motorcycles again and the button is relabeled *Cars*.

The button click traps the `onclick` event and calls the `ResetOptionList()` function, passing the function the value of the button. The `ResetOptionList()` function compares the value with the two possible values, Cars and Bikes, and then resets the text and value attributes of each option and resets the value of the button (Figure 7-10). The *value* is the button label.

```
<!DOCTYPE html PUBLIC
       "-//W3C//DTD XHTML 1.0 Transitional//EN">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
   <title>Change Labels on Elements</title>
      <script language="Javascript" type="text/javascript">
         <!--
         function ResetOptionList(ElementValue)
         {
            with (document.forms.Contact)
            {
```

```
                if (ElementValue == 'Cars')
                {
                  SwitchButton.value = 'Bikes'
                  OptionList [0].text = 'Classic'
                  OptionList [0].value = 1
                  OptionList [1].text = 'Police Cruiser'
                  OptionList [1].value = 2
                }
                if (ElementValue == 'Bikes')
                {
                  SwitchButton.value = 'Cars'
                  OptionList [0].text = 'Ford'
                  OptionList [0].value = 1
                  OptionList [1].text = 'Chevy'
                  OptionList [1].value = 2
                }
             }
          }
          -->
      </script>
</head>
   <body>
      <FORM name="Contact"
              action="http://www.jimkeogh.com" method="post">
         <P>
          <select name="OptionList" size="2">
              <option Value=1>Classic
              <option Value=2>Police Cruiser
          </select>
          <BR>
          <INPUT name="Submit"
              value="Submit" type="submit"/>
          <INPUT name="SwitchButton" value="Bikes" type="reset"
              onclick="ResetOptionList(this.value)" >
        </P>
      </FORM>
   </body>
</html>
```

Disabling Elements

It is common to display a form with some elements disabled, which prevents the user from entering information into the element. A disabled element appears on the form, but no information can be entered into the element until it is enabled, usually after required information is entered into another element on the form.

You can use a JavaScript function to disable and enable elements on the form. This is shown in the next example. Notice that the `Email` element is disabled (Figure 7-11). It doesn't become enabled until the user enters both a first and last name, since the e-mail address is composed of both names in this case.

An element is disabled and enabled by setting the value of the `disabled` attribute. Initially, the `disabled` attribute of the `Email` element is set to true, which means that the `Email` element is disabled. Each time there is a change to the first and or last name elements, the `EnableEmail()` function is called, which examines the content of the `Fname` and `Lname` elements. If a value has been

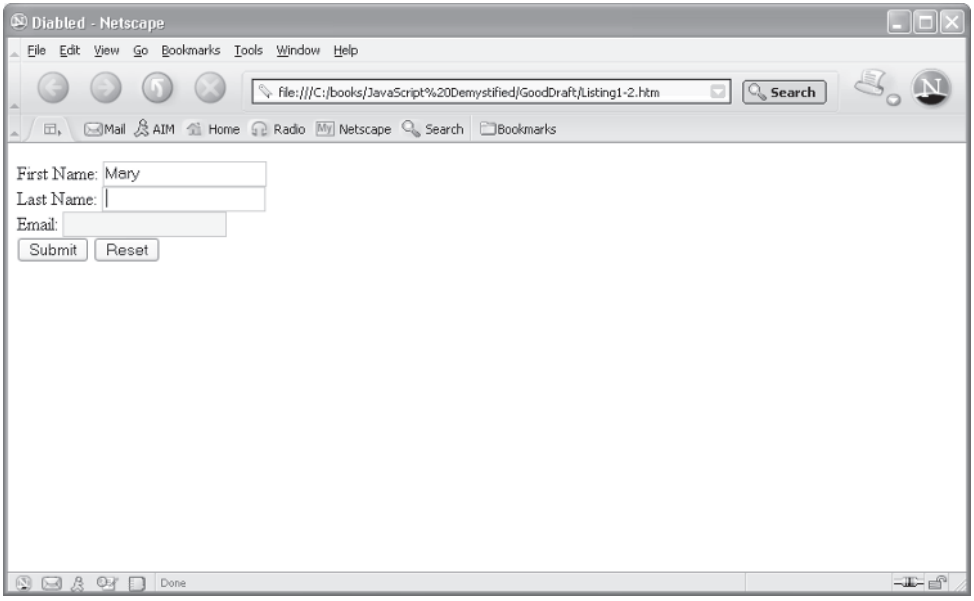

Figure 7-11 The Email element is disabled until the first and last names are entered into the form.

entered for both, then the `Email` element is enabled by resetting the `disabled` attribute to false.

```
<!DOCTYPE html PUBLIC
         "-//W3C//DTD XHTML 1.0 Transitional//EN">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
   <title>Disabled</title>
      <script language="Javascript" type="text/javascript">
         <!--
         function EnableEmail()
         {
            with (document.forms.Contact)
            {
               if (Fname.value.length >0
                     && Lname.value.length >0)
               {
                  Email.disabled = false
               }
            }
         }
         -->
      </script>
</head>
   <body>
      <FORM name="Contact"
              action="http://www.jimkeogh.com" method="post">
         <P>
         First Name: <INPUT type="text"
             name="Fname" onchange=" EnableEmail()"/> <BR>
         Last Name: <INPUT type="text"
               name="Lname" onchange=" EnableEmail()"/><BR>
         Email: <INPUT type="text"
                name="Email" disabled=true/><BR>
         <INPUT name="Submit" value="Submit" type="submit"/>
         <INPUT name="Reset" value="Reset" type="reset">
        </P>
     </FORM>
   </body>
</html>
```

Read-Only Elements

You can use a JavaScript function to change the value of an element that the user cannot change (a read-only element). This is possible by setting an element's `readonly` attribute. If the `readonly` attribute is set to true, then no one, including your JavaScript function, can change the value of the element. If the `readonly` attribute is set to false, then anyone, including the user entering information into the form, can change the value of the element.

You can change the value of the `readonly` attribute from within your JavaScript function. This is demonstrated in the next example, which was used earlier in the chapter when the JavaScript function created an e-mail address based on the user's first and last names.

Look carefully and you'll see a new twist in this new JavaScript, however. Notice that the `Email` element is set to `readonly`. This means that the user cannot enter an e-mail address. Each time the value of the `Fname` and `Lname` elements change, the `SetEmail()` function is called. This function examines the `Fname` and `Lname` elements and creates the e-mail address if both names have been entered. However, before assigning the e-mail address to the `Email` element, the function resets the `readonly` attribute to false, thereby enabling the function to write to the `Email` element. After the e-mail address is assigned to the `Email` element, the function sets the `readonly` attribute back to true, thus preventing the user from changing the e-mail address.

```
<!DOCTYPE html PUBLIC
      "-//W3C//DTD XHTML 1.0 Transitional//EN">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
   <title>Read Only</title>
      <script language="Javascript" type="text/javascript">
         <!--
         function SetEmail()
         {
            with (document.forms.Contact)
            {
               if (Fname.value.length >0
                   && Lname.value.length >0)
               {
                  Email.readonly = false
                  Email.value = Fname.value.charAt(0)
                      + Lname.value + '@mcgrawhill.com'
                  Email.readonly = true
               }
```

```
            }
         }
         -->
      </script>
</head>
   <body>
      <FORM name="Contact"
               action="http://www.jimkeogh.com" method="post">
         <P>
         First Name: <INPUT type="text"
             name="Fname" onchange="SetEmail()"/><BR>
         Last Name: <INPUT type="text"
              name="Lname" onchange="SetEmail()"/><BR>
         Email: <INPUT type="text"
              name="Email" readonly=true/><BR>
         <INPUT name="Submit" value="Submit" type="submit"/>
         <INPUT name="Reset" value="Reset" type="reset">
        </P>
     </FORM>
   </body>
</html>
```

Looking Ahead

You can make a form come alive by using a little JavaScript. A form consists of elements, such as radio buttons and check boxes, that are used to gather information from a user. An element can contain one or more attributes, such as a name and other values that can be changed by statements within a JavaScript.

A JavaScript can be executed when an event occurs while the user is entering information into a form. An event is something the user does to the form, such as clicking a button, selecting a check box, or moving the cursor away from an element. In this chapter, you learned about the various events that occur while the form is displayed on the screen.

You identify the event to which you want to respond by using the name of the event within the opening tag of the element that is affected by the event. You also must assign the name of the JavaScript function that you want called when the event occurs.

Two kinds of JavaScript functions can be called: intrinsic functions that are defined by JavaScript, such as `submit()` and `reset()`, and functions that you