CHAPTER 10

Forms

At some point, almost every site other than a purely personal one will use a form somewhere within it. The form may be as simple as a contact form, or it may be part of a database-driven section or a content-management system.

Expression Web has several built-in tools to help you create forms and collect the form results. Whether you are using the form to send e-mail from site visitors or to update and maintain a database, Expression Web has a tool to make your job as the website designer easier. There are two basic types of forms you can use in Expression Web. The first is traditional HTML forms, which are sent to a script for processing. If you are using a web server that does not support ASP.NET 2.0, you would use this type of form. The second type of form uses ASP.NET 2.0 form controls and passes information directly by using a postback or calling a function on a `.ascx` page. Whether you are using HTML `<form>` elements or ASP.NET form controls, the basic method of creating a form is the same.

In the first part of this chapter, we will be using HTML `<form>` elements. In the second part, any differences using ASP.NET form controls will be addressed. As we create our form, we will also cover ways to improve the usability of your forms and make them accessible to visitors using alternative access devices and screen readers.

Required Elements for a Form

Regardless of the method used to create a form, certain components must be included in order for a form to function as follows:

- *Form element*: In the `<form>`, you set the action that happens when a user submits a form and specifies the form name.
- *Form field*: There are many types of form fields, ranging from text boxes to radio buttons and check boxes.
- *Submit button*: Use this button to send the form to the form handler.

Expression Web makes it very easy to create your own form by dragging `<form>` elements from the HTML or ASP.NET section of the Toolbox.

When the Form Controls section is expanded, as shown in Figure 10-1, you can see a list of the available form field types, as well as other form-related options available for you to insert on your web page.

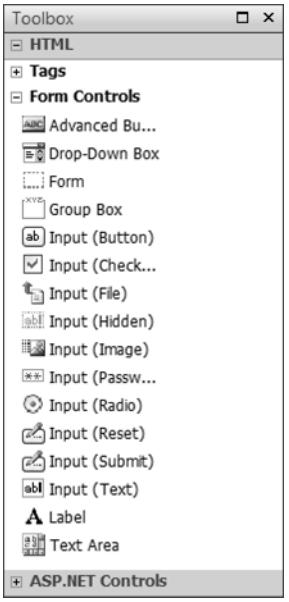

Figure 10-1. *Toolbox HTML Form Controls*

■ **Caution** Form field names are for your server-side processing; they do not show as labels on the web page.

To access the dialog boxes to set the properties for each of the form fields, you will need to right-click the form control after you drag it from the Toolbox to your page. The Form Field Properties dialog box is where you set the field name and any options that are appropriate for the type of form control you will be using.

NAMING YOUR FORM FIELDS

You should decide on a naming convention that you will use on all your websites. A consistent convention will make it easier to know what the fields do when you come back to update your pages. By the same token, you should use descriptive names, such as `firstname` or `fname`, instead of the default `text1`.

There are reserved words that you should never use for form field names, since they are used by server-side processing scripts and/or databases to perform specific tasks. The following list provides some of the reserved words that you might use as a descriptive name without realizing the word is reserved:

- date
- desc

- end
- false
- from
- name
- number
- state
- time
- true

For a more complete list, see `http://msdn.microsoft.com/library/default.asp?url=/library/en-us/tsqlref/ts_ra-rz_9oj7.asp`.

Form fields should always have a label associated with the form field. Using a label with check boxes and radio buttons is especially important. If you have ever had difficulty selecting the correct radio button or checking a small check box, you will appreciate that using the `<label>` element provides a larger clickable area. This is both a usability and accessibility feature.

```
<label id="lblred">
<input name="color" type="checkbox" value="red" /> Red</label>
```

A check box using the preceding label syntax means that double-clicking the name "red" or tabbing to it and using the spacebar will check or uncheck the box. A common practice is to use the control name or in the case of a check box the value prefixed with `lbl` for the label ID.

Let's go through some of the form controls:

Advanced Button: The primary difference between the Advanced Button form control and the Input (Button) control is the ability to set height and width, as shown in Figure 10-2.

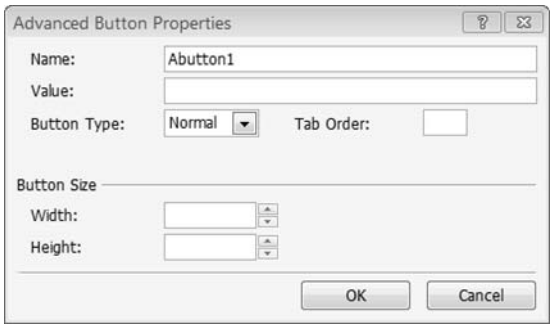

Figure 10-2. *With the Advanced Button Properties dialog, you can set the height and width of a button.*

Drop-Down Box: This control is used to create a drop-down list of choices. Figure 10-3 shows the Add Choice dialog box as well as the Drop-Down Box Properties.

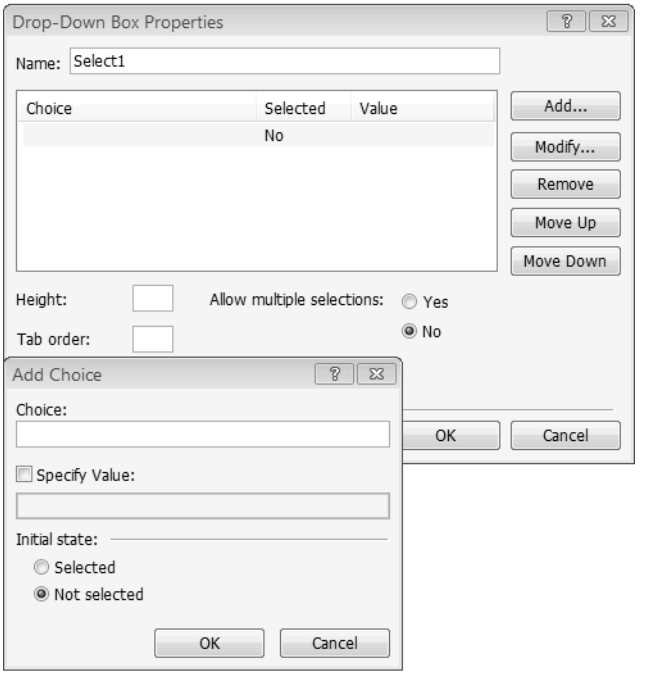

Figure 10-3. *Add choices to your drop-down form field.*

Group Box: This creates a fieldset to contain related form fields. In the following example, an Expression Web Group Box control was used to wrap a fieldset around a series of check boxes. Fieldsets serve an important accessibility function, since they tell screen reader users that the fields are related by reading the `<legend>` before reading the form controls, while the border does the same for sighted visitors. Figure 10-4 shows setting the fieldset legend text or, as Expression Web calls it, the Label using the Group Box Properties dialog, which creates the fieldset around the field group.

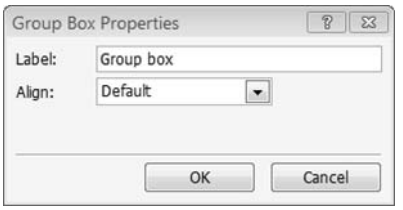

Figure 10-4. *Create a fieldset by using the Group Box control from the HTML Toolbox form control.*

Here's the resultant code:

```
<fieldset name="Group1">
  <legend>Colors</legend>
  <label id="lblred"><input name="color" type="checkbox" value="red" /> Red</label>
  <label id="lblblue"><input name="color" type="checkbox" /> Blue</label>
  <label id="lblgreen"><input name="color" type="checkbox" /> Green</label>
</fieldset>
```

Figure 10-5 shows the resulting display in a web browser; note the thin outline surrounding the group of related check boxes.

Figure 10-5. *When a fieldset is used, a faint outline will be displayed in the browser.*

Input (Button) lets you insert a button that will be sized based on the text displayed in the Values/label text field. Figure 10-6 shows the Push Button Properties dialog box, which is the same dialog box displayed for the Input (Reset) and Input (Submit) buttons. The only difference is which of the three radio buttons is selected by default.

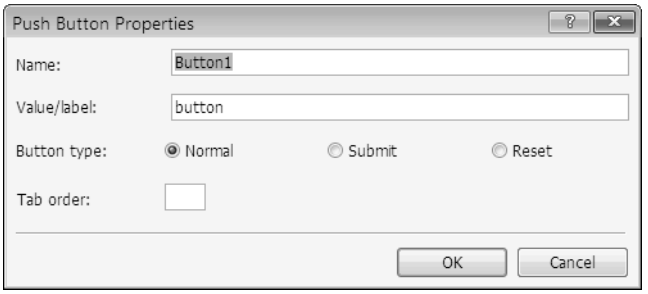

Figure 10-6. *Basic button choices*

Input (Checkbox) toggles the field as either off (unchecked) or on (checked), as shown in Figure 10-7.

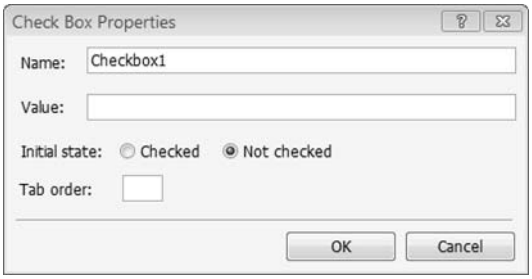

Figure 10-7. *You can select a default state for your check box.*

■ **Tip** Unlike most input controls, check boxes and radio buttons can share the same control name so that your results are grouped when you receive them.

Input (File) opens the File Upload Properties dialog that allows you to add a box for a file name and a browse button to allow users to select a file to submit with the form. Make sure that your form destination is writable. Its properties dialog is shown in Figure 10-8.

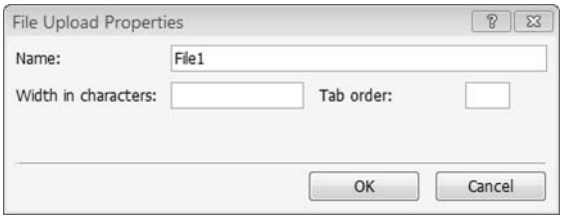

Figure 10-8. *File uploads require you to have write permissions on the destination folder.*

Input (Hidden) is a control that users can't see in their browser (but can see in the source code), which submits information with the form. There is no form properties dialog box available for this control, so the name and value must be set in Code view or by using the Advanced button on the Form Properties dialog box, as shown in Figure 10-9.

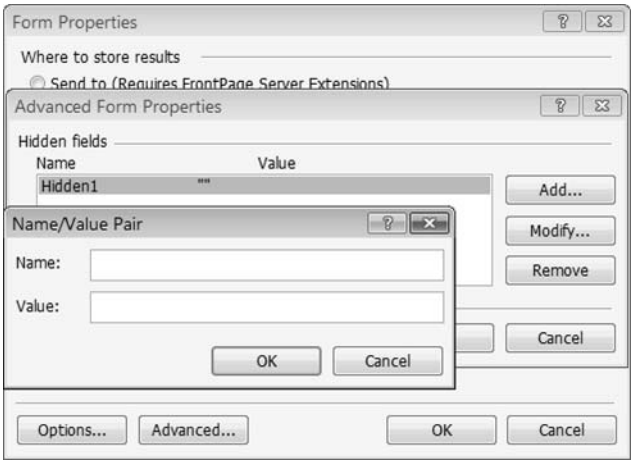

Figure 10-9. *Use the Advanced Form Properties dialog box to add hidden field name and value pairs.*

Input (Image) opens the Picture Properties dialog, which allows you to use an image as a submit button, as shown in Figure 10-10.

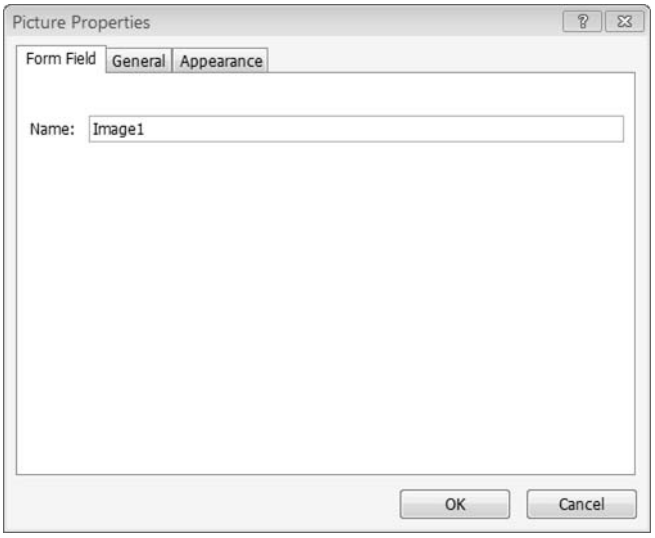

Figure 10-10. *The same dialog box to name the button and select the image appears whether you select Picture Properties or Form Field Properties from the right-click menu.*

Input (Password) is a single-line box for users to enter a password. The password is hidden, usually with asterisks displayed in the browser. The field and its form field properties are identical to Input (Text) with the exception of which button is selected for the text box type, as shown in Figure 10-11.

Figure 10-11. *A password text box is a text box where the input is replaced by asterisks and not displayed in the browser.*

Input (Radio) creates radio buttons. Radio buttons are exclusive, that is only one button in a group of radio buttons with the some name can be selected at a time. If you want to allow multiple choices to be selected, use a check box instead. Because radio buttons are small, it is very important that you use a label with this type of form field. For a group of buttons, make sure that you use the same group name in the Option Button Properties box, shown in Figure 10-12, and a fieldset to group them, as we did with the check boxes in Figure 10-4.

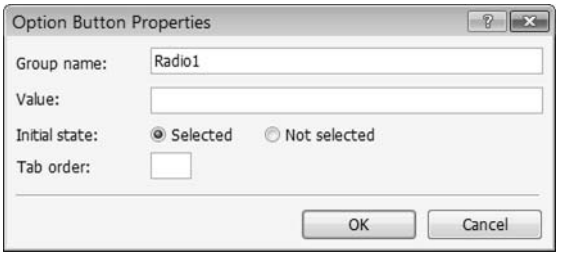

Figure 10-12. *It is usually best not to start with a radio button selected when you have more than one choice.*

Input (Reset) creates a button that resets the form (see Figure 10-10).

Input (Submit) creates a button that submits the form (see Figure 10-10).

Input (Text) creates a single line box for users to enter text (see Figure 10-8).

Label adds a label to describe another form control. Select the form element you want to label, right-click Label in the HTML Toolbox Form Controls section, and select Wrap, as shown in Figure 10-13.

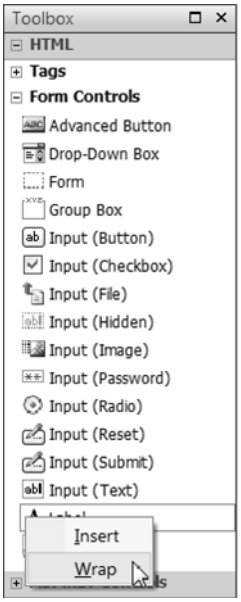

Figure 10-13. *To enclose your selection with a label, use Wrap, not Insert.*

Text Area creates a large box for users to enter text; you specify the height and width of a text area in rows and characters (see Figure 10-14), which are wider than most text characters. Make sure you have enough space on your page, or you may end up with a horizontal scrollbar in your browser.

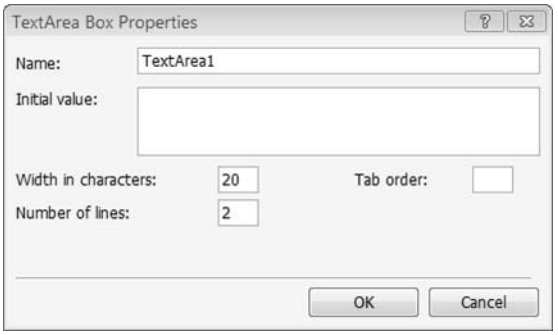

Figure 10-14. *Don't forget to check the width of your text box in your browsers.*

Exercise 10-1. Creating a Simple Contact Form

In this exercise, we will create a simple contact form allowing your visitors to send you feedback from your website:

1. Create a new blank page, and save it as `form1.html`. Add **Contact Us** text, and make it an `<h1>`.
2. To create the form, drag the Form control from Toolbox ➤ HTML ➤ Form Controls, and drop it in Design view just below your `<h1>` page title.
3. Next, inside your form, type **Name:**, drag out an Input (text) control, and drop it to the right of the text.
4. Select the input, and right-click to launch the Form Field Properties dialog box, as shown in Figure 10-15. Name your text field `sname`, short for "sender name," and click OK. The name attribute is necessary to send your form's field value to your form's processor and for older browsers to associate the label with the form field. For modern browsers, you should also add `id="sname"` using the Tag Properties task pane or the Quick Tag Editor.

Figure 10-15. *Change the form field name before you add a label.*

5. After your text field has been renamed, wrap a Label control around the text by selecting the text, right-clicking Label in the Toolbox, and selecting Wrap, as shown in Figure 10-16.

Figure 10-16. *Highlighing the text and the form field will wrap the label around the selected items.*

6. The resulting code will look like this:

```
<form method="post" action="">
  <label id="label1">Name: </label> <input name="sname" type="text" /></form>
```

7. We now need to associate the label with the form field. Right-click `<label#label1>` in the Quick Tab bar, and choose Edit Tag, as shown in Figure 10-17.

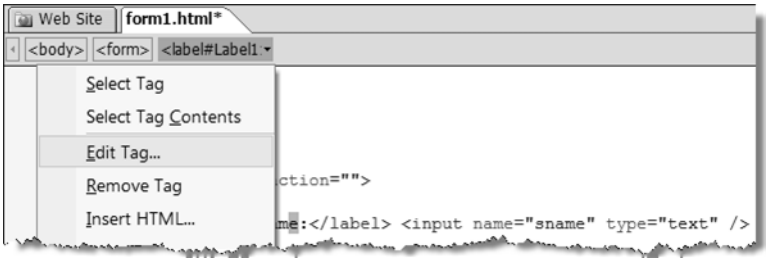

Figure 10-17. *Edit the label to associate it with your name text box.*

8. When the Quick Editor is launched, change the label ID from `lable1` to `lblsname`, and type **for="sname"** or choose `for` from the IntelliSense drop-down, as shown in Figure 10-18.

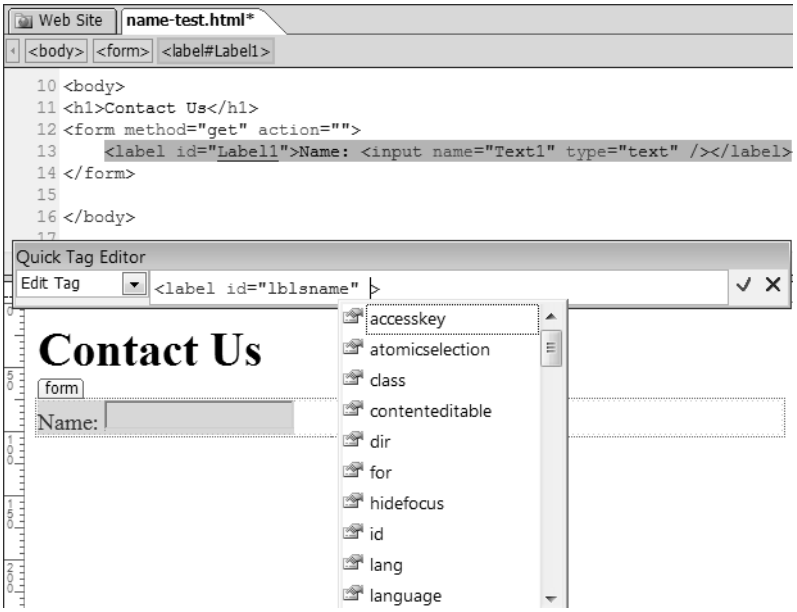

Figure 10-18. *Associate the label with the form field using Quick Editor.*

9. We are using the `for` attribute instead of wrapping our form fields with the label so that we can style the form once it is finished. By changing the `id` from the default to `lbl[field_name]`, we are able to keep track of which label goes with each field. You can associate more than one label with an individual control.

10. Use Shift+Enter to start your next field on a new line. Next, add a text box for an e-mail field.

■ **Tip** Use your right arrow key or your space bar after you have finished wrapping your selected content in a label before you start a new line to make sure that the label doesn't continue to the next line as well.

---

11. Add another line break, and type **Message:** followed by a TextArea control, as shown in Figure 10-19. Adjust the default size to give your visitors visual room to write in their messages, but make sure you preview it, since it may be wider than you think.

Figure 10-19. *Use the TextArea Box Properties dialog to create the e-mail field.*

12. Now, add a submit button on a new line at the bottom of the using Input (Submit). Following the process described in steps 5–10, add labels to all the input fields except the submit button; a label is not needed on a submit button. The finished form should look like the one shown in Figure 10-20.

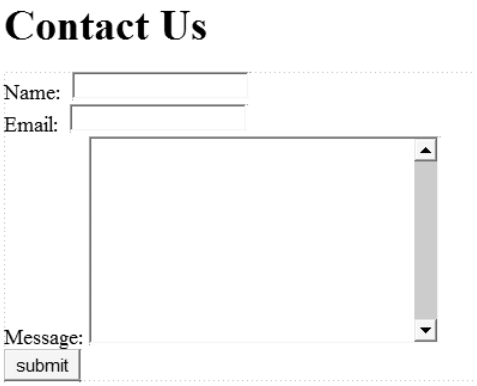

Figure 10-20. *The unstyled contact form is not very attractive.*

13. The final step in this exercise will be to style our form so that it is more visually appealing. By styling the label element, we can correctly line up all of our fields except the submit button, which does not have a label. You can create a class with padding on the left side to match the width of the `<label>` elements if

you wish. However, because of differences in how the browsers calculate margin and padding, the submit button may be in slightly different locations depending on the browser you are using. An alternative is to place the submit button in a new paragraph and use `text-align: center;` to center the submit button in the form box. Use the following CSS style definitions:

```
form {
  border: thin solid #000000;
  padding: .5em;
  margin: 1em auto 1em auto;
  width: 25em;
  background-color: #CEDBAE;
  font-weight: bold;
}
label {
  float: left;
  padding-right: .5em;
  width: 5em;
  text-align: right;
  font-weight: bold;
}
.submit {
  font-weight: bold;
  text-align: center;
}
```

14. Apply the submit class to the <p> holding the submit button. When viewed in a browser with the preceding styles applied, your form should now look like the one shown in Figure 10-21.

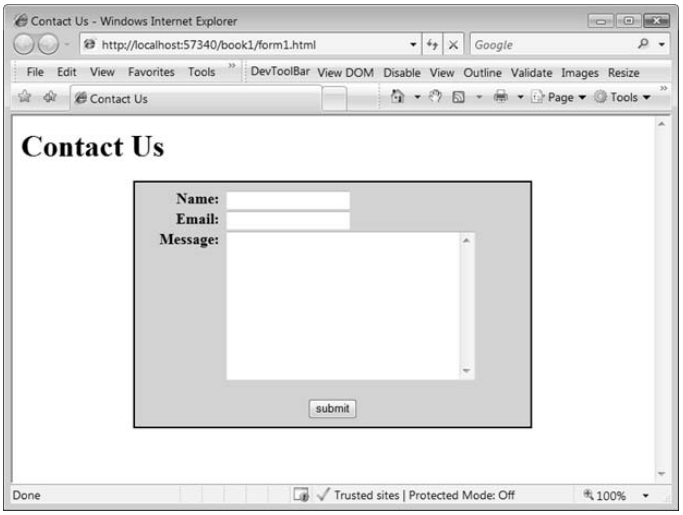

Figure 10-21. *Styled using just HTML elements and one class, the form now looks good.*

15. Use different background colors, fonts, and other style properties to create the form presentation you desire.

The HTML form fields we have just used will allow you to create a wide variety of forms, but unless the form is connected to a forms processor, the information collected will be lost in cyberspace. The next section will connect our forms to a server-side processing script to do something with the information collected.

Processing Forms

Normally, a form handler is some sort of server-side script. Some Expression Web users will be migrating from FrontPage and will have been using the FrontPage Server Extensions (FPSE) to process their forms. If your web host offers FPSE, you do not need to know how to write a server-side script; Microsoft has included some common form-processing tasks using web-bots. The Form Properties dialog box is also where you would be sending your HTML form results to a custom script. You set these options via the Form Properties dialog box, as shown in Figure 10-22.

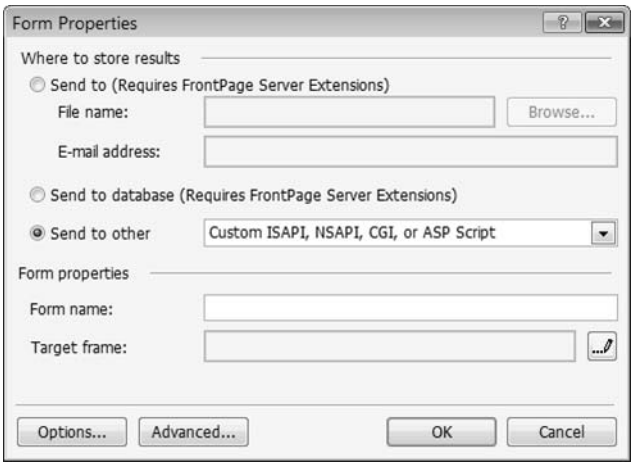

Figure 10-22. *The Form Properties dialog is where you specify what you want Expression Web to do with the collected form information.*

If you are hosting on a web server that supports ASP.NET 2.0, you should use the ASP.NET Form Controls, covered in the "ASP.NET 2.0 Form Controls" section later in this chapter, instead of the forms processing covered in the following section.

Setting Form Properties

First, we will be using FPSE to process the web form to send the result by e-mail. Right-click anywhere inside your form, and select Form Properties to launch the Form Properties dialog box.

The options you can set in the Form Properties dialog box follow:

- *Send to File name*: This is the default action that will send the form results to a text file using a comma-separated values format. Unlike FrontPage, Expression Web will not create the `_private` directory with the correct permission in which to store your file results. You will need to create one yourself locally if you have not downloaded one from your hosted site. Web servers are normally configured so that folders beginning with an underscore are not served to the browser directly. This security depends on the server being configured correctly, so do not rely on it for sensitive information.
- *Send to E-mail address*: If you want the form results to be sent by e-mail, place the e-mail address in this box. You can send a copy to multiple e-mail addresses if you separate them with commas, but there have been reports that this is buggy; it is better to daisy chain forms to send them to multiple e-mail addresses. Unfortunately, the current webbot will include your e-mail address in the code, which makes it accessible to form-processing bots. Avoiding having your e-mail address harvested is one of the benefits of using a form instead of using `mailto:email` as the `action` on the `<form>` element. When you do this, Expression Web will give you a warning that this action requires FPSE to function, as shown in Figure 10-23.

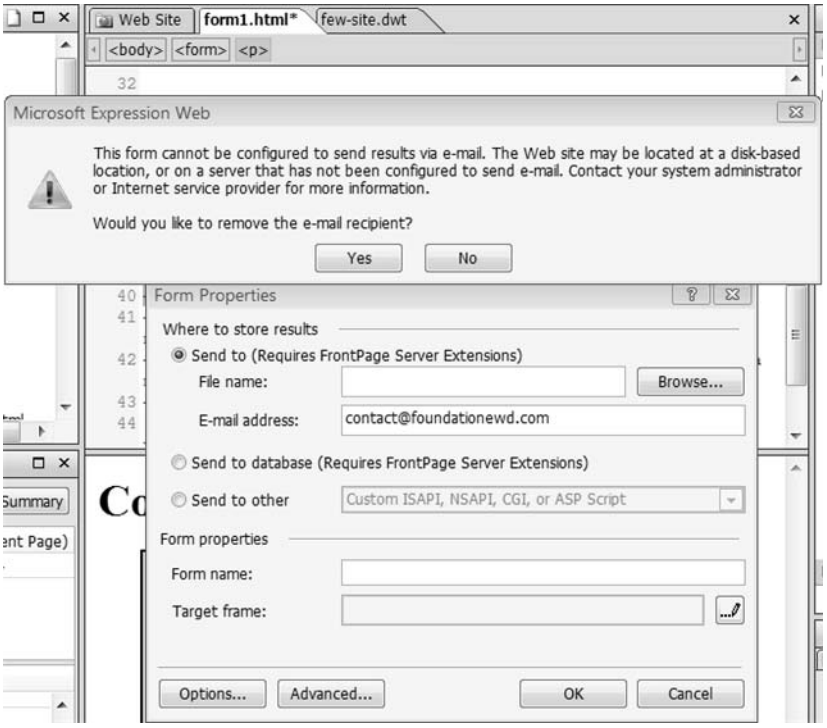

Figure 10-23. *FrontPage Server Extensions are required to use this Send to E-mail address method.*

Click No; otherwise, the e-mail address will be removed, and you will not receive the e-mail. Until the form is published to a web server with FPSE installed, you will see a warning, when you save or preview locally, that a web component is being used that requires the FPSE.

You may use both the send-to-file and send-to-e-mail options simultaneously. This provides a backup of the form information in case there is a problem with receiving the e-mail. (If you have spam filters set up, make sure the `from` address the form uses is whitelisted.)

- *Send to database*: This option is only available on a Windows server. When this option is selected, you need to use the Options button to specify the database and connection information. If you do not have an existing database, Expression Web creates an Access database to receive the results of the form when you follow the prompts. Since FPSE is being phased out, it is better to use ASP.NET 2.0 for new databases.
- *Send to other*: This is usually your best option for secure forms processing that does not expose your e-mail address to bots for harvesting. Many web hosts have preinstalled scripts available. Usually, those are something like CGI Formmail. Other options include sending the form results to an ASP or other custom script.
- *Form name*: Give your form a name that will allow you to keep track of the form's purpose instead of using the default Expression Web name.
- *Target frame*: If you are using a frameset, this allows you to send the results to a specific target `<frame>` or `<iframe>`.
- *Options*: The dialog box that appears when you use this button will vary depending on the radio button selected. The primary dialog boxes triggered follow:
  - *Saving Results*: This dialog box responds to the "Send to File name" or "Send to E-mail address" radio buttons.
  - *Options for Saving Results to Database*: This one responds to the "Send to database" radio button for specifying your database connection or creating a new one using the FPSE. You do not create ASP.NET 2.0 connections using this method.
  - *Options for Custom Form Handler*: This dialog responds to the "Send to other" radio button. This is also where you set whether your form will send the results in a URL string or hidden in the file headers, where the visitor cannot see or save them.
- *Advanced*: This triggers the Advanced Form Properties dialog box, where you add hidden fields that the visitor will not see but that contain information that transfers data to the forms processor, as shown in Figure 10-9.

If you use one of the FPSE bots to handle your forms processing, all you need to do is follow the prompts. If you are using a custom script, consult the documentation that comes with your script. To send the results of the form we created in the preceding exercise, set your form result options as shown in Figure 10-24.

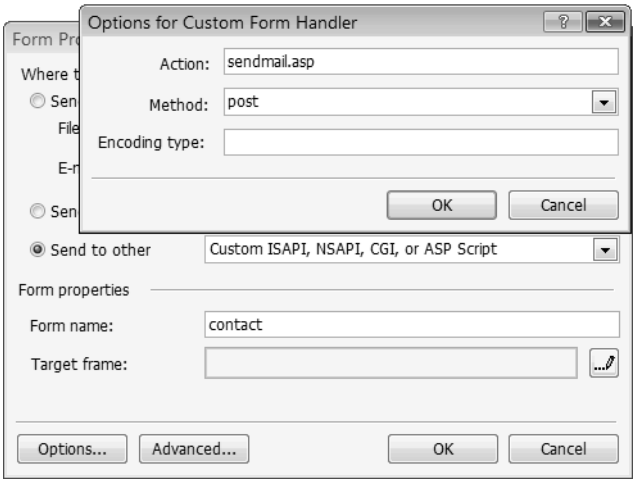

Figure 10-24. *This sends results to a classic ASP page for processing.*

Let's have a look at a couple of forms-processing scripts, starting with `sendmail.asp`.

ASP Send with CDO

If you are hosted on a Windows 2000 or Windows 2003 server, you can send e-mail using Classic ASP and CDO. Listing 10-1 shows `sendmail.asp`, which will send your `form1.html` results to the e-mail address you specify. Make sure that you create a `thankyou.html` page. To use the following ASP code, you must be hosted on a Windows server that supports CDONTs. Check with your web host; your host may have a different method such as ASPEmail or ASP-Mail that you should use instead.

■ **Note** Replace `youremail@yourdomain.com` with the e-mail address you want the form sent to. You can change the subject line to the one of your choice.

Listing 10-1. *A Classic ASP Page for Processing Our Contact Form*

```
<%@LANGUAGE="VBSCRIPT" CODEPAGE="1252"%>
<%
dim sname, email, message, subject

sname = Request.Form("sname")
email = Request.Form("email")
subject = "Domain Contact Form"
smessage = Request.Form("message")
```

```
Dim ObjMail
 Set ObjMail = Server.CreateObject("CDO.Message")
 objMail.From = sname & " <" & email & ">"
 objMail.To = "yourmail@yourdomain.com"
 objMail.Subject = "Contact from Your Website"
 objMail.TextBody = smessage
 objMail.Send
%>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
    <title>Thank You</title>
  </head>

  <body>
    <p>Thank You <%=sname %></p>
    <p>If your message:</p>
    <blockquote>
      <p>&quot;<%=smessage %>&quot;</p>
    </blockquote>
    <p> Requires a response it will be sent to <%=email%></p>
  </body>
</html>
```

If you are hosted on a Linux server, you cannot use ASP to send your mail, but in most cases, you can use PHP.

PHP Send Mail

The code in Listing 10-2 will send results from our simple contact form using PHP and display a thank you page to your site visitor. Change the action of the form to `sendmail.php`, and copy the following code to a page named `sendfile.php`. Otherwise, the instructions are the same as for the previous ASP method.

■ **Note** Replace `youremail@domain.com` with the e-mail address you want the form sent to. You can change the `$subject` line to the one of your choice.

Listing 10-2. *A PHP Page for Processing Our Contact Form*

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
```

```
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
    <title>Thank You</title>
  </head>
  <body>
    <p>
      <?php
      $email = $_POST['email'] ;
      $sname= $_POST['sname'] ;
      $message = $_POST['message'] ;
      $subject = "Contact from Your Website" ;
      //mail($email, $subject, $message, "From: $email" );
      mail("youremail@domain.com", "$subject", "$message","From: $sname <$email>");

      ?>
    </p>
    <p>Thank you, <?php echo $sname ?></p>
    <p>If your message:</p>
    <blockquote>
      <p>&quot;<?php echo $message ?>&quot;</p>
    </blockquote>
    <p>Requires a response it will be sent to : <?php echo $email ?>
    </p>
  </body>
</html>
```

Neither of the contact forms–to–e-mail processing scripts have antispam checks. If your web host offers a more robust form processor, such as ASPEmail, you should use it.

ASP.NET 2.0 Form Controls

To create an ASP.NET 2.0 form, begin by creating a `.aspx` page. If you wish to use the default language of C#, you can click the new file button on the standard toolbar and select ASPX, as shown in Figure 10-25.

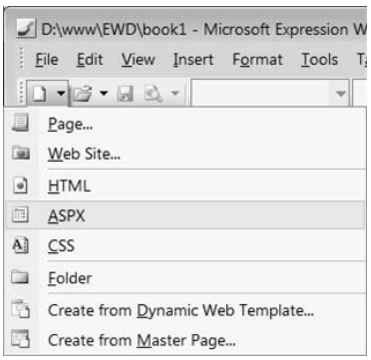

Figure 10-25. *A new ASP page using your Expression Web default language settings*

If you wish to change the `.aspx` page language, you must choose the Page tab and change the language in the Options section at the bottom of the Description column of the dialog box, as shown in Figure 10-26.

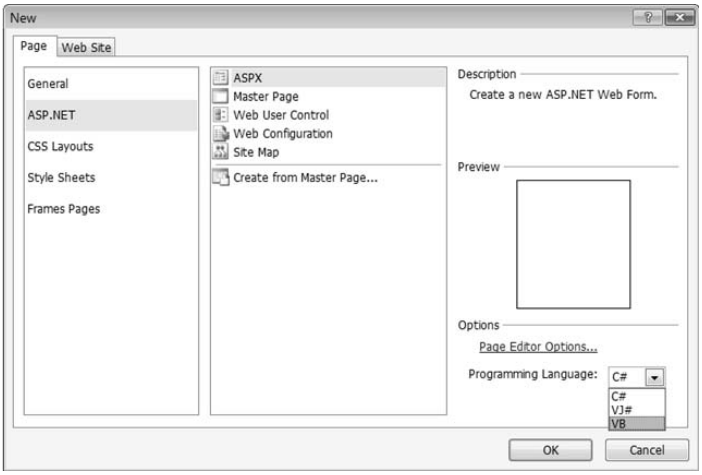

Figure 10-26. *Change the page language using the drop-down options with ASP.NET as the chosen page category.*

You can choose which of three supported ASP.NET languages you prefer to use in the Programming Language drop-down. In Exercise 10-2, we will be using VB.NET (VB). The other supported language is Visual JavaScript (VJ#). When you create a `.aspx` web page, a form control is added by default, since all ASP.NET is processed on the server. A new ASP.NET 2.0 page has more code on it than a simple HTML page, as shown in Figure 10-27.

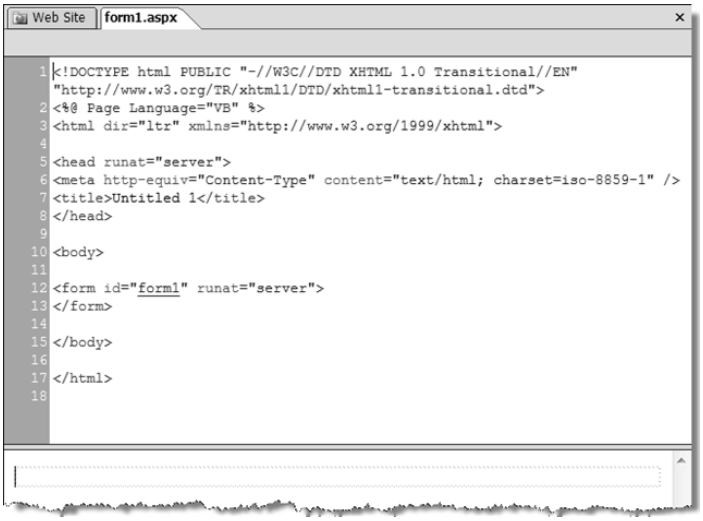

Figure 10-27. *The server-side page language and <form> element are added to all new* `.aspx` *web pages.*

In the Toolbox under ASP.NET, you will find the form controls in the Standard section along with a bunch of other controls not relevant to forms, as shown in Figure 10-28.

Figure 10-28. *ASP.NET 2.0 Standard Toolbox controls*

With a few exceptions, such as Group Box (fieldset), the form controls you see will be the same as those in the HTML Form Controls. For the most part, the ASP.NET form controls function in the same way as their HTML equivalents, but there are not separate controls for text box and textarea elements. The number of rows and columns you set for a TextBox determine whether the control will display as a text field or as a text-area. None of the ASP.NET form controls have dialog boxes associated with them to set the control properties, because the control attributes are set using the Tag Properties task pane.

Figure 10-29 shows the properties available for an ASP.NET label control. With an ASP.NET label control, the ID is used by scripting, and you associate the label with a form control using the `AssociatedControlID` property instead of the `for` attribute. You set the `AssociatedControlID` attribute using a drop-down menu with the ID of all the ASP.NET controls on your page.

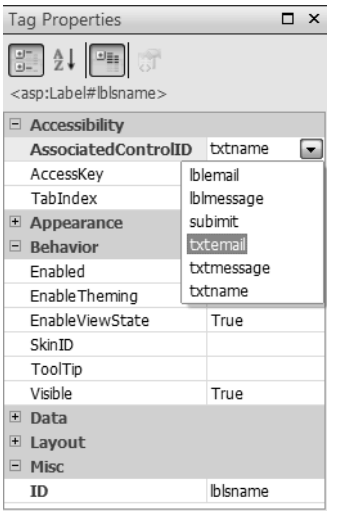

Figure 10-29. *ASP.NET Label Control properties.*

Exercise 10-2. Creating an ASP.NET 2.0 Contact Form

In Exercise 10-1, we created an HTML contact form. In this exercise, we will be creating the same contact form using ASP.NET to create the form and send the form results to you by e-mail.

We will begin with a new `.aspx` web page, but we will not be using the default C#. Instead, we will use VB.NET, since the forms–to–e-mail script we will be using to send results is written in VB.NET and will work with ASP.NET 1, 1.1, and 2.0 to provide compatibility with a wider variety of ASP.NET servers.

1. Begin by selecting New ➤ Page ➤ ASPX, making sure that VB is selected as the Programming Language.
2. Drag, from the ASP.NET Controls Standard section of the toolbox, the following controls using Ctrl+Enter for a line break at the end of each section as shown—do not set the attributes at this time:

```
Label space TextBox
Label space TextBox
Label space TextBox
Button
```

3. Before you create your label text and associate the labels with form controls, name your form fields. With ASP.NET controls, you can easily add required form validation, which will be covered in Chapter 13. For this exercise, select the first TextBox, and give it the ID of `txtname`. Notice in Figure 10-30 that the first text field is set to SingleLine, which will render in your browser the same as in the HTML text box.

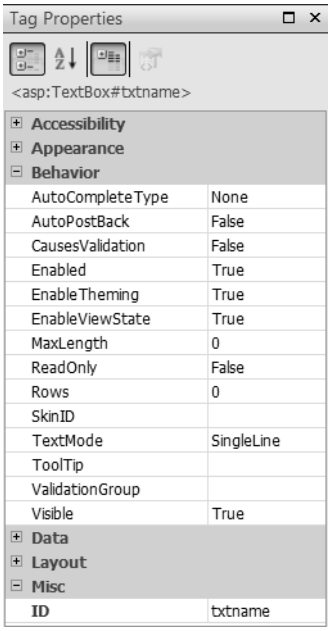

Figure 10- 30. *ID is the last of the ASP.NET Contro Propertys in the Tag Properties task pane.*

4. Repeat step 2 for the next TextBox, and use `txtemail` for the ID. For the third TextBox, use `txtmessage` as the ID, and change the number of rows to 10. ASP.NET controls use width instead of columns to set the width of the text area; set the width to 300px. To finish transforming this TextBox into the equivalent of an HTML text area form field, select the TextMode attribute to trigger the options drop-down menu, and select MultiLine, as shown in Figure 10-31.

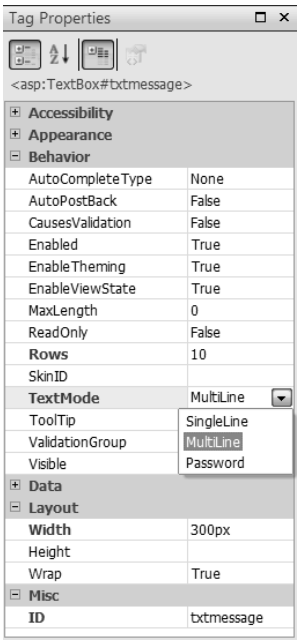

Figure 10-31. *Change a text box to a text area by selecting MultiLine.*

5. Now that your form fields have meaningful IDs assigned, associate each of your label fields with the control to its right, using the `AssociatedControlD` property. Give each label the ID of `lbl[control_name]`, as shown in Figure 10-32.

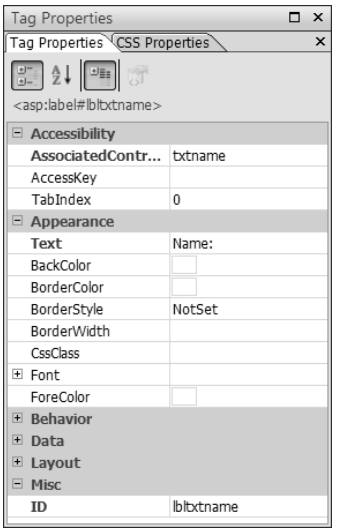

Figure 10-32. *ASP.NET label properties when associated with a TextBox*

6. The last ASP.NET control we need to set is the button, which we need to turn into a submit button. Select the button, and from the Tag Properties dialog, match the settings in Figure 10-33, particularly the `UseSubmitBehavior` property.

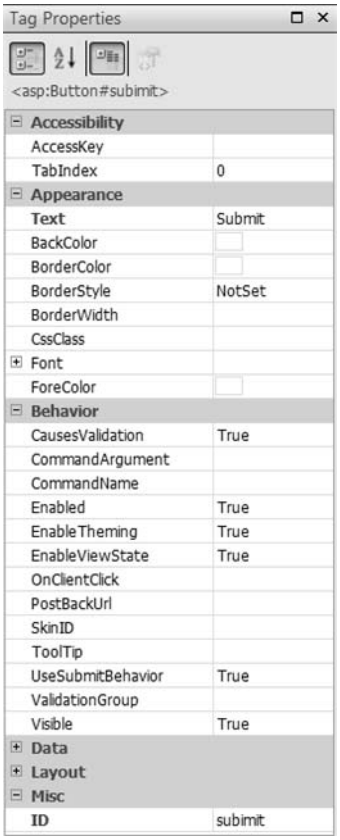

Figure 10-33. *To turn an asp:button into a submit button, you must set the UseSubmitBehavior to True*

7. Save your file, and don't forget to put in the page title. Your code should now look like this:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<%@ Page Language="VB" %>
<html dir="ltr" xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
  <title> ASP.NET form </title>
</head>
```

```
<body>
  <form id="form1" runat="server">
    <asp:Label runat="server" Text="Name: " id="lblsname"
               AssociatedControlID="txtname"></asp:Label>
    <asp:TextBox runat="server" id="txtname"></asp:TextBox>
    <br />
    <asp:Label runat="server" Text="Email: " id="lblemail"
               AssociatedControlID="txtemail"></asp:Label>
    <asp:TextBox runat="server" id="txtemail"></asp:TextBox>
    <br />
    <asp:Label runat="server" Text="Message: " id="lblmessage"
               AssociatedControlID="txtmessage"></asp:Label>
    <asp:TextBox runat="server" id="txtmessage" Width="300px" Rows="10"
                 TextMode="MultiLine"></asp:TextBox>
    <br />
    <asp:Button runat="server" text="Submit" id="submit" />
  </form>
</body>

</html>
```

8. When viewed in a browser, our web page should now look like the one shown in Figure 10-34. Notice that the ASP.NET form is no more attractive than the HTML form when it is not styled.

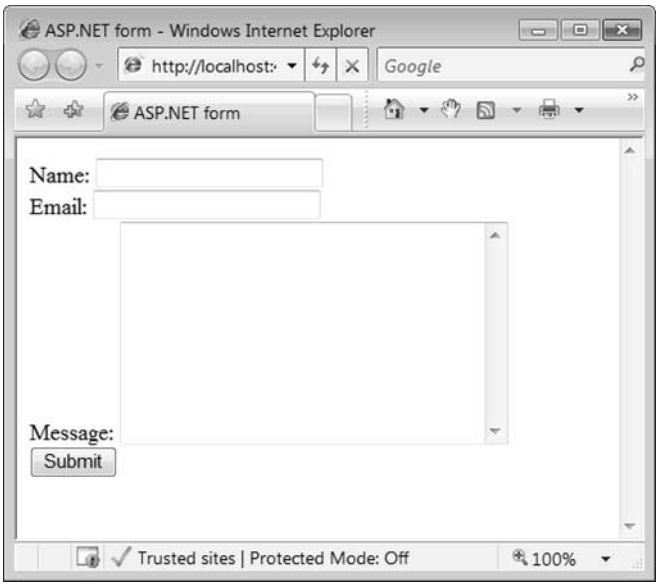

Figure 10-34. *A completed ASP.NET form looks no different in your browser than an HTML form.*

9. Select View Source in your web browser, and you will see that the ASP.NET controls output HTML to the browser. Compare the following code that your browser receives with the code you see in Expression Web:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html dir="ltr" xmlns="http://www.w3.org/1999/xhtml">
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
  <title>
    ASP.NET form
  </title>
</head>

<body>
  <form name="form1" method="post" action="form1.aspx" id="form1">
    <div>
      <input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
             value="/wEPDwULLTE1OTk3MzUzNjZkZJXkbwsMdcsZkJTDwcGKDZXW33Au" />
    </div>
    <label for="txtname" id="lblsname">Name: </label>
    <input name="txtname" type="text" id="sname" />
    <br />
    <label for="txtemail" id="lblemail">Email: </label>
    <input name="txdtemail" type="text" id="email" />
    <br />
    <label for="txtmessage" id="lblmessage">Message: </label>
    <textarea name="txtmessage" rows="10" cols="20" id="message"
              style="width:300px;"></textarea>
    <br />
    <input type="submit" name="submit" value="Submit" id="submit" />
    <div>
      <input type="hidden" name="__EVENTVALIDATION" id="__EVENTVALIDATION" ➥
value="/wEWBQLb9vGECgLNOvKVCgKyzcaDDQLs7cyPDAKy24GUBJSNoMUkmv6e2oM2tVrgnT7gdRG1"
/>
    </div>
  </form>
</body>

</html>
```

With the exception of the view state values, the output is standard XHTML. Earlier versions of ASP.NET did not output standards-compliant HTML. Your form is now complete and ready to add processing. Since Expression Web does not have tools for creating form–to–e-mail processing natively, the code you will need to add is not part of this exercise.

Unlike a classic ASP or PHP form, you do not send an ASP.NET form to a separate page for processing. Instead, you have two methods to choose from to send your form results by e-mail. Usually, this is accomplished by a postback condition where the ASP.NET process runs only when the form submits back to itself. You can also send form results using a separately compiled code-behind page created in Visual Studio or Visual Web Developer Express that is specified in an `@ Page` directive. After this the code, whether it is on the web page or contained in a code-behind page, processes the form results.

Code-behind pages are created in Visual Studio or Visual Web Developer Express and are beyond the scope of this book. However, basic form–to–e-mail functionality using ASP.NET code is included in the final ASP.NET page in this chapter. Since we will not be sending the visitor to another page after the form is processed, another label field will be added to the form to display our thank you message. The forms-processing script is compatible with all versions of ASP.NET 1.0, 1.1, and 2.0.

If you are using this contact form, you will need to add another label box to your web page. I have added it just above the first line of form fields with the label ID of `lblresponse` to hold the confirmation message. To keep our form from shifting after the form is submitted I added the following lines:

```
<h3>Contact Us</h3>.
<h3><asp:Label ID="lblResponse" runat="server">
Please provide the following information:</asp:Label></h3>
```

In Chapter 13, we will add validation and error checking to the forms processor.

SENDING E-MAIL FROM YOUR DOMAIN

Before you use any form to send e-mail from your website, you must check with your web host. The ASP and PHP form-processing examples will send you the e-mail results as from the person who filled in the form information. Some web hosts will only allow forms hosted on their sites to send with a "from" address that is a valid e-mail address on the domain sending the form. If this is the case, you will need to modify the scripts or use a script provided by your web host.

The following ASP.NET script sends from the webmaster at your domain account and puts the visitor's name and e-mail in the body of the message. Therefore, you cannot click to reply, but the script will work on most web servers that require your forms to use a local account.

```
<%@ Page Language="VB" %>
<%@ Import Namespace="System.Web.Mail" %>
<script runat=server>
Sub Page_Load(Sender as Object, E as EventArgs)
   If Page.IsPostBack Then
       lblResponse.Text = "Your message has been sent."
   End If
End Sub
```

```
Sub doEmail(Source as Object, E as EventArgs)
  Dim sMsg as String
  sMsg+="Submission from domain contact form:" & vbcrlf
  sMsg+="Name : " & txtname.Text & vbcrlf
  sMsg+="Message : " & txtmessage.Text & vbcrlf

  Dim objEmail as New MailMessage
  objEmail.To= "you@domain.com"
  objEmail.FROM="webmaster@domain.com"

  objEmail.SUBJECT="From domain website" & vbcrlf
  objEmail.BODY=sMsg
  objEmail.BodyFormat = MailFormat.Text
  SmtpMail.SmtpServer ="localhost"
  SmtpMail.Send(objEmail)
End Sub
</script>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html dir="ltr" xmlns="http://www.w3.org/1999/xhtml">
  <head runat="server">
  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
  <title>ASP.NET form</title>
</head>

<body>
  <h3>Contact Us</h3>.
  Please provide the following information:</asp:Label></h3>
  <form id="form1" runat="server">
    <h3><asp:Label ID="lblResponse" runat="server">
    <asp:Label runat="server" Text="Name: " id="lblsname"
               AssociatedControlID="txtname"></asp:Label>
    <asp:TextBox runat="server" id="txtname"></asp:TextBox>
    <br />
    <asp:Label runat="server" Text="Email: " id="lblemail"
               AssociatedControlID="txtemail"></asp:Label>
    <asp:TextBox runat="server" id="txtemail"></asp:TextBox>
    <br />
    <asp:Label runat="server" Text="Message: " id="lblmessage"
               AssociatedControlID="txtmessage"></asp:Label>
    <asp:TextBox runat="server" id="txtmessage" Width="300px" Rows="10"
                 TextMode="MultiLine"></asp:TextBox>
    <br />
    <asp:Button runat="server" text="Submit" id="subimit" />
  </form>
</body>

</html>
```

When your form is submitted using the submit button, the visitor will see the contents of the `lblresponse` field on the page. You can add the same style definitions as we used in the HTML form by creating a stylesheet called `form.cs`, attaching it to `form1.html`, and dragging the styles to the new stylesheet. Because ASP.NET uses IDs for programming purposes, it is generally better to use classes or contextual selectors to style your ASP.NET controls instead of individual IDs. Associating an ASP.NET form control with a CSS class must be done in the Tag Properties dialog, not by double-clicking the style name as you would with an HTML form element. Once your stylesheet is attached, you can use the `CssClass` drop-down field to choose the class you wish to apply to the page, as shown in Figure 10-35.

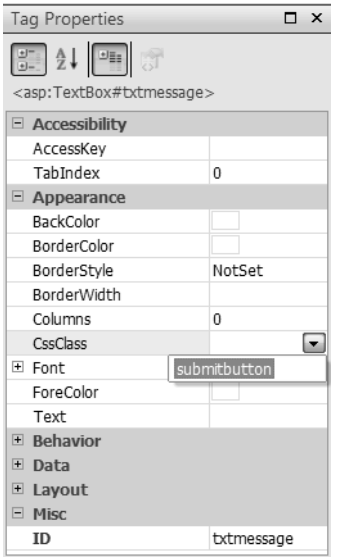

Figure 10-35. *Associating a class with an ASP.NET control*

Summary

In this chapter, you learned how to create a form, what the HTML form elements and their affiliated ASP.NET form controls do, and how to use Expression Web to create a simple contact form using both HTML and ASP.NET. You have been provided with our basic form script, in three languages, as well as instructions for using FrontPage Server Extensions to send the form's results by e-mail. All forms follow the same principals, and there are many forms-processing solutions available to you. In Chapter 13, we will add form field validation for our ASP.NET form and notify our visitor if the form is unable to send the e-mail. We will also password protect a section of your website. In Chapter 11, before we move to more ASP.NET when we use Master Pages, we will address some of the usability and legal issues that you need to be aware of as a website designer.