CHAPTER 13

Beyond the Basics in CSS and Scripting

In Chapters 5, 6, and 7, you learned how to use CSS to make your website attractive and easy to maintain with styles. In this chapter, we will look at styling your content for different media. In several places in this book, I have pointed out that you have more control over how pages display when printed than over how they display on the screen. Using a print media stylesheet allows you to specify how a page should print. You can reformat a page so that only the parts of the page you choose are printed and so that you use print measurements, such as margins in inches or centimeters and type in points.

In Chapter 10, we created two contact forms. The first was a standard HTML form and the second used ASP.NET. In both cases, the form we created was simple, and the visitor could submit the form without filling in any of the form fields. Having empty forms submitted is usually not desirable. You can prevent form fields from being submitted without content by validating your form before submitting the form for processing. ASP.NET has built-in validation controls, which you can easily use in Expression Web, but these controls are only available if you have ASP.NET on your web server.

In this chapter, we will add form field validation to our ASP.NET forms using the built-in validation controls. For those who do not have ASP.NET, simple form–to–e-mail scripts were provided in Chapter 10, and we will add simple JavaScript code to ensure that these HTML forms have content before they are submitted.

This chapter will conclude by using ASP.NET to password protect a folder on our website. The Appendix will look at some of the third-party add-ins available for Expression Web.

Alternative Stylesheet Types

In this book, we have been creating one stylesheet that will be used whether the visitor is reading the page on a conventional web browser, a pocket PC, a cell phone, a printout, or with some sort of assistive technology such as a screen reader. This is not the only way you can use stylesheets. You can serve a different stylesheet depending on what type of device is being used to access your web page.

By using media stylesheets, you can target specific devices or methods of viewing without having to change the HTML. You can control how a page will be rendered or viewed using the following media types:

- `all`: This is the default type applied to all media if you do not specify a media type.
- `aural`: This media type is intended for use with speech synthesizers or screen readers but is supported by only a few screen readers. See the W3C section on aural stylesheets at `http://www.w3.org/TR/REC-CSS2/aural.html` for details.
- `braille`: This rarely used media type sends instructions to tactile Braille feedback devices.
- `embossed`: Use this media type for Braille printers.
- `handheld`: For PDAs, some of the new smart phones, and other devices with small screens and limited bandwidth, use this media type.
- `print`: Stylesheets for printing are the best supported media type. This type of stylesheet provides instructions for printers and uses measurements appropriate for printing. More information is available on the W3C paged media section: `http://www.w3.org/TR/REC-CSS2/page.html`.
- `projection`: For projected presentations, such as projectors or print to transparencies, use this media type.
- `screen`: This type is best suited for computer screens using a web browser.
- `tty`: For media using a fixed-pitch character grid, such as teletypes, terminals, or portable devices with limited display capabilities, use the `tty` type. Devices that use the `tty` media type generally do not support pixels as a unit of measurement.
- `tv`: Use the `tv` type for television-type devices, such as MSNTV; a game console, such as Xbox or PlayStation; or other devices using low-resolution, limited scroll ability screens, such as traditional CRT televisions.

Media type names are case insensitive, but if you are using XHTML, it is case sensitive, so lowercase names is a good practice. Note that not all media types are well supported.

Applying Different Media Types

You may write styles for various media in the `<head>` section by specifying the `media` attribute in the `<style>` element as follows:

```
<style type="text/css" media="tty">
<style type="text/css" media="tv">
<style type="text/css" media="projection">
<style type="text/css" media="handheld">
<style type="text/css" media="print">
<style type="text/css" media="braille">
<style type="text/css" media="aural">
<style type="text/css" media="all">
<style type="text/css" media="screen">
```

If you just want to have one or two styles that would apply to a specific media style, you can use the `@media` rule. You use this rule to define different types of stylesheets as follows:

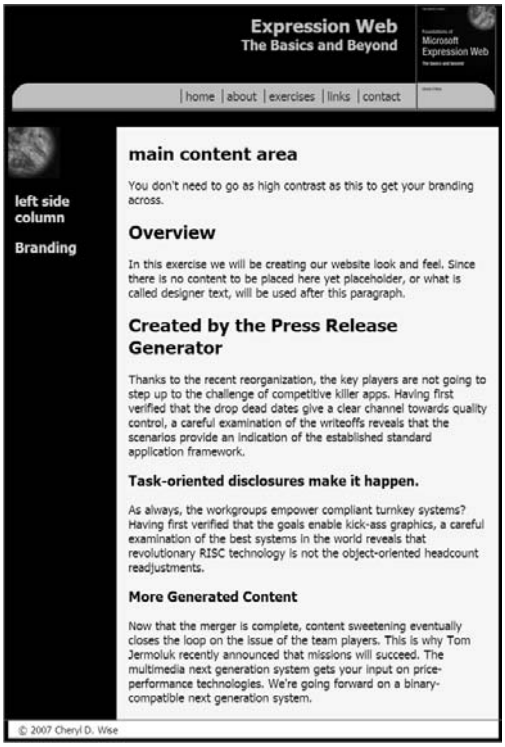
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Media Example</title>
  <style type="text/css">
  @media screen {
    p {
      font-size: small;
      color: blue;
      font-family: Tahoma, Arial, sans-serif
    }
    body {
      margin: 0;
      padding: 10px;
    }
  }
  @media print {
    p {
      font-size: 10pt;
    }
    body {
      font-family: "Times New Roman", Times, serif;
      margin: 1in; color: black;
    }
  }
  @media projection {
    p {
      font-size: x-large;
      color: red;
      font-family: Arial, Helvetica, sans-serif;
    }
  }
  </style>
</head>
<body>
  <h1>Altering the Paragraph Tag for Media</h1>
  <p>This paragraph should appear blue on the screen with small text
  and 10px margins.</p>
  <p>When viewed on a projector the text would be larger and red with
  the default browser margins. </p>
  <p>For print the page should have 1inch margins and 10pt type with
  black type. </p>
</body>
</html>
```

An alternative method of adding media styles is to use linked stylesheets with the `media` attribute:

```
<link rel="stylesheet" media="aural" href="aural.css" type="text/css">
<link rel="stylesheet" media="screen" href="screen.css" type="text/css">
<link rel="stylesheet" media="print" href="print.css" type="text/css">
<link rel="stylesheet" href="site.css" type="text/css">
```

Print Stylesheets

A separate style for printing is the most commonly used media stylesheet. Pages generally contain elements that are not necessary when you are printing a page. After all, a menu of hyperlinks does not help when you are viewing a printed page, and decorative images that would waste ink are another item you might prefer not to print. In addition, a printed page using measurements appropriate to print, such as points and margins in inches instead of pixels or ems, may be a better option. In Chapter 9, we created a web page that looks like Figure 13-1 when viewed in a browser.



main content area

You don't need to go as high contrast as this to get your branding across.

Overview

In this exercise we will be creating our website look and feel. Since there is no content to be placed here yet placeholder, or what is called designer text, will be used after this paragraph.

Created by the Press Release Generator

Thanks to the recent reorganization, the key players are not going to step up to the challenge of competitive killer apps. Having first verified that the drop dead dates give a clear channel towards quality control, a careful examination of the writeoffs reveals that the scenarios provide an indication of the established standard application framework.

Task-oriented disclosures make it happen.

As always, the workgroups empower compliant turnkey systems? Having first verified that the goals enable kick-ass graphics, a careful examination of the best systems in the world reveals that revolutionary RISC technology is not the object-oriented headcount readjustments.

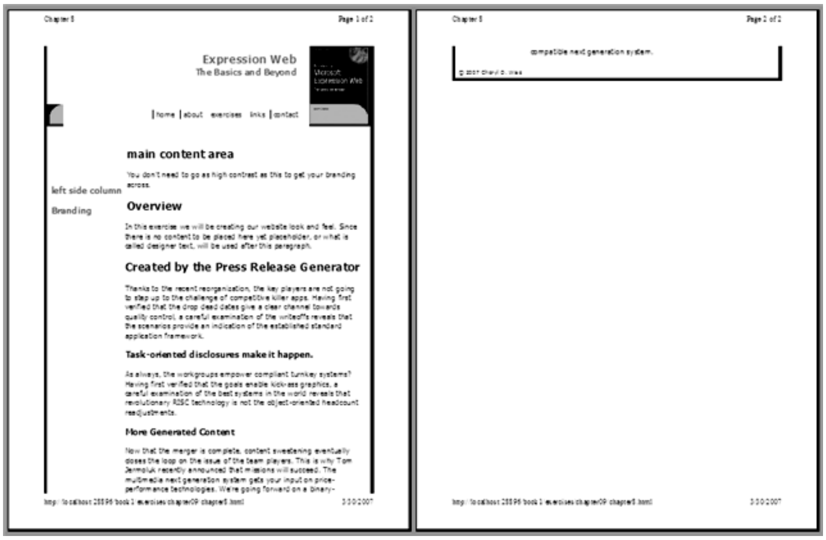
More Generated Content

Now that the merger is complete, content sweetening eventually closes the loop on the issue of the team players. This is why Tom Jermoluk recently announced that missions will succeed. The multimedia next generation system gets your input on price-performance technologies. We're going forward on a binary-compatible next generation system.

© 2007 Cheryl D. Wise

Figure 13-1. *A web page viewed in a browser without a print stylesheet applied.*

You would think the page shown in Figure 13-1 would print on one page, but as Figure 13-2 shows, without a print stylesheet the web page would equal two printed pages, and since background images do not print, the page is not attractive when printed.



Chapter 8 Page 1 of 2

Expression Web
The Basics and Beyond

| home | about exercises links | contact

main content area

You don't need to go as high contrast as this to get your branding across.

left side column

Branding

Overview

In this exercise we will be creating our website look and feel. Since there is no content to be placed here yet placeholder, or what is called designer text, will be used after this paragraph.

Created by the Press Release Generator

Thanks to the recent reorganization, the key players are not going to step up to the challenge of competitive killer apps. Having first verified that the drop dead dates give a clear channel towards quality control, a careful examination of the writeoffs reveals that the scenarios provide an indication of the established standard application framework.

Task-oriented disclosures make it happen.

As always, the workgroups empower compliant turnkey systems? Having first verified that the goals enable kick-ass graphics, a careful examination of the best systems in the world reveals that revolutionary RISC technology is not the object-oriented headcount readjustments.

More Generated Content

Now that the merger is complete, content sweetening eventually closes the loop on the issue of the team players. This is why Tom Jermoluk recently announced that missions will succeed. The multimedia next generation system gets your input on price-performance technologies. We're going forward on a binary-

http://localhost:28896/book1/exercises/chapter09/chapter8.html 3/30/2007

Chapter 8 Page 2 of 2

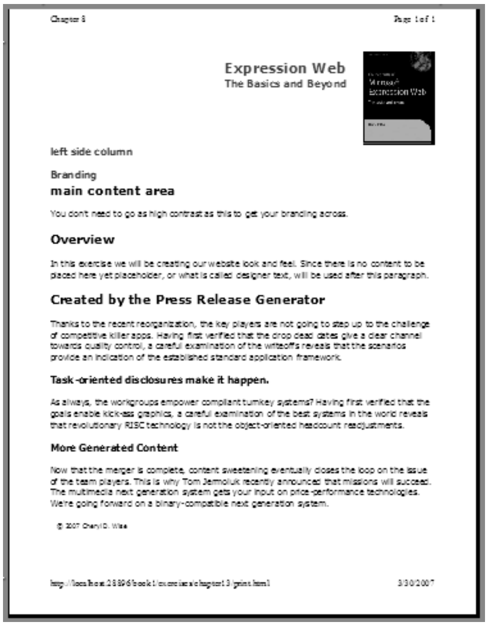
compatible next generation system.

© 2007 Cheryl D. Wise

http://localhost:28896/book1/exercises/chapter09/chapter8.html 3/30/2007

Figure 13-2. *Without a print stylesheet, the web page prints on two pages and without background images. A single stylesheet can lead to pages that do not print properly.*

However, if we add a `media="print"` stylesheet that removes the navigation and makes the content of the side menu full width, the page will print as shown in Figure 13-3, which not only looks better but also fits on one page.



Chapter 8 Page 1 of 1

Expression Web
The Basics and Beyond

left side column

Branding

main content area

You don't need to go as high contrast as this to get your branding across.

Overview

In this exercise we will be creating our website look and feel. Since there is no content to be placed here yet placeholder, or what is called designer text, will be used after this paragraph.

Created by the Press Release Generator

Thanks to the recent reorganization, the key players are not going to step up to the challenge of competitive killer apps. Having first verified that the drop dead dates give a clear channel towards quality control, a careful examination of the writeoffs reveals that the scenarios provide an indication of the established standard application framework.

Task-oriented disclosures make it happen.

As always, the workgroups empower compliant turnkey systems? Having first verified that the goals enable kick-ass graphics, a careful examination of the best systems in the world reveals that revolutionary RISC technology is not the object-oriented headcount readjustments.

More Generated Content

Now that the merger is complete, content sweetening eventually closes the loop on the issue of the team players. This is why Tom Jermoluk recently announced that missions will succeed. The multimedia next generation system gets your input on price-performance technologies. We're going forward on a binary-compatible next generation system.

© 2007 Cheryl D. Wise

http://localhost:28896/book1/exercises/chapter13/print.html 3/30/2007

Figure 13-3. *The same page with a print stylesheet attached*

The stylesheets are attached in the following order:

```
<link rel="stylesheet" type="text/css" href="few-site.css" />
<link rel="stylesheet" type="text/css" href="print.css" media="print" />
```

The styles from the main stylesheet that we will be changing in the print stylesheet are as follows:

```
#masthead h1 {
  padding: .5em 1em;
  margin: 0;
  font-size: 1.5em;
}
.tagline {
  font-size: .75em;
}
#menu {
  background: url('images/menu-bg.gif') repeat-x bottom #FDB928;
  height: 45px;
}
#container {
  border-right: solid 6px #000000;
  border-left: solid 6px #000000;
  padding: 0;
  margin: 0;
  background-color: #000000;
  color: #CEDBAE;
}
#leftcol {
  background: #000000 url('images/book-corner.jpg') no-repeat;
  float: left;
  width: 150px;
  padding: 100px 4px 10px 10px;
  font-size: .88em;
}
#main_content {
  padding: .1em 1em;
  margin: 0 0 0 165px;
  background-color: #F4F5EF;
  color: #000000;
}
#footer {
  margin: 0px;
  padding: .1em 1em;
  font-size: .8em;
  border: solid 1px #666666;
}
```

The new style definitions in our print stylesheet are

```
body {
  font-size: 12pt;
}
#masthead h1 {
  font-size: 20pt;
}
.tagline {
  font-size: 14pt;
}
#menu {
  width: 0;
  height: 0;
  overflow: hidden;
}
#container {
  border: none;
}
#leftcol {
  width: 100%;
  padding: 0;
  font-size: 1em;
}
#main_content {
  padding: 0;
  margin: 0;
}
#footer {
  font-size: 9pt;
  border: none;
}
```

Changes to the Print Layout

By changing the menu to have a height and width of 0 with overflow hidden, all browsers will collapse the space, and your menu will not print. Setting the width of the left column to 100 percent takes the content to the full width of the printed pages. The last adjustment to the page layout was to remove the left margin from the `maincontent` `<div>` so that it would also print the full width of the page. All of the borders were removed by using `border: none` for each of the divs where borders were specified in the original stylesheet.

Changes to Text

A print font size of 12pt was specified as the default, and the left-side text was reset from .88em, which would be just under 11pt text, to the same size as the main content text. All other font sizes were changed from resizable screen measurements to points as well. The `<h1>` and tagline text were changed from orange to black text to print better on white paper and avoid wasting color ink. Remember that background colors and images do not print by default. This means you should either use print preview in your browser or actually print the web pages that you believe your visitors may want to print. Examples of pages that are commonly printed are directions, receipts, form confirmation, product pages, and price sheets.

You should use print stylesheets whenever you expect your pages to be printed.

Validating your Forms

Form validation is used to make sure that your visitor has completed the form fields needed for you to respond to a request or fulfill the purpose of the form and is something many people want to do before a form is submitted. There are two basic types of form validation. The first is client-side validation, which happens before the form is submitted to the form's processing script. The second happens on the server before the form is processed and sends an e-mail or performs another action.

Client-side validation takes place in the visitor's browser, which makes it very fast and allows you to provide immediate feedback to a visitor who did not fill in required information or formatted the information incorrectly. This method requires JavaScript to be enabled in the visitor's browser. While the vast majority of web surfers (90 percent is the generally accepted figure as provided by `http://www.w3schools.com/browsers/browsers_stats.asp` and `thecounter.com`) do have JavaScript available in their browsers, a significant number of visitors may still submit forms without having the form field contents validated.

As a result, many people use both client-side and server-side form validation. ASP.NET provides both by default: it adds JavaScript validation to your forms, but if validation fails there because of a lack of JavaScript support, the form field contents are checked on the server before being submitted for processing.

ASP.NET Form Validation Controls

ASP.NET validation controls are found in the Toolbox. To use the validation controls, expand the Validation section of the ASP.NET Controls in the Toolbox task pane, as shown in Figure 13-4.

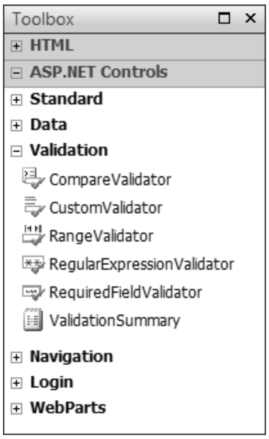


Figure 13-4. *Available ASP.NET validation controls*

The control you should use depends on what type of validation your form fields need. Available controls are as follows:

- `CompareValidator` control compares the form field entry against a preset value or against the value of another control using less than, equals, or greater than. Common uses of this validator are checking that two e-mail form fields contain the same value or are of a specific data type.
- `CustomValidator` control checks the form field entry against code that you write yourself.
- `RangeValidator` control checks that a visitor's entry is between a minimum and a maximum pair of numbers, alphabetic characters, or dates.
- `RegularExpressionValidator` control checks that the form field contents match a pattern defined by a regular expression. This allows you to check for predictable sequences of characters, such as those in e-mail addresses, telephone numbers, or zip and postal codes.
- `RequiredFieldValidator` control prevents the form from being submitted when the form fields are empty.
- `ValidationSummary` control does not validate form fields but can be used with the other validation controls to display the error messages from all the validation controls on the page in one block instead of next to the fields being validated.

ASP.NET Validation Controls Tag Properties

Before you can use the ASP.NET validation controls, you must have an ASP.NET form with ASP.NET form fields and labels. In this section, I will be using the `form.aspx` page we created in Chapter 10 with the `forms.css` stylesheet applied.

With each of the ASP.NET validation controls, the values and properties are set in the Tag Properties task pane. There are six sections of properties you can set for each control. They are as follows:

- *Accessibility*, shown in Figure 13-5, allows you to set an access key and tab index for your form fields or validation controls. To avoid conflicts with the user's settings, do not set either of these properties.

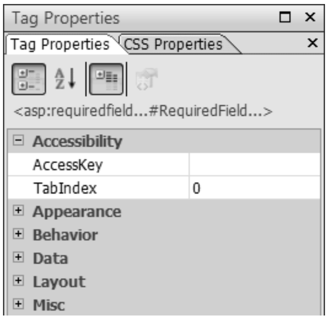
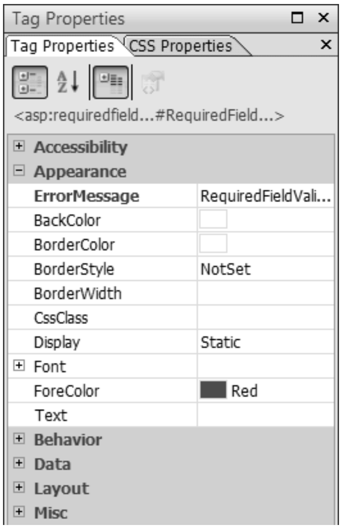


Figure 13-5. *ASP.NET validation controls Accessibility properties*

- *Appearance*, shown in Figure 13-6, will be the same on every validation control with the exception of the `ValidationSummary` control, which will display the `ErrorMessage` property of the individual validation types. By default, the error messages are red when displayed. You can change the color using the `ForeColor` property, but if you wish to change the default display, it is best to create a CSS class and apply it to the error message using the `CssClass` property. For all of the validation controls except `ValidationSummary`, you must replace the default error message in the `ErrorMessage` property.

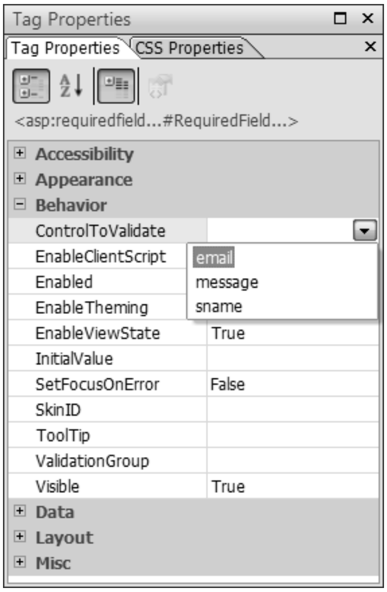


<asp:requiredfield...#RequiredField...>

⊞ Accessibility	
⊟ Appearance	
ErrorMessage	RequiredFieldVali...
BackColor	
BorderColor	
BorderStyle	NotSet
BorderWidth	
CssClass	
Display	Static
⊞ Font	
ForeColor	■ Red
Text	
⊞ Behavior	
⊞ Data	
⊞ Layout	
⊞ Misc	

Figure 13-6. *Replace the default error message with one appropriate for your form field.*

- *Behavior* is where you set properties that differ among the types of validation controls. Later in this section, we will be looking at each of the validation controls individually. However, the following properties in the Behavior section remain constant among the validation controls:
  - `ControlToValidate`: Here, you select the control to be validated on each of the validation controls except for the `ValidationSummary` control. Use the drop-down list to make sure you get the correct field, as shown in Figure 13-7.
  - `Enabled`: The default is True. If you change this to False, the control will not function.
  - `EnableTheming`: The default is True. This allows you to apply a Visual Studio theme to your site.
  - `EnableViewState`: The default is True. Do not change the default if you want to allow the server to keep track of the status of the validation field.
  - `InitialValue`: This property is blank and should be left as such.



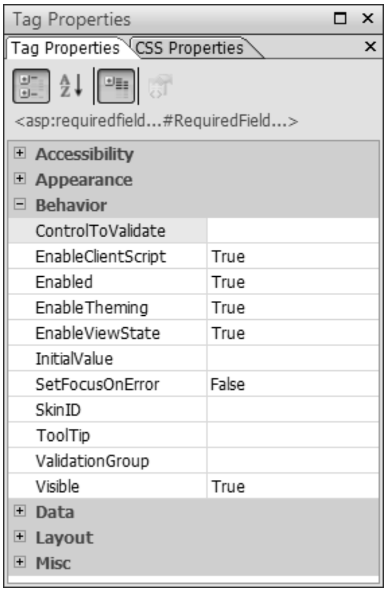
<asp:requiredfield...#RequiredField...>

Accessibility
Appearance
Behavior

ControlToValidate	
EnableClientScript	email
Enabled	message
EnableTheming	sname
EnableViewState	True
InitialValue	
SetFocusOnError	False
SkinID	
ToolTip	
ValidationGroup	
Visible	True

Data
Layout
Misc

Figure 13-7. *All of the form controls available on the page will be available in the drop-down list.*



<asp:requiredfield...#RequiredField...>

Accessibility
Appearance
Behavior

ControlToValidate	
EnableClientScript	True
Enabled	True
EnableTheming	True
EnableViewState	True
InitialValue	
SetFocusOnError	False
SkinID	
ToolTip	
ValidationGroup	
Visible	True

Data
Layout
Misc

Figure 13-8. *Behavior options for Validation Controls*

- `SetFocusOnError`: The default is False. If you set this to True, when the validation routine finds an error, it will take the visitor directly to the error message instead of going to the top of the page. When you use the `ValidationSummary` control, you should set this property to True.
- `SkinID`: This property allows you to associate the control with a skin or theme ID. As with the `EnableTheming` property, you can safely leave the defaults alone. Skins and themes are created in Visual Studio.
- `ToolTip`: This sets the `title` attribute to provide additional information if the visitor mouses over the control.
- `ValidationGroup`: By default, this property is blank. Leave it that way.

- *Data* contains only one property, `Expressions`, which would be used if you were connecting to a database or other data source to validate against (see Figure 13-9).

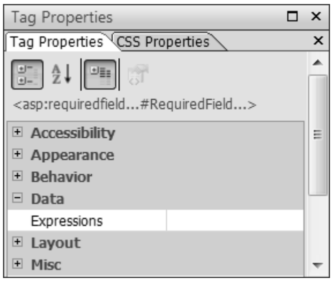


Figure 13-9. *The only Data option for validation controls is Expressions.*

- *Layout*, shown in Figure 13-10, has two properties: `Height` and `Width`, which should be contained in a CSS class instead of being set individually for each control.

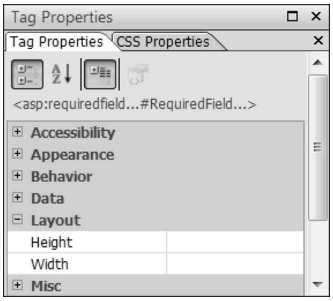
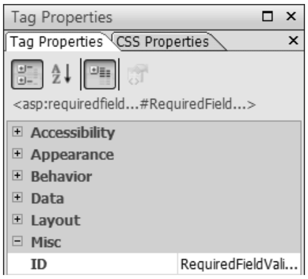


Figure 13-10. *Layout options*

- *Misc* has only one property—`ID`, as shown in Figure 13-11. You should change its name to a more meaningful one than the default, which is the type of validation control. As you add more validation controls of the same type to your page, a number that increments will be added to the default to avoid duplicate names.



ID	RequiredFieldVali...

Figure 13-11. *Change the default name for the control ID.*

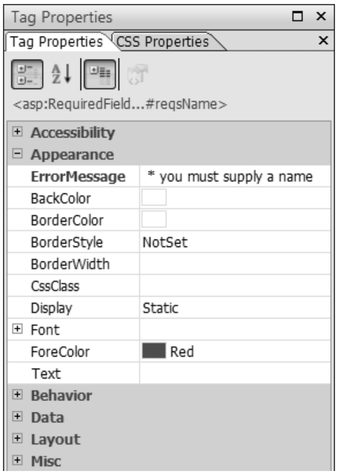
■ **Tip** Decide on a naming convention and stick with it for ease of maintenance.

Individual Validation Control Properties

ASP.NET has a variety of validation controls available. You can check if the form field is empty or has content. You can also check if the information is the correct type, such as a date or e-mail address. In addition, you can determine that the content matches either another form field or a static value. Each of the six validation controls are addressed individually in the following sections.

RequiredFieldValidator

The `RequiredFieldValidator` does exactly what its name says—it checks for content in the form field before the form is submitted for processing. It does not check the contents of the form field, only whether there is something in the field. Make sure that you provide a meaningful error message, as shown in Figure 13-12.



ErrorMessage	* you must supply a name
BackColor	
BorderColor	
BorderStyle	NotSet
BorderWidth	
CssClass	
Display	Static
Font	
ForeColor	■ Red
Text	

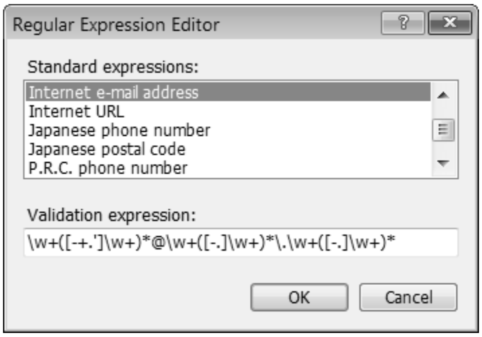
Figure 13-12. *Remember to insert the error message for each validation control.*

RegularExpressionValidator

A regular expression is a string that describes or matches a set of strings, according to certain syntax rule; it's similar to using wild cards to match text. Expression Web includes some of the common regular expressions used to validate form information on the Web. Use the `RegularExpressionValidator` to ensure that any e-mail address submitted using the contact form is in the proper format. In addition to e-mail addresses, you can validate URLs, phone number formats used in various countries, postal or zip codes, and other common form data that follows a regular pattern.

■ **Note** You can only validate the form of the e-mail. You can not use the regular expression validation control to confirm that the address is a real address. The only reliable way to validate a live e-mail address is to send an e-mail and get a response from the e-mail account holder.

To use the `ValidationExpression` control, click once to bring up the three dots in the `ValidationExpression` property in the Tag Properties task pane. Click again to launch the Regular Expression Editor shown in Figure 13-13.



Validation expression:

```
\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*
```

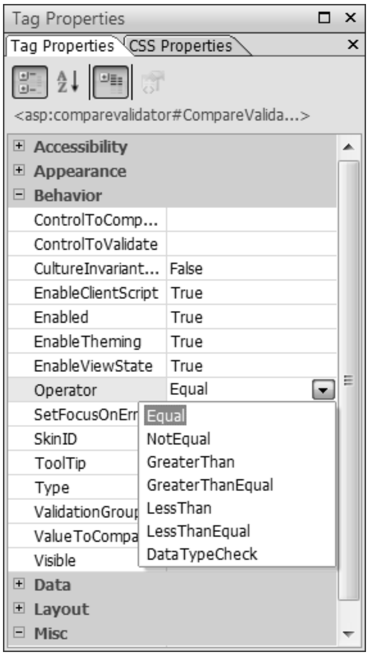
Figure 13-13. *Many commonly used validation expressions are available from the Regular Expression Editor.*

■ **Tip** You can have more than one validation control on a form field. In the case of our simple form, we may want to use the `RequiredFieldValidator` to make sure that the e-mail field is not empty.

Compare Validator

The compare validator compares the contents of the form field to either the contents of another field or a value you supply.

- To compare one field with another in the same form, select the form field for comparison from `ControlToCompare` property.
- To compare the `ControlToValidate` field with a value specified in the `ValueToCompare` property, choose the type of comparison to make from the Operator drop-down as shown in Figure 13-14.



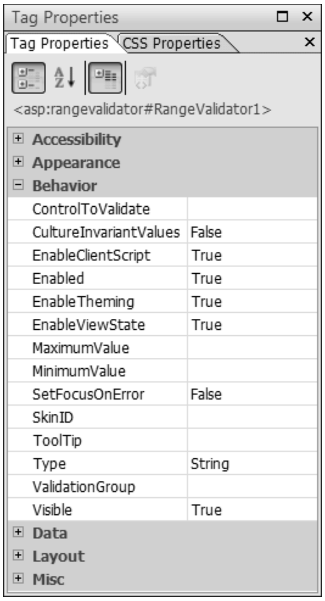
ControlToComp...	
ControlToValidate	
CultureInvariant...	False
EnableClientScript	True
Enabled	True
EnableTheming	True
EnableViewState	True
Operator	Equal
SetFocusOnErr	
SkinID	
ToolTip	
Type	
ValidationGroup	
ValueToCompa	
Visible	

Equal
NotEqual
GreaterThan
GreaterThanEqual
LessThan
LessThanEqual
DataTypeCheck

Figure 13-14. *Select the two form fields you wish to compare.*

Required Range Validator

You would use a required range validator to place an upper and lower limit, such as a minimum and maximum order quantity or a date range, as shown in Figure 13-15. In our simple form example, there is no field that would be appropriate for the required range validator.



<asp:rangevalidator#RangeValidator1>

⊞ Accessibility	
⊞ Appearance	
⊟ Behavior	
ControlToValidate	
CultureInvariantValues	False
EnableClientScript	True
Enabled	True
EnableTheming	True
EnableViewState	True
MaximumValue	
MinimumValue	
SetFocusOnError	False
SkinID	
ToolTip	
Type	String
ValidationGroup	
Visible	True
⊞ Data	
⊞ Layout	
⊞ Misc	

Figure 13-15. *Use a minimum and maximum value for the form field contents.*

Validation Summary

Up until this point, each of the validation controls we have been adding is displayed where the control is placed on the page. While having the error message next to the form fields tells the visitor which form field was missed or incorrectly filled out, this can lead to problems with page display not only in Expression Web, as shown in Figure 13-16, but also in the browser even before the error messages are displayed, as shown in Figure 13-17.

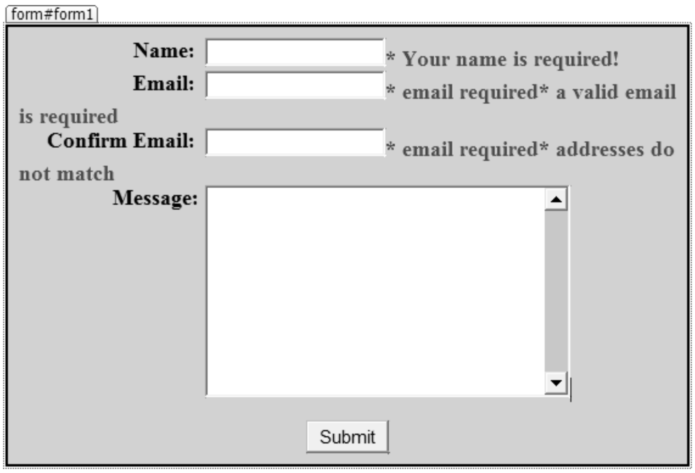

form#form1
Name:
* Your name is required!
Email:
* email required* a valid email is required
Confirm Email:
* email required* addresses do not match
Message:
Submit

Figure 13-16. *Form with validation in Design view*

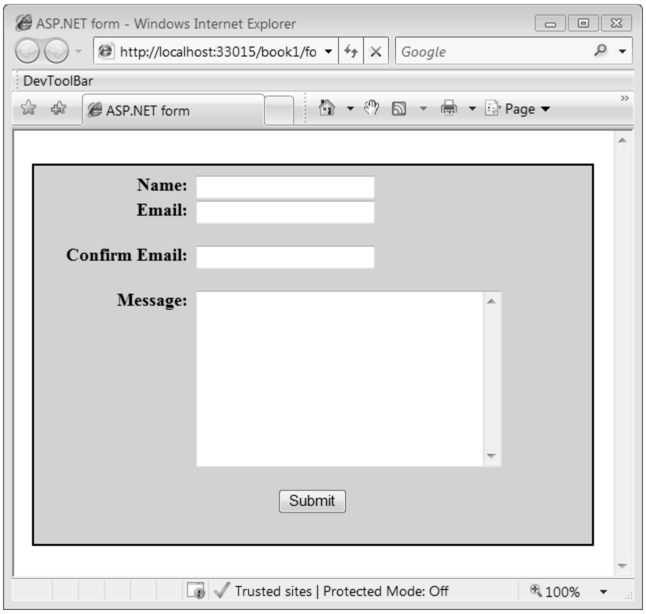
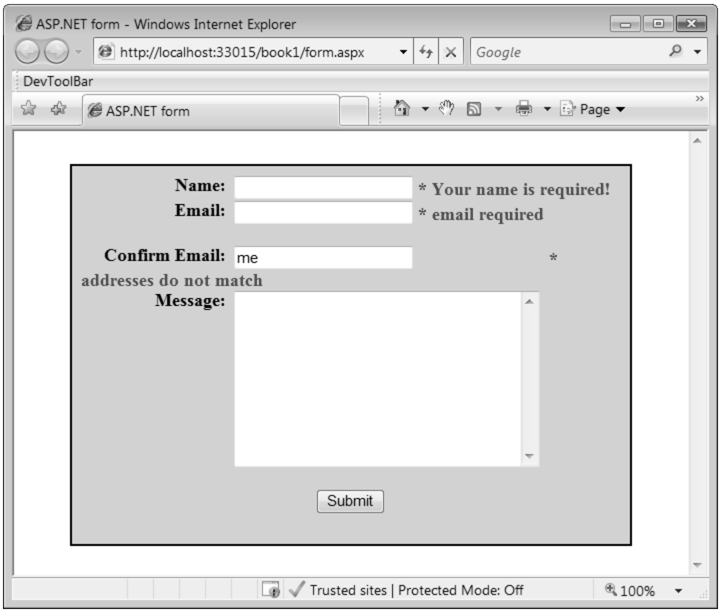

ASP.NET form - Windows Internet Explorer
http://localhost:33015/book1/fo
Google
DevToolBar
ASP.NET form
Page
Name:
Email:
Confirm Email:
Message:
Submit
Trusted sites | Protected Mode: Off
100%

Figure 13-17. *The same form displayed in Internet Explorer 7*

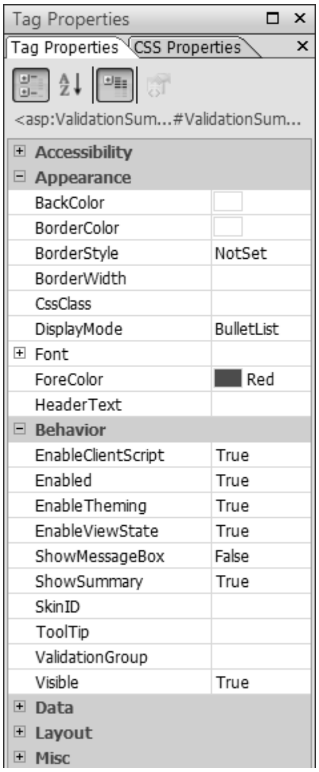
The gaps you see in Figure 13-17 are where the validation messages take up space but are not displayed until the form is submitted. When the form is submitted, depending on which error message is triggered, the display may look even stranger, as shown in Figure 13-18.


ASP.NET form - Windows Internet Explorer
http://localhost:33015/book1/form.aspx
Google
DevToolBar
ASP.NET form
Page
Name:
Email:
Confirm Email:
Message:
Submit
Trusted sites | Protected Mode: Off
100%

* Your name is required!
* email required
me
* addresses do not match

Figure 13-18. *When the second validation error message displays, there may be mysterious gaps.*

You can prevent this sort of unattractive display by using the validation summary control, the properties of which are shown in Figure 13-19. The summary can be displayed as a list, a bulleted list, or a single paragraph, based on the `DisplayMode` property. The summary will be displayed on the web page as a bulleted list without changing the defaults.



Accessibility	
Appearance	
BackColor	
BorderColor	
BorderStyle	NotSet
BorderWidth	
CssClass	
DisplayMode	BulletList
Font	
ForeColor	■ Red
HeaderText	
Behavior	
EnableClientScript	True
Enabled	True
EnableTheming	True
EnableViewState	True
ShowMessageBox	False
ShowSummary	True
SkinID	
ToolTip	
ValidationGroup	
Visible	True
Data	
Layout	
Misc	

Figure 13-19. *The default provides a bulleted list and an error message where the control is located.*

Exercise 13-1. Using ASP.NET Form Validation

In Chapter 10, we created a simple form with four ASP.NET form controls. In this exercise, you will add validation for each of the form fields and use the `ValidationSummary` control to display the error messages. The form code before we begin is as follows:

```
<!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">
```

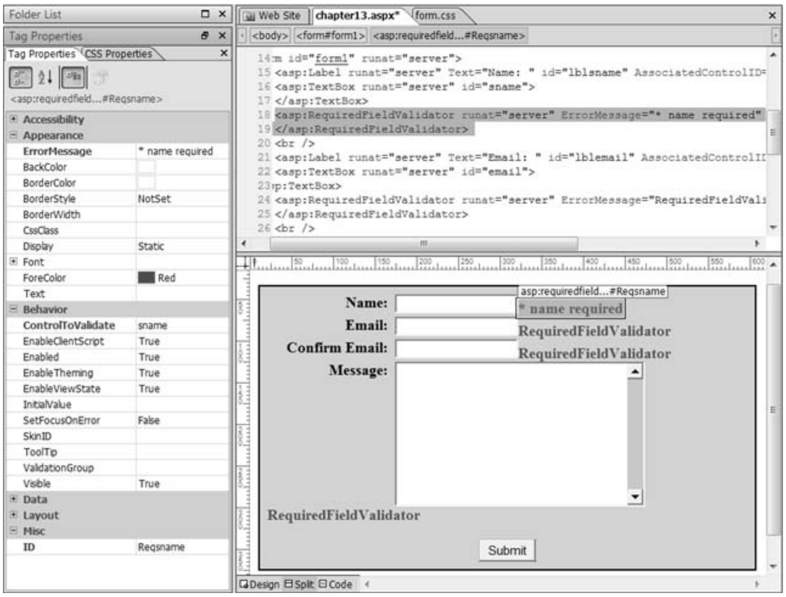
```
<head runat="server">
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Chapter 13</title>
<link rel="stylesheet" type="text/css" href="form.css" />
</head>

<body>
<form id="form1" runat="server">
   <asp:Label runat="server" Text="Name: " id="lblsname"
   AssociatedControlID="sname"></asp:Label>
   <asp:TextBox runat="server" id="sname">
   </asp:TextBox>
   <br />
   <asp:Label runat="server" Text="Email: " id="lblemail"
   AssociatedControlID="email"></asp:Label>
   <asp:TextBox runat="server" id="email">
</asp:TextBox>
   <br />
   <asp:Label runat="server" Text="Confirm Email: " id="lblconfirmemail"
   AssociatedControlID="email"></asp:Label>
   <asp:TextBox runat="server" id="confirmemail">
</asp:TextBox>
   <br />
   <asp:Label runat="server" Text="Message: " id="lblmessage"
   AssociatedControlID="message"></asp:Label>
   <asp:TextBox runat="server" id="message" Width="300px" Rows="10"
   TextMode="MultiLine"></asp:TextBox>
   <p class="submit">
   <asp:Button runat="server" text="Submit" id="subimit" /></p>
</form>
</body>
</html>
```

The stylesheet referenced in the HTML code for this exercise is the same one created in Chapter 10 and is optional. If you do not use that stylesheet, your form will not be inside the shaded area shown in this exercise's screen images.

Before you begin adding validation controls, you need to add a label and form field for a second e-mail address just below the existing e-mail form field so that you can compare the e-mail address to check for entry errors.

```
   <asp:Label runat="server" Text=" Confirm Email: " id="lblconfirmemail"
   AssociatedControlID="email"></asp:Label>
   <asp:TextBox runat="server" id="confirmemail">
   </asp:TextBox>
 <br />
```

1. Begin by dragging a `RequiredFieldValidator` control out to the right of each of your form fields, as shown in Figure 13-20.



```
m id="form1" runat="server">
<asp:Label runat="server" Text="Name: " id="lblsname" AssociatedControlID=
<asp:TextBox runat="server" id="sname">
</asp:TextBox>
<asp:RequiredFieldValidator runat="server" ErrorMessage="* name required"
</asp:RequiredFieldValidator>
<br />
<asp:Label runat="server" Text="Email: " id="lblemail" AssociatedControlII
<asp:TextBox runat="server" id="email">
p:TextBox>
<asp:RequiredFieldValidator runat="server" ErrorMessage="RequiredFieldVal:
</asp:RequiredFieldValidator>
<br />
```

<asp:requiredfield...#Reqsname>

Accessibility	
Appearance	
ErrorMessage	* name required
BackColor	
BorderColor	
BorderStyle	NotSet
BorderWidth	
CssClass	
Display	Static
Font	
ForeColor	▇ Red
Text	
Behavior	
ControlToValidate	sname
EnableClientScript	True
Enabled	True
EnableTheming	True
EnableViewState	True
InitialValue	
SetFocusOnError	False
SkinID	
ToolTip	
ValidationGroup	
Visible	True
Data	
Layout	
Misc	
ID	Reqsname

Figure 13-20. *Configure the RequiredFieldValidator properties for the sname field.*

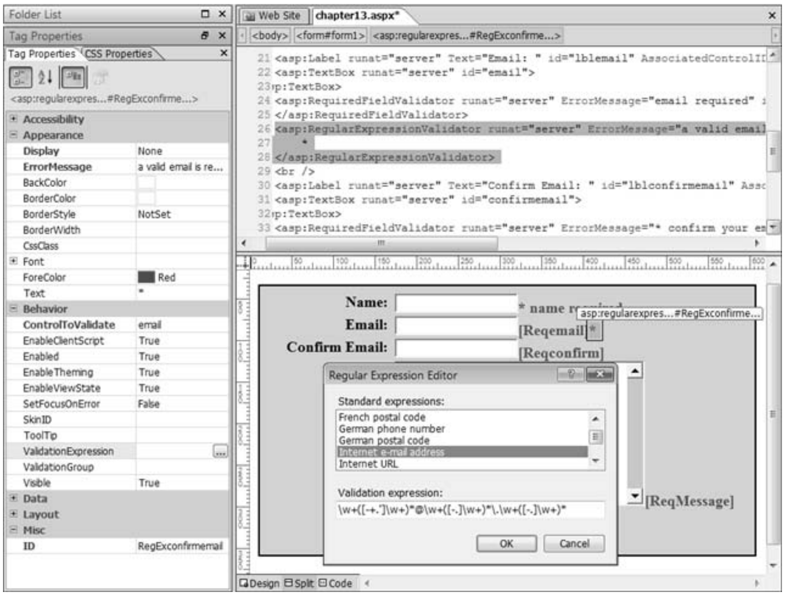
2. The properties we need to change for the Name form field are as follows:

   - `ErrorMessage: Name Required`: Set the text to display when the control displays.
   - `Display: Dynamic`: By changing the display property from static to dynamic, no space will be taken up by the error message unless there is an error.
   - `ControlToValidate`: Use the drop-down list to select the sname form field.
   - `ID: RequiredSname`: This means you will be able to tell which field the RequiredFieldValidator is validating.

3. Repeat step 2 for the Email, Confirm Email, and Message form fields, except that you should set `Display` to None on the Message form field. When the `Display` property is set to None, a `ValidationSummary` control must be used, which we will add later in this exercise. Make sure that you select the appropriate form field from the `ControlToValidate` drop-down for every validation control you use.

4. Next, drag out the `RegularExpression` validator, and drop it to the right of where the Email Required text shows in Design view. Type the `ErrorMessage` text, set `Display` to None, and set `ID` to RegExemail. When we use a `ValidationSummary` control and place an the asterisk in the `Text` property, an asterisk will display next to the e-mail form field, and the full message will appear in the summary message box. Next, click the three dots in `ValidationExpression` property to launch the Regular Expression Editor, and choose Internet e-mail address, as shown in Figure 13-21. Click OK.

5. Now, drag a `CompareValidator` control to the right of the `confirmemail` field. Type the `ErrorMessage`, set `Display` to None, place an asterisk in the `Text` property, and set `ID` to Compemail (see Figure 13-22).



```
<asp:Label runat="server" Text="Email: " id="lblemail" AssociatedControlII
<asp:TextBox runat="server" id="email">
p:TextBox>
<asp:RequiredFieldValidator runat="server" ErrorMessage="email required" i
</asp:RequiredFieldValidator>
<asp:RegularExpressionValidator runat="server" ErrorMessage="a valid email
    *
</asp:RegularExpressionValidator>
<br />
<asp:Label runat="server" Text="Confirm Email: " id="lblconfirmemail" Assc
<asp:TextBox runat="server" id="confirmemail">
p:TextBox>
<asp:RequiredFieldValidator runat="server" ErrorMessage="* confirm your em
```

Tag Properties: <asp:regularexpres...#RegExconfirme...>

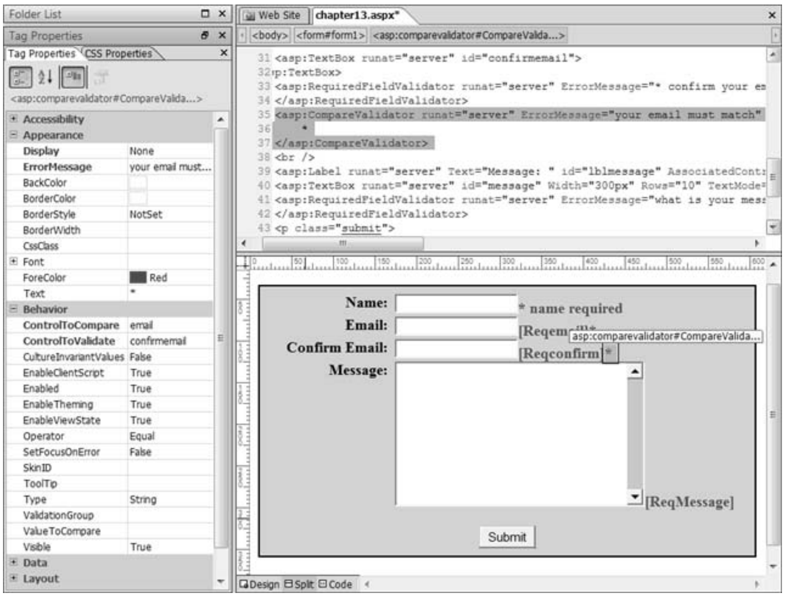
Display	None
ErrorMessage	a valid email is re...
BackColor	
BorderColor	
BorderStyle	NotSet
BorderWidth	
CssClass	
Font	
ForeColor	■ Red
Text	*
ControlToValidate	email
EnableClientScript	True
Enabled	True
EnableTheming	True
EnableViewState	True
SetFocusOnError	False
SkinID	
ToolTip	
ValidationExpression	
ValidationGroup	
Visible	True
ID	RegExconfirmemail

Regular Expression Editor

Standard expressions: French postal code, German phone number, German postal code, Internet e-mail address, Internet URL

Validation expression: `\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*`

Figure 13-21. *Expression Web includes the regular expression code for e-mail address validation.*



```
<asp:TextBox runat="server" id="confirmemail">
p:TextBox>
<asp:RequiredFieldValidator runat="server" ErrorMessage="* confirm your em
</asp:RequiredFieldValidator>
<asp:CompareValidator runat="server" ErrorMessage="your email must match"
    *
</asp:CompareValidator>
<br />
<asp:Label runat="server" Text="Message: " id="lblmessage" AssociatedCont:
<asp:TextBox runat="server" id="message" Width="300px" Rows="10" TextMode=
<asp:RequiredFieldValidator runat="server" ErrorMessage="what is your mess
</asp:RequiredFieldValidator>
<p class="submit">
```

Tag Properties: <asp:comparevalidator#CompareValida...>

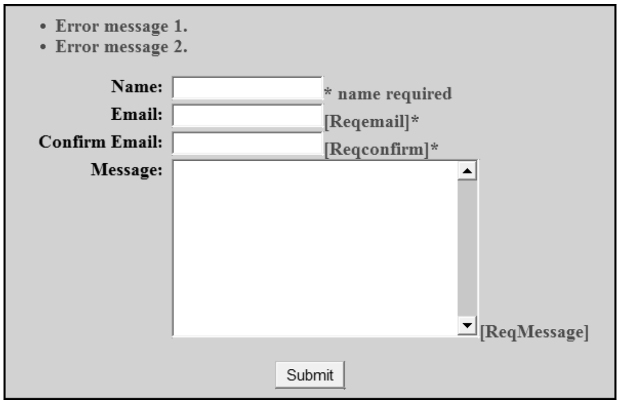
Display	None
ErrorMessage	your email must...
BackColor	
BorderColor	
BorderStyle	NotSet
BorderWidth	
CssClass	
Font	
ForeColor	■ Red
Text	*
ControlToCompare	email
ControlToValidate	confirmemail
CultureInvariantValues	False
EnableClientScript	True
Enabled	True
EnableTheming	True
EnableViewState	True
Operator	Equal
SetFocusOnError	False
SkinID	
ToolTip	
Type	String
ValidationGroup	
ValueToCompare	
Visible	True

Figure 13-22. *Compare e-mail addresses entered in the form.*

6. The last validation control we need to add is the `ValidationSummary` control. Drag it to the top of your form, well above the Name label, before you let go; otherwise, the control tends to land between the Name label and the `sname` textbox control.

7. While there is no need to change any of the default properties of the `ValidationSummary` control, you may prefer not to have your error list display bullets if you choose to use an asterisk as the `Text` property for the e-mail regular expression validation.

8. Save and view your form, which should look like Figure 13-23 in Design view.

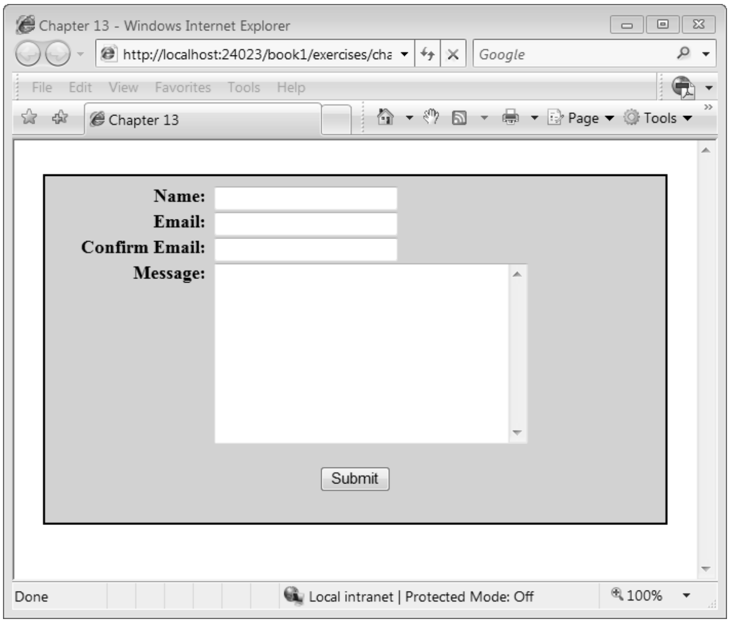


- Error message 1.
- Error message 2.

Name:		* name required
Email:		[Reqemail]*
Confirm Email:		[Reqconfirm]*
Message:		[ReqMessage]

Figure 13-23. *In Design view, you will see the error message or text properties unless Display is set to None.*

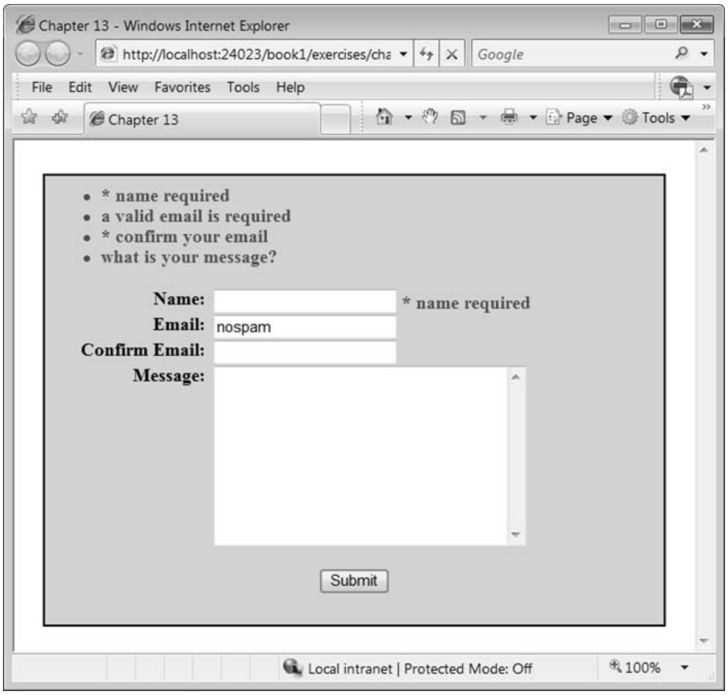
9. When viewed in the browser, the form will display the same as it did before ASP.NET validation was added, as shown in Figure 13-24.



Chapter 13 - Windows Internet Explorer
http://localhost:24023/book1/exercises/cha
Google
File Edit View Favorites Tools Help
Chapter 13
Page Tools
Name:
Email:
Confirm Email:
Message:
Submit
Done
Local intranet | Protected Mode: Off
100%

Figure 13-24. *Error messages appear only after the form is submitted without text or with a malformed e-mail address.*

10. See what happens when you skip form fields or enter an invalid e-mail address, as shown in Figure 13-22. Try changing the `Display` properties on the different validation controls to Static and see what difference that makes.


Chapter 13 - Windows Internet Explorer
http://localhost:24023/book1/exercises/cha
Google
File Edit View Favorites Tools Help
Chapter 13
Page
Tools
Local intranet | Protected Mode: Off
100%

- * name required
- a valid email is required
- * confirm your email
- what is your message?

Name:		* name required
Email:	nospam	
Confirm Email:		
Message:		

Submit

Figure 13-25. *Errors appear inline and in the summary area when Display is set to Dynamic.*

HTML Forms Validation

In the previous section, we used the ASP.NET validation controls. You must have ASP.NET support on your web server to use them on your website. So what do you do if you do not have ASP.NET support on your web server?

The options are to

- Use JavaScript for client-side validation before the form is submitted.
- Use classic ASP, PHP, or some other server-side technology supported by your web server that will check the data before continuing to process the form.
- Use a combination of JavaScript on the client side plus server-side scripting. This is how the ASP.NET validation controls work.

In this book, we will limit the non-ASP.NET forms validation to required field validation using client-side JavaScript. In Chapter 10, we created a form named contact; this is the code we will validate:

```
<form method="post" action="sendmail.asp" name="contact">
  <p>
    <label id="lblsname" for="sname">Name:</label>
    <input name="sname" type="text" id="sname" /> <br />
    <label id="lblemail" for="email">Email: </label>
    <input name="email" type="text" id="email" /><br />
    <label id="lblmessage" for="message" id="message">Message:</label>
    <textarea name="message" cols="30" rows="10"></textarea>
  </p>
  <p class="submit">
    <input name="submit1" id="submit1" type="submit" value="submit" />
  </p>
</form>
```

In the form, we have three fields with the names of `sname`, `email`, and `message`. Simple required field validation can be performed using the following JavaScript code that checks for the presence of any content in each of the form fields specified in the `if` statements:

```
<script language="javascript"  type="text/javascript">
function validateForm() {
  with (document.contact) {
    var alertMsg = "The following REQUIRED fields\nhave been left empty:\n";
    if (sname.value == "") alertMsg += "\nName";
    if (email.value == "") alertMsg += "\nEmail";
    if (message.value == "") alertMsg += "\nMessage";
    if (alertMsg != "The following REQUIRED fields\nhave been left empty:\n") {
      alert(alertMsg);
      return false;
    } else {
      return true;
    }
  }
}
</script>
```

For forms with different form field names, in the following code, replace `fieldname` and `Error-Message` with the name of the form field you wish to validate and the message you want displayed in the alert box:

```
if (fieldname.value == "") alertMsg += "\nError-Message";
```

Add or remove the `if` line in the preceding code as necessary for the fields in your form. In order to run this JavaScript script, we must change the `<form>` element and replace

```
<form method="post" action="sendmail.asp" name="contact">
```

with

```
<form method="post" action="sendmail.asp" name="contact"
      onsubmit="return validateForm()">
```

Since these alerts appear in a separate alert box on top of your visitor's browser window, I recommend adding information to your form to tell the visitor which form fields are required. That way, your visitor is more likely to fill out the required fields before submitting the form.

While an asterisk is frequently used to indicate a required field, it is a better practice to spell out "required" next to the form to make it clear.

Password Protecting a Folder

The ability to password protect a section of a website is something many web designers wish to do for one or more folders in their websites. In some cases, you can accomplish this by using the control panel provided by your web host, but if you have ASP.NET 2.0 on your web server, you can control access to folders in your site using the `web.config` file and a login form. Whenever your site visitor tries to access an `.aspx` page in the folder, they will be directed to a `login.aspx` page in the main section of your website. If they provide the correct user name and password, the page they requested will be displayed. Only `.aspx` pages can be protected in this manner; HTML pages can still be viewed.

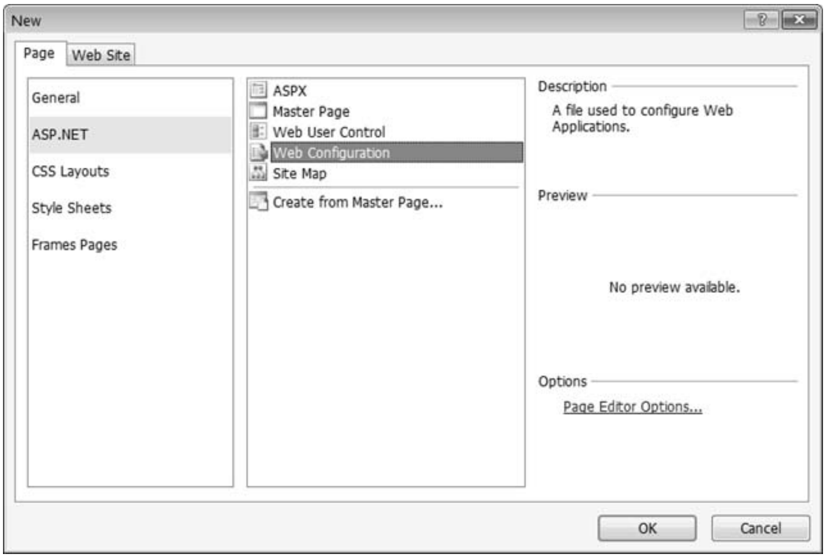
The `web.config` file is a special XML file that holds information about your website. Depending on your hosting provider, you may already have a `web.config` file in your website from when your web hosting was set up. If you have an existing `web.config` file, make a backup copy before you proceed. If you do not have an existing `web.config` file, you will need to create one.

Unlike all of the other code you have been able to download from `http://foundationsofexpressionweb.com`, the code for this last section of this chapter is not in the exercise section but in the extras folder. To download the code to password protect a folder on your site, you must go to `http://foundationsofexpressionweb.com/extras` and log in with the following credentials:

- *User ID*: book
- *Password*: chapter13

Creating a web.config File

If you do not have a `web.config` file, create one by selecting File ➤ New ➤ Page, and from the ASP.NET section, choosing Web Configuration, as shown in Figure 13-26. Save the file created as `web.config`.



A file used to configure Web Applications.

Figure 13-26. *Web Configuration creates a web.config file.*

Expression Web creates a basic `web.config` file with basic settings and explanatory text surrounded by comments so that your web server does not attempt to execute the code. Without the extra information in the comments, the `web.config` file created is as follows:

```
<configuration xmlns="http://schemas.microsoft.com/.NetConfiguration/v2.0">
  <appSettings/>
  <connectionStrings/>
  <system.web>
    <compilation debug="false"/>
    <authentication mode="Windows"/>
    <!--
    <customErrors mode="RemoteOnly" defaultRedirect="GenericErrorPage.htm">
      <error statusCode="403" redirect="NoAccess.htm"/>
      <error statusCode="404" redirect="FileNotFound.htm"/>
    </customErrors>
    -->
  </system.web>
</configuration>
```

■ **Tip** The `<customErrors>` section is used if you want to design your own custom page to display when the visitor tries to access a page that does not exist. Simply replace the `FileNotFound.htm` file with the name of the file you create.

To create a protected area, we must replace the default code in the `<authentication mode="Windows"/>` section and add a `<location>` section so that your `<system.web>` section is as follows:

```
<configuration xmlns="http://schemas.microsoft.com/.NetConfiguration/v2.0">
  <appSettings/>
  <connectionStrings/>
  <system.web>
    <authentication mode="Forms">
      <forms loginUrl="login.aspx" timeout="30" >
        <credentials passwordFormat="Clear">
          <user name="book" password="chapter13" />
        </credentials>
      </forms>
    </authentication>
  </system.web>
  <connectionStrings/>
  <location path="extras">
    <system.web>
      <authorization>
        <deny users="?" />
      </authorization>
    </system.web>
  </location>
</configuration>
```

This code changes the method of authentication from Windows operating system to forms based and sets a login expiration time of 30 minutes after the last new password-protected page is accessed.

`<credentials>` says how the password is sent. Here, it is sent as clear text, but if you are protecting really sensitive information, you should use one of the encoded password methods such as SHA1 or MD5.

The user name and password specify the login name and password required. You can have more than one user name and password by repeating the `<user name="book" password="chapter13" />` line with different values for name and password immediately below the current location, for example:

```
<user name="book" password="chapter13" />
<user name="foundations" password="expression" />
<user name="admin" password="%fi?257" />
```

The `<location>` section specifies which folder in your website is protected. In this case, it is the extras folder off the root. If you have not done so already, try it out at `http://foundationsofexpressionweb.com/extras`.

This is where you will find bonus material for people who own this book. When you attempt to access this section you will be redirected to the `login.aspx` page. Use the following credentials discussed earlier to confirm it is working:

- *User ID*: book
- *Password*: chapter13

■ **Note** The user name and password are case sensitive, and no spaces are allowed. Make sure you type them correctly as **book** and **chapter13**.

If your website has a `web.config` file provided by your hosting provider, check with your provider before you edit the file to make sure that you do not break any functions or database connections on your site.

The completed `web.config` file should now look like this:

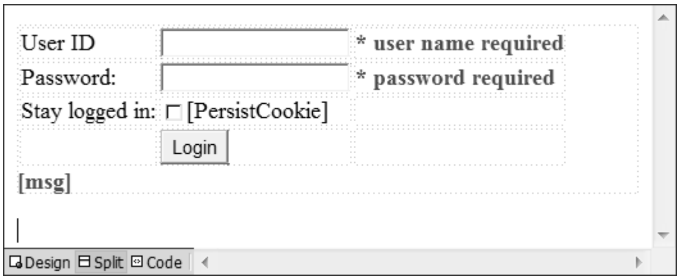
```
<?xml version="1.0"?>
<configuration xmlns="http://schemas.microsoft.com/.NetConfiguration/v2.0">
  <appSettings/>
  <connectionStrings/>
  <system.web>
    <authentication mode="Forms">
      <forms loginUrl="login.aspx" timeout="30" >
        <credentials passwordFormat="Clear">
          <user name="book" password="chapter13" />
        </credentials>
      </forms>
    </authentication>
  </system.web>
  <location path="extras">
    <system.web>
      <authorization>
        <deny users="?" />
      </authorization>
    </system.web>
  </location>
  <!--
  Note you can have more than one system.web section in your
  web.config file as long as they do not conflict.
  -->

  <system.web>
```

```
    <!--
    Set compilation debug="true" to insert debugging
    symbols into the compiled page. Because this
    affects performance, set this value to true only
    during development.
    -->
    <compilation debug="false"/>

    <!--
    The <customErrors> section enables configuration
    of what to do if/when an unhandled error occurs
    during the execution of a request. Specifically,
    it enables developers to configure html error pages
    to be displayed in place of a error stack trace.

    <customErrors mode="RemoteOnly" defaultRedirect="GenericErrorPage.htm">
      <error statusCode="403" redirect="NoAccess.htm"/>
      <error statusCode="404" redirect="FileNotFound.htm"/>
    </customErrors>
    -->
  </system.web>
</configuration>
```

Now that we have created a secured area, next we will need to create a login form to allow those with the specified credentials to access the protected folder.

Exercise 13-2. Creating a Login Form

In this exercise, we'll create a login form that looks like the one shown in Figure 13-27. Unlike the contact form we created in Chapter 10, the textboxes in this login form must have the same names as shown in the code at the end of this exercise. The only optional form field is the checkbox that sets a cookie that keeps the visitor logged in.



User ID		* user name required
Password:		* password required
Stay logged in:	☐ [PersistCookie]	
	Login	

[msg]

Figure 13-27. *Login form with validation*

Follow these steps to create the login form:

1. Start by creating a four-row, three-column table.
2. Drag three ASP.NET label controls into the first column.
3. Drag two ASP.NET textbox controls to the second column with an ASP.NET checkbox in the third row of the second column.
4. The last row contains only an ASP.NET button.
5. The form is finished with an ASP.NET label below the table to hold an error message if the credentials supplied are incorrect.
6. Make a note of the control names in the previous code, and use the Tag Properties task pane to set your control properties to match.
7. With two required field validation controls added for the two textboxes, the complete code for our login form is as follows:

```
<%@ Page Language="C#" %>
<%@ Import Namespace="System.Web.Security " %>
<!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>Log In</title>
  <style type="text/css">
  .error {
    color: red;
    font-weight: bold;
  }
  </style>
</head>
<body>
  <form id="form1" runat="server">
    <table>
      <tr>
        <td>
          <asp:Label runat="server" Text="User ID" id="lblname"
          AssociatedControlID="loginName"></asp:Label>
        </td>
        <td>
          <asp:TextBox runat="server" id="loginName"></asp:TextBox>
        </td>
        <td>
```

```
          <asp:RequiredFieldValidator runat="server"
          ErrorMessage="* user name required" id="requser" CssClass="error"
          ControlToValidate="loginName">
          </asp:RequiredFieldValidator>
        </td>
      </tr>
      <tr>
        <td>
          <asp:Label runat="server" Text="Password:" id="lblpassword"
          AssociatedControlID="Password"></asp:Label>
        </td>
        <td>
          <asp:TextBox runat="server" id="Password"></asp:TextBox>
        </td>
        <td>
          <asp:RequiredFieldValidator runat="server"
          ErrorMessage="* password required" id="reqpasswrod"
          ControlToValidate="Password" CssClass="error">
          </asp:RequiredFieldValidator>
        </td>
      </tr>
      <tr>
        <td>
          <asp:Label runat="server" Text="Stay logged in:" id="Label3">
          </asp:Label>
        </td>
        <td>
          <asp:CheckBox runat="server" id="PersistCookie" />
        </td>
        <td>&nbsp;</td>
      </tr>
      <tr>
        <td>&nbsp;</td>
        <td>
          <asp:Button runat="server" text="Login" id="login" onclick="Login" />
        </td>
        <td>&nbsp;</td>
      </tr>
    </table>
    <asp:Label runat="server" id="msg" CssClass="error"></asp:Label>
  </form>
</body>
</html>
```

When viewed in a browser, we see the simple login form shown in Figure 13-28.

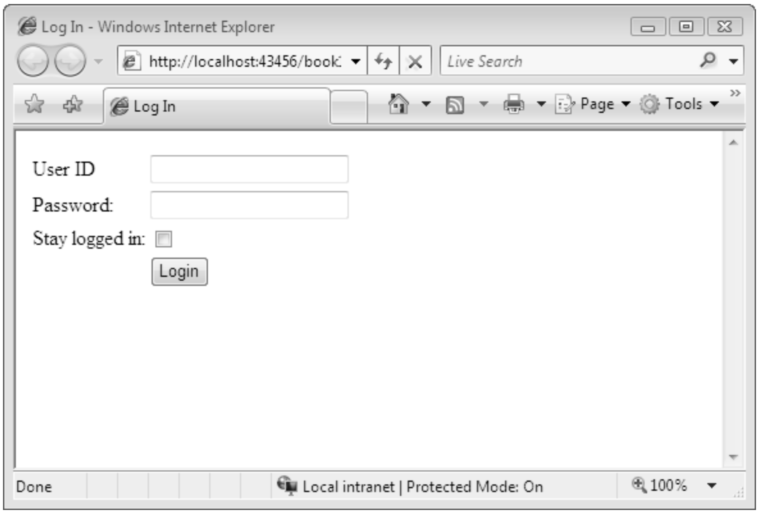

Log In - Windows Internet Explorer
http://localhost:43456/book
Live Search
Log In
Page
Tools
User ID
Password:
Stay logged in:
Login
Done
Local intranet | Protected Mode: On
100%

Figure 13-28. *A login form that can be styled how you choose.*

There remains one small piece of ASP.NET code to add in the `<head>` section of the page; it checks the login name and password against the ones you specified in the `web.config` file:

```
<script runat="server">
public void Login (Object src, EventArgs e)
{
  if (FormsAuthentication.Authenticate (loginName.Text, Password.Text))
    FormsAuthentication.RedirectFromLoginPage(loginName.Text,
                                                PersistCookie.Checked);
  else
    msg.Text ="invalid login: please try again.";
}
</script>
```

If they match, you will be transferred to the page you tried to access in the protected folder. If not, `msg.Text` will be displayed on the login form.

Your folder is now password protected, and nothing else is required, since the user's login will time out after 30 minutes or the time you set in the `<forms loginUrl="login.aspx" timeout="30" >` line in your `web.config` file.

Summary

In this chapter, we have moved beyond the basics of creating a website. The concept of media-specific stylesheets was introduced. The ways that print stylesheets can allow you to decide how your web pages will print when your visitor chooses to print was demonstrated.

Next, we moved on to validating the fields in our web forms to help you get the information necessary to respond to inquiries from your site visitors. We used ASP.NET 2.0 validation controls, which validate first on the client and again on the server. For situations when ASP.NET 2.0 is not available to us, we used JavaScript for client-side validation of HTML forms.

Our last exercise showed how to password protect a section of our website so that only those who know the User ID and password can see the `.aspx` pages in the folder.