CHAPTER 3

Using the Validation Controls

IN THIS CHAPTER

In this chapter, you learn how to validate form fields when a form is submitted to the web server. You can use the validation controls to prevent users from submitting the wrong type of data into a database table. For example, you can use validation controls to prevent a user from submitting the value "Apple" for a birth date field.

In the first part of this chapter, you are provided with an overview of the standard validation controls included in the ASP.NET 2.0 Framework. You learn how to control how validation errors are displayed, how to highlight validation error messages, and how to use validation groups. You are provided with sample code for using each of the standard validation controls.

Next, we extend the basic validation controls with our own custom validation controls. For example, you learn how to create an `AjaxValidator` control that enables you to call a server-side validation function from the client.

Overview of the Validation Controls

Six validation controls are included in the ASP.NET 2.0 Framework:

- `RequiredFieldValidator`—Enables you to require a user to enter a value in a form field.
- `RangeValidator`—Enables you to check whether a value falls between a certain minimum and maximum value.
- `CompareValidator`—Enables you to compare a value against another value or perform a data type check.

- `RegularExpressionValidator`—Enables you to compare a value against a regular expression.
- `CustomValidator`—Enables you to perform custom validation.
- `ValidationSummary`—Enables you to display a summary of all validation errors in a page.

You can associate the validation controls with any of the form controls included in the ASP.NET Framework. For example, if you want to require a user to enter a value into a `TextBox` control, then you can associate a `RequiredFieldValidator` control with the `TextBox` control.

NOTE

Technically, you can use the validation controls with any control that is decorated with the `ValidationProperty` attribute.

The page in Listing 3.1 contains a simple order entry form. It contains three `TextBox` controls that enable you to enter a product name, product price, and product quantity. Each of the form fields are validated with the validation controls.

LISTING 3.1 `OrderForm.aspx`

```
<%@ Page Language="VB" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<script runat="server">

    Protected Sub btnSubmit_Click(ByVal sender As Object, ByVal e
➥As System.EventArgs)
        If Page.IsValid Then
            lblResult.Text = "<br />Product: " & txtProductName.Text
            lblResult.Text &= "<br />Price: " & txtProductPrice.Text
            lblResult.Text &= "<br />Quantity: " & txtProductQuantity.Text
        End If
    End Sub
</script>
<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
    <title>Order Form</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>

    <fieldset>
```

LISTING 3.1 Continued

```
    <legend>Product Order Form</legend>

    <asp:Label
        id="lblProductName"
        Text="Product Name:"
        AssociatedControlID="txtProductName"
        Runat="server" />
    <br />
    <asp:TextBox
        id="txtProductName"
        Runat="server" />
    <asp:RequiredFieldValidator
        id="reqProductName"
        ControlToValidate="txtProductName"
        Text="(Required)"
        Runat="server" />

    <br /><br />

    <asp:Label
        id="lblProductPrice"
        Text="Product Price:"
        AssociatedControlID="txtProductPrice"
        Runat="server" />
    <br />
    <asp:TextBox
        id="txtProductPrice"
        Columns="5"
        Runat="server" />
    <asp:RequiredFieldValidator
        id="reqProductPrice"
        ControlToValidate="txtProductPrice"
        Text="(Required)"
        Display="Dynamic"
        Runat="server" />
    <asp:CompareValidator
        id="cmpProductPrice"
        ControlToValidate="txtProductPrice"
        Text="(Invalid Price)"
        Operator="DataTypeCheck"
        Type="Currency"
        Runat="server" />

    <br /><br />
```

LISTING 3.1 Continued

```
    <asp:Label
        id="lblProductQuantity"
        Text="Product Quantity:"
        AssociatedControlID="txtProductQuantity"
        Runat="server" />
    <br />
    <asp:TextBox
        id="txtProductQuantity"
        Columns="5"
        Runat="server" />
    <asp:RequiredFieldValidator
        id="reqProductQuantity"
        ControlToValidate="txtProductQuantity"
        Text="(Required)"
        Display="Dynamic"
        Runat="server" />
    <asp:CompareValidator
        id="CompareValidator1"
        ControlToValidate="txtProductQuantity"
        Text="(Invalid Quantity)"
        Operator="DataTypeCheck"
        Type="Integer"
        Runat="server" />

    <br /><br />

    <asp:Button
        id="btnSubmit"
        Text="Submit Product Order"
        OnClick="btnSubmit_Click"
        Runat="server" />

    </fieldset>

    <asp:Label
        id="lblResult"
        Runat="server" />

    </div>
    </form>
</body>
</html>
```

A separate RequiredFieldValidator control is associated with each of the three form fields. If you attempt to submit the form in Listing 3.1 without entering a value for a field, then a validation error message is displayed (see Figure 3.1).

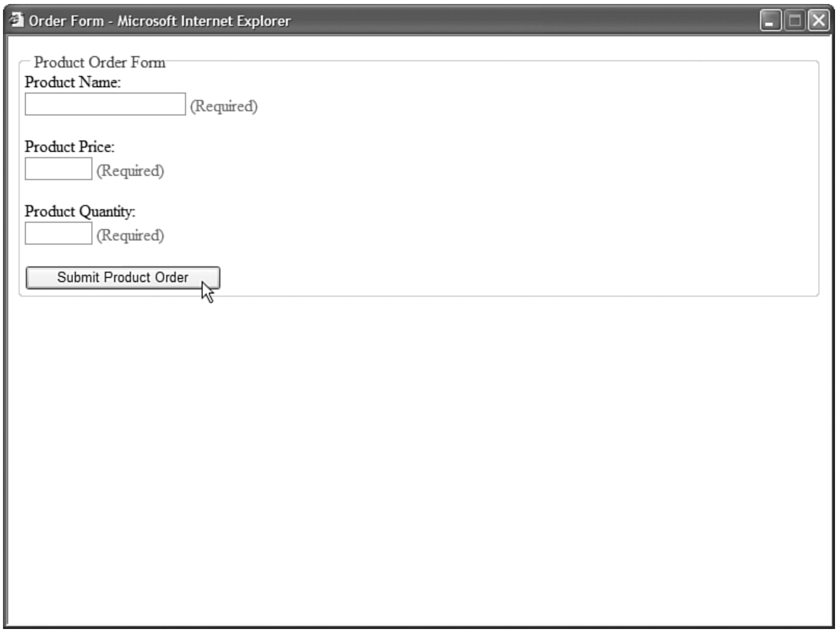


FIGURE 3.1 Displaying a validation error message.

Each RequiredFieldValidator is associated with a particular control through its ControlToValidate property. This property accepts the name of the control to validate on the page.

CompareValidator controls are associated with the txtProductPrice and txtProductQuantity TextBox controls. The first CompareValidator is used to check whether the txtProductPrice text field contains a currency value, and the second CompareValidator is used to check whether the txtProductQuantity text field contains an integer value.

Notice that there is nothing wrong with associating more than one validation control with a form field. If you need to make a form field required and check the data type entered into the form field, then you need to associate both a RequiredFieldValidator and CompareValidator control with the form field.

Finally, notice that the Page.IsValid property is checked in the btnSubmit_Click() handler after the form data is submitted. When using the validation controls, you should always check the Page.IsValid property before doing anything with the data submitted to a page. This property returns the value true when, and only when, there are no validation errors on the page.

Validation Controls and JavaScript

By default, the validation controls perform validation on both the client (the browser) and the server. The validation controls use client-side JavaScript. This is great from a user experience perspective because you get immediate feedback whenever you enter an invalid value into a form field.

NOTE

The `RequiredFieldValidator` will not perform client-side validation until after you attempt to submit a form at least once or you enter and remove data in a form field.

Client-side JavaScript is supported on any uplevel browser. Supported browsers include Internet Explorer, Firefox, and Opera. This is a change from the previous version of ASP.NET, which supported only Internet Explorer as an uplevel browser.

You can use the validation controls with browsers that do not support JavaScript (or do not have JavaScript enabled). If a browser does not support JavaScript, the form must be posted back to the server before a validation error message is displayed.

Even when validation happens on the client, validation is still performed on the server. This is done for security reasons. If someone creates a fake form and submits the form data to your web server, the person still won't be able to submit invalid data.

If you prefer, you can disable client-side validation for any of the validation controls by assigning the value `False` to the validation control's `EnableClientScript` property.

Using `Page.IsValid`

As mentioned earlier, you should always check the `Page.IsValid` property when working with data submitted with a form that contains validation controls. Each of the validation controls includes an `IsValid` property that returns the value `True` when there is not a validation error. The `Page.IsValid` property returns the value `True` when the `IsValid` property for all of the validation controls in a page returns the value `True`.

It is easy to forget to check the `Page.IsValid` property. When you use an uplevel browser that supports JavaScript with the validation controls, you are prevented from submitting a form back to the server when there are validation errors. However, if someone requests a page using a browser that does not support JavaScript, the page is submitted back to the server even when there are validation errors.

For example, if you request the page in Listing 3.1 with a browser that does not support JavaScript and submit the form without entering form data, then the `btnSubmit_Click()` handler executes on the server. The `Page.IsValid` property is used in Listing 3.1 to prevent downlevel browsers from displaying invalid form data.

WARNING

Unfortunately, I've made the mistake of forgetting to include a check of the `Page.IsValid` property several times when building applications. Because you do not normally develop a web application with a downlevel browser, you won't notice the problem described in this section until you start getting invalid data in your database tables.

Setting the Display Property

All the validation controls include a `Display` property that determines how the validation error message is rendered. This property accepts any of the following three possible values:

- Static
- Dynamic
- None

By default, the `Display` property has the value `Static`. When the `Display` property has this value, the validation error message rendered by the validation control looks like this:

```
<span id="reqProductName" style="color:Red;visibility:hidden;">(Required)</span>
```

Notice that the error message is rendered in a `<span>` tag that includes a Cascading Style Sheet style attribute that sets the visibility of the `<span>` tag to `hidden`.

If, on the other hand, you set the `Display` property to the value `Dynamic`, the error message is rendered like this:

```
<span id="reqProductName" style="color:Red;display:none;">(Required)</span>
```

In this case, a Cascading Style Sheet `display` attribute hides the contents of the `<span>` tag.

Both the visibility and display attributes can be used to hide text in a browser. However, text hidden with the `visibility` attribute still occupies screen real estate. Text hidden with the `display` attribute, on the other hand, does not occupy screen real estate.

In general, you should set a validation control's `Display` property to the value `Dynamic`. That way, if other content is displayed next to the validation control, the content is not pushed to the right. All modern browsers (Internet Explorer, Firefox, and Opera) support the Cascading Style Sheet `display` attribute.

The third possible value of the `Display` property is `None`. If you prefer, you can prevent the individual validation controls from displaying an error message and display the error messages with a `ValidationSummary` control. You learn how to use the `ValidationSummary` control later in this chapter.

Highlighting Validation Errors

When a validation control displays a validation error, the control displays the value of its `Text` property. Normally, you assign a simple text string, such as `"(Required)"` to the `Text` property. However, the `Text` property accepts any HTML string.

For example, the page in Listing 3.2 displays an image when you submit the form without entering a value for the First Name text field (see Figure 3.2).

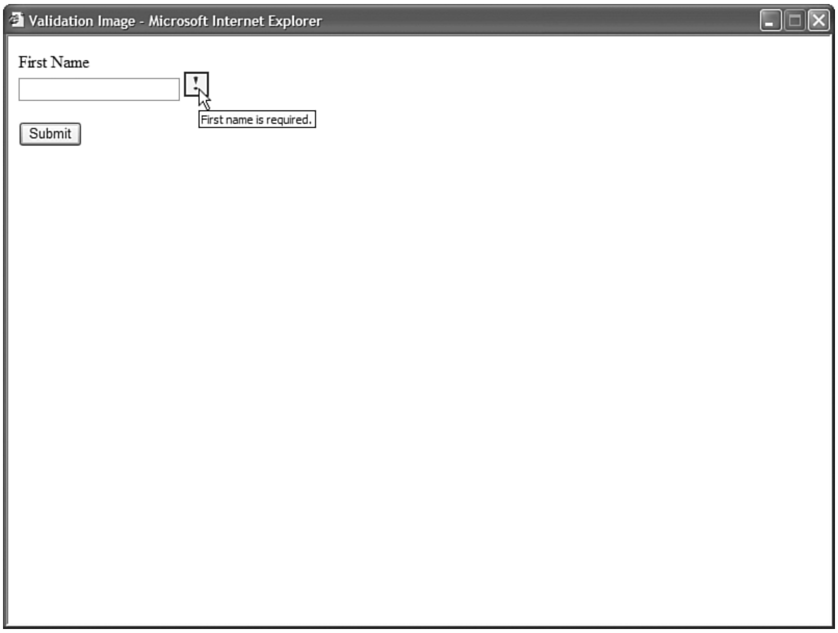


FIGURE 3.2 Displaying an image for a validation error.

LISTING 3.2 `ValidationImage.aspx`

```
<%@ Page Language="VB" %>
<!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 id="Head1" runat="server">
    <title>Validation Image</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>

    <asp:Label
        id="lblFirstName"
```

LISTING 3.2 Continued

```
        Text="First Name"
        AssociatedControlID="txtFirstName"
        Runat="server" />
    <br />
    <asp:TextBox
        id="txtFirstName"
        Runat="server" />
    <asp:RequiredFieldValidator
        id="reqFirstName"
        ControlToValidate="txtFirstName"
        Text="<img src='Error.gif' alt='First name is required.' />"
        Runat="server" />

     <br /><br />

     <asp:Button
        id="btnSubmit"
        Text="Submit"
        Runat="server" />

    </div>
    </form>
</body>
</html>
```

In Listing 3.2, the `Text` property contains an HTML `<img>` tag. When there is a validation error, the image represented by the `<img>` tag is displayed.

Another way that you can emphasize errors is to take advantage of the `SetFocusOnError` property that is supported by all the validation controls. When this property has the value `True`, the form focus is automatically shifted to the control associated with the validation control when there is a validation error.

For example, the page in Listing 3.3 contains two `TextBox` controls that are both validated with `RequiredFieldValidator` controls. Both `RequiredFieldValidator` controls have their `SetFocusOnError` properties enabled. If you provide a value for the first text field and not the second text field and submit the form, the form focus automatically shifts to the second form field.

LISTING 3.3 `ShowSetFocusOnError.aspx`

```
<%@ Page Language="VB" %>
<!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" >
```

LISTING 3.3 Continued

```
<head id="Head1" runat="server">
    <title>Show SetFocusOnError</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>

    <asp:Label
        id="lblFirstName"
        Text="First Name"
        AssociatedControlID="txtFirstName"
        Runat="server" />
    <br />
    <asp:TextBox
        id="txtFirstName"
        Runat="server" />
    <asp:RequiredFieldValidator
        id="reqFirstName"
        ControlToValidate="txtFirstName"
        Text="(Required)"
        SetFocusOnError="true"
        Runat="server" />

    <br /><br />

    <asp:Label
        id="lblLastName"
        Text="Last Name"
        AssociatedControlID="txtLastName"
        Runat="server" />
    <br />
    <asp:TextBox
        id="txtLastname"
        Runat="server" />
    <asp:RequiredFieldValidator
        id="reqLastName"
        ControlToValidate="txtLastName"
        Text="(Required)"
        SetFocusOnError="true"
        Runat="server" />

     <br /><br />

     <asp:Button
```

LISTING 3.3 Continued

```
        id="btnSubmit"
        Text="Submit"
        Runat="server" />

    </div>
    </form>
</body>
</html>
```

Finally, if you want to really emphasize the controls associated with a validation error, then you can take advantage of the `Page.Validators` property. This property exposes the collection of all the validation controls in a page. In Listing 3.4, the `Page.Validators` property is used to highlight each control that has a validation error (see Figure 3.3).

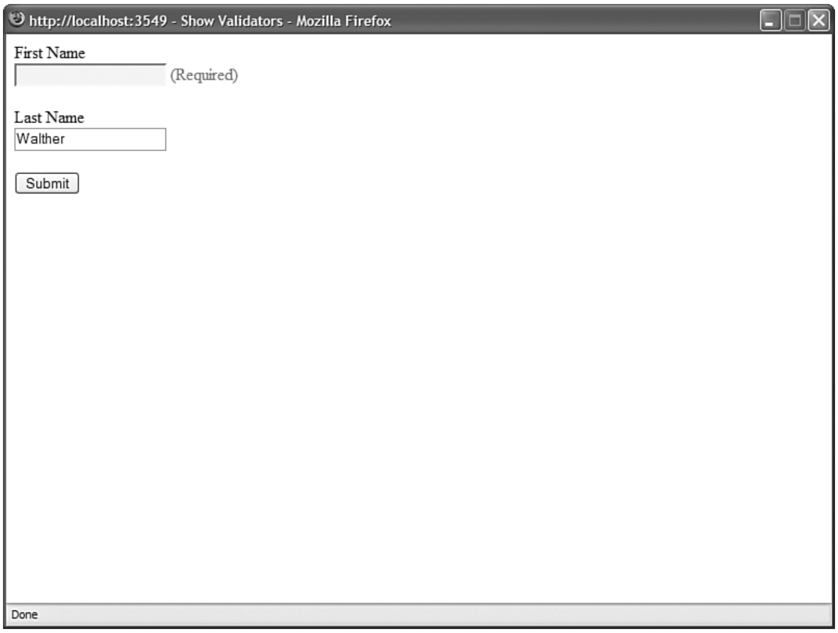


FIGURE 3.3 Changing the background color of form fields.

LISTING 3.4 `ShowValidators.aspx`

```
<%@ Page Language="VB" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<script runat="server">
```

LISTING 3.4 Continued

```
    Sub Page_PreRender()
        For Each valControl As BaseValidator In Page.Validators
            Dim assControl As WebControl =
➥Page.FindControl(valControl.ControlToValidate)
            If Not valControl.IsValid Then
                assControl.BackColor = Drawing.Color.Yellow
            Else
                assControl.BackColor = Drawing.Color.White
            End If
        Next
    End Sub
</script>
<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
    <title>Show Validators</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>

    <asp:Label
        id="lblFirstName"
        Text="First Name"
        AssociatedControlID="txtFirstName"
        Runat="server" />
    <br />
    <asp:TextBox
        id="txtFirstName"
        Runat="server" />
    <asp:RequiredFieldValidator
        id="reqFirstName"
        ControlToValidate="txtFirstName"
        Text="(Required)"
        EnableClientScript="false"
        Runat="server" />

    <br /><br />

    <asp:Label
        id="lblLastName"
        Text="Last Name"
        AssociatedControlID="txtLastName"
        Runat="server" />
    <br />
```

LISTING 3.4 Continued

```
    <asp:TextBox
        id="txtLastname"
        Runat="server" />
    <asp:RequiredFieldValidator
        id="reqLastName"
        ControlToValidate="txtLastName"
        Text="(Required)"
        EnableClientScript="false"
        Runat="server" />

     <br /><br />

     <asp:Button
        id="btnSubmit"
        Text="Submit"
        Runat="server" />

    </div>
    </form>
</body>
</html>
```

The `Page.Validators` property is used in the `Page_PreRender()` handler. The `IsValid` property is checked for each control in the `Page.Validators` collection. If `IsValid` returns `False`, then the control being validated by the validation control is highlighted with a yellow background color.

Using Validation Groups

In the first version of the ASP.NET Framework, there was no easy way to add two forms to the same page. If you added more than one form to a page, and both forms contained validation controls, then the validation controls in both forms were evaluated regardless of which form you submitted.

For example, imagine that you wanted to create a page that contained both a login and registration form. The login form appeared in the left column and the registration form appeared in the right column. If both forms included validation controls, then submitting the login form caused any validation controls contained in the registration form to be evaluated.

In ASP.NET 2.0, you no longer face this limitation. The ASP.NET 2.0 Framework introduces the idea of validation groups. A validation group enables you to group related form fields together.

For example, the page in Listing 3.5 contains both a login and registration form and both forms contain independent sets of validation controls.

LISTING 3.5 ShowValidationGroups.aspx

```
<%@ Page Language="VB" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<script runat="server">

    Protected Sub btnLogin_Click(ByVal sender As Object, ByVal e As EventArgs)
        If Page.IsValid() Then
            lblLoginResult.Text = "Log in successful!"
        End If
    End Sub

    Protected Sub btnRegister_Click(ByVal sender As Object, ByVal e As EventArgs)
        If Page.IsValid() Then
            lblRegisterResult.Text = "Registration successful!"
        End If
    End Sub
</script>
<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
    <style type="text/css">
        html
        {
            background-color:silver;
        }
        .column
        {
            float:left;
            width:300px;
            margin-left:10px;
            background-color:white;
            border:solid 1px black;
            padding:10px;
        }
    </style>
    <title>Show Validation Groups</title>
</head>
<body>
    <form id="form1" runat="server">

    <div class="column">
    <fieldset>
```

LISTING 3.5 Continued

```
    <legend>Login</legend>
    <p>
    Please log in to our Website.
    </p>
    <asp:Label
        id="lblUserName"
        Text="User Name:"
        AssociatedControlID="txtUserName"
        Runat="server" />
    <br />
    <asp:TextBox
        id="txtUserName"
        Runat="server" />
    <asp:RequiredFieldValidator
        id="reqUserName"
        ControlToValidate="txtUserName"
        Text="(Required)"
        ValidationGroup="LoginGroup"
        Runat="server" />
    <br /><br />
    <asp:Label
        id="lblPassword"
        Text="Password:"
        AssociatedControlID="txtPassword"
        Runat="server" />
    <br />
    <asp:TextBox
        id="txtPassword"
        TextMode="Password"
        Runat="server" />
    <asp:RequiredFieldValidator
        id="reqPassword"
        ControlToValidate="txtPassword"
        Text="(Required)"
        ValidationGroup="LoginGroup"
        Runat="server" />
    <br /><br />
    <asp:Button
        id="btnLogin"
        Text="Login"
        ValidationGroup="LoginGroup"
        Runat="server" OnClick="btnLogin_Click" />
    </fieldset>
```

LISTING 3.5 Continued

```
    <asp:Label
        id="lblLoginResult"
        Runat="server" />

    </div>

    <div class="column">
    <fieldset>
    <legend>Register</legend>
    <p>
    If you do not have a User Name, please
    register at our Website.
    </p>
    <asp:Label
        id="lblFirstName"
        Text="First Name:"
        AssociatedControlID="txtFirstName"
        Runat="server" />
    <br />
    <asp:TextBox
        id="txtFirstName"
        Runat="server" />
    <asp:RequiredFieldValidator
        id="reqFirstName"
        ControlToValidate="txtFirstName"
        Text="(Required)"
        ValidationGroup="RegisterGroup"
        Runat="server" />
    <br /><br />
    <asp:Label
        id="lblLastName"
        Text="Last Name:"
        AssociatedControlID="txtLastName"
        Runat="server" />
    <br />
    <asp:TextBox
        id="txtLastName"
        Runat="server" />
    <asp:RequiredFieldValidator
        id="reqLastName"
        ControlToValidate="txtLastName"
        Text="(Required)"
        ValidationGroup="RegisterGroup"
        Runat="server" />
```

LISTING 3.5 Continued

```
    <br /><br />
    <asp:Button
        id="btnRegister"
        Text="Register"
        ValidationGroup="RegisterGroup"
        Runat="server" OnClick="btnRegister_Click" />
    </fieldset>

    <asp:Label
        id="lblRegisterResult"
        Runat="server" />

    </div>

    </form>
</body>
</html>
```

Notice that the validation controls and the button controls all include `ValidationGroup` properties. The controls associated with the login form all have the value `"LoginGroup"` assigned to their `ValidationGroup` properties. The controls associated with the register form all have the value `"RegisterGroup"` assigned to their `ValidationGroup` properties.

Because the form fields are grouped into different validation groups, you can submit the two forms independently. Submitting the Login form does not trigger the validation controls in the Register form (see Figure 3.4).

You can assign any string to the `ValidationGroup` property. The only purpose of the string is to associate different controls in a form together into different groups.

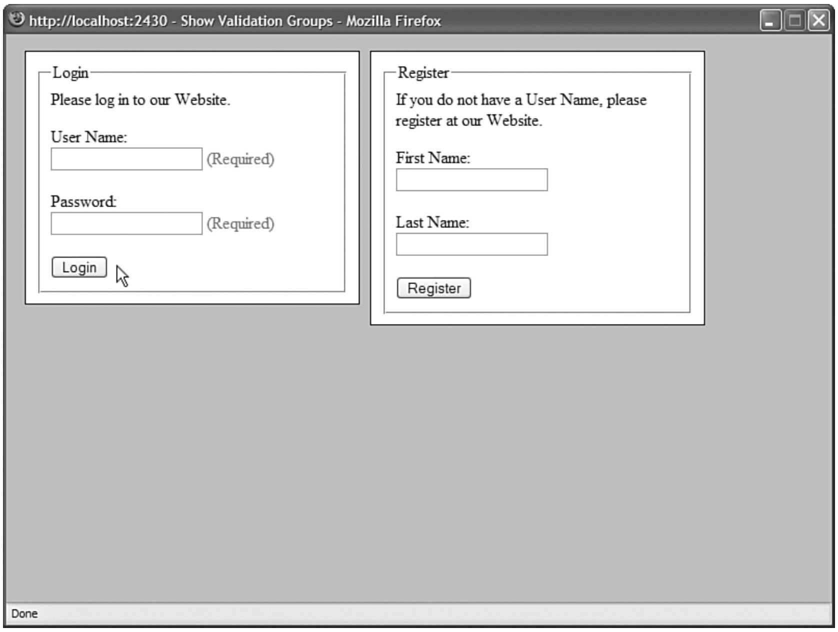
NOTE

Using validation groups is particularly important when working with Web Parts because multiple Web Parts with different forms might be added to the same page.

Disabling Validation

All the button controls—the `Button`, `LinkButton`, and `ImageButton` control—include a `CausesValidation` property. If you assign the value `False` to this property, then clicking the button bypasses any validation in the page.

Bypassing validation is useful when creating a Cancel button. For example, the page in Listing 3.6 includes a Cancel button that redirects the user back to the `Default.aspx` page.


http://localhost:2430 - Show Validation Groups - Mozilla Firefox
Login
Register
Login
Register
Done

Please log in to our Website.

User Name: (Required)

Password: (Required)

If you do not have a User Name, please register at our Website.

First Name:

Last Name:

FIGURE 3.4 Using validation groups.

LISTING 3.6 `ShowDisableValidation.aspx`

```
<%@ Page Language="VB" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<script runat="server">

    Protected Sub btnCancel_Click(ByVal sender As Object, ByVal e
➥As System.EventArgs)
        Response.Redirect("~/Default.aspx")
    End Sub
</script>
<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
    <title>Show Disable Validation</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>

    <asp:Label
        id="lblFirstName"
        Text="First Name:"
```

LISTING 3.6 Continued

```
        AssociatedControlID="txtFirstName"
        Runat="server" />
    <asp:TextBox
        id="txtFirstName"
        Runat="server" />
    <asp:RequiredFieldValidator
        id="reqFirstName"
        ControlToValidate="txtFirstName"
        Text="(Required)"
        Runat="server" />
    <br /><br />
    <asp:Button
        id="btnSubmit"
        Text="Submit"
        Runat="server" />
    <asp:Button
        id="btnCancel"
        Text="Cancel"
        OnClick="btnCancel_Click"
        CausesValidation="false"
        Runat="server" />

    </div>
    </form>
</body>
</html>
```

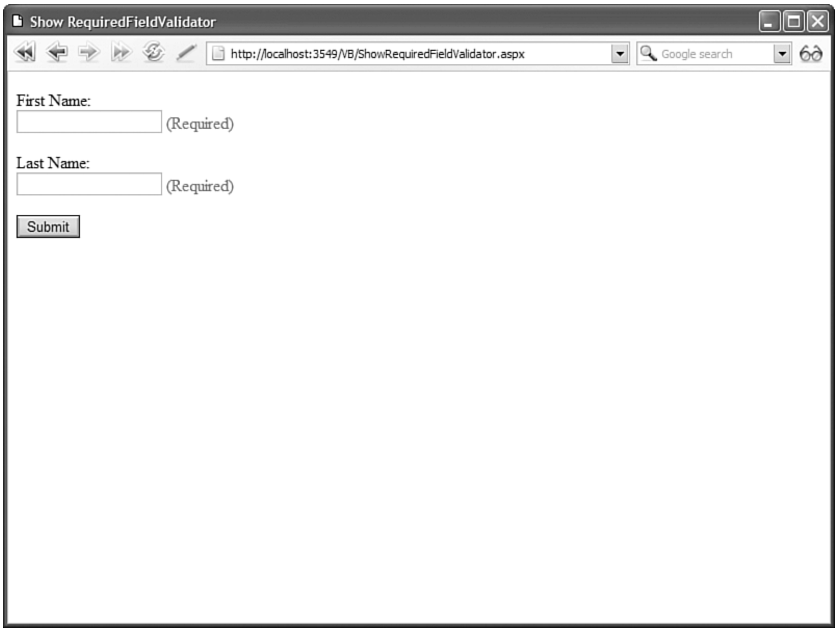
Notice that the Cancel button in Listing 3.6 includes the `CausesValidation` property with the value `False`. If the button did not include this property, then the `RequiredFieldValidator` control would prevent you from submitting the form when you clicked the Cancel button.

Using the `RequiredFieldValidator` Control

The `RequiredFieldValidator` control enables you to require a user to enter a value into a form field before submitting the form. You must set two important properties when using the `RequiredFieldValdiator` control:

- `ControlToValidate`—The ID of the form field being validated.
- `Text`—The error message displayed when validation fails.

The page in Listing 3.1 illustrates how you can use the `RequiredFieldValidator` control to require a user to enter both a first and last name (see Figure 3.5).


Show RequiredFieldValidator
http://localhost:3549/VB/ShowRequiredFieldValidator.aspx
Google search

First Name:

(Required)

Last Name:

(Required)

Submit

FIGURE 3.5 Requiring a user to enter form field values.

LISTING 3.7 ShowRequiredFieldValidator.aspx

```
<%@ Page Language="VB" %>
<!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 id="Head1" runat="server">
    <title>Show RequiredFieldValidator</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>

    <asp:Label
        id="lblFirstName"
        Text="First Name:"
        AssociatedControlID="txtFirstName"
        Runat="server" />
    <br />
    <asp:TextBox
        id="txtFirstName"
        Runat="server" />
    <asp:RequiredFieldValidator
```

LISTING 3.7 Continued

```
        id="reqFirstName"
        ControlToValidate="txtFirstName"
        Text="(Required)"
        Runat="server" />

    <br /><br />

    <asp:Label
        id="lblLastName"
        Text="Last Name:"
        AssociatedControlID="txtLastName"
        Runat="server" />
    <br />
    <asp:TextBox
        id="txtLastName"
        Runat="server" />
    <asp:RequiredFieldValidator
        id="reqLastName"
        ControlToValidate="txtLastName"
        Text="(Required)"
        Runat="server" />

    <br /><br />

    <asp:Button
        id="btnSubmit"
        Text="Submit"
        Runat="server" />

    </div>
    </form>
</body>
</html>
```

By default, the `RequiredFieldValidator` checks for a nonempty string (spaces don't count). If you enter anything into the form field associated with the `RequiredFieldValidator`, then the `RequiredFieldValidator` does not display its validation error message.

You can use the `RequiredFieldValidator` control's `InitialValue` property to specify a default value other than an empty string. For example, the page in Listing 3.8 uses a `RequiredFieldValidator` to validate a `DropDownList` control (see Figure 3.6).

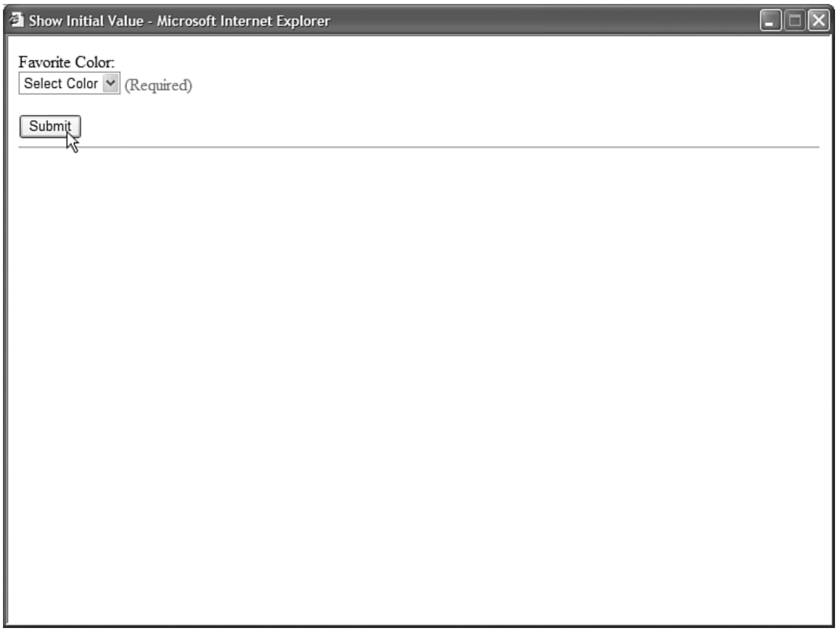


FIGURE 3.6 Using a `RequiredFieldValidator` with a `DropDownList` control.

LISTING 3.8 `ShowInitialValue.aspx`

```
<%@ Page Language="VB" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<script runat="server">

    Protected Sub btnSubmit_Click(ByVal sender As Object, ByVal e
➥As System.EventArgs)
        If Page.IsValid Then
            lblResult.Text = dropFavoriteColor.SelectedValue
        End If
    End Sub
</script>
<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
    <title>Show Initial Value</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>

    <asp:Label
```

LISTING 3.8 Continued

```
        id="lblFavoriteColor"
        Text="Favorite Color:"
        AssociatedControlID="dropFavoriteColor"
        Runat="server" />
    <br />
    <asp:DropDownList
        id="dropFavoriteColor"
        Runat="server">
        <asp:ListItem Text="Select Color" Value="none" />
        <asp:ListItem Text="Red" Value="Red" />
        <asp:ListItem Text="Blue" Value="Blue" />
        <asp:ListItem Text="Green" Value="Green" />
    </asp:DropDownList>
    <asp:RequiredFieldValidator
        id="reqFavoriteColor"
        Text="(Required)"
        InitialValue="none"
        ControlToValidate="dropFavoriteColor"
        Runat="server" />

    <br /><br />

    <asp:Button
        id="btnSubmit"
        Text="Submit"
        Runat="server" OnClick="btnSubmit_Click" />

    <hr />

    <asp:Label
        id="lblResult"
        Runat="server" />

    </div>
    </form>
</body>
</html>
```

The first list item displayed by the `DropDownList` control displays the text `"Select Color"`. If you submit the form without selecting a color from the `DropDownList` control, then a validation error message is displayed.

Notice that the `RequiredFieldValidator` control includes an `InitialValue` property. The value of the first list from the `DropDownList` control is assigned to this property.

Using the `RangeValidator` Control

The `RangeValidator` control enables you to check whether the value of a form field falls between a certain minimum and maximum value. You must set five properties when using this control:

- `ControlToValidate`—The ID of the form field being validated.
- `Text`—The error message displayed when validation fails.
- `MinimumValue`—The minimum value of the validation range.
- `MaximumValue`—The maximum value of the validation range.
- `Type`—The type of comparison to perform. Possible values are `String`, `Integer`, `Double`, `Date`, and `Currency`.

For example, the page in Listing 3.9 includes a `RangeValidator` that validates an age form field. If you do not enter an age between 5 and 100, then a validation error is displayed (see Figure 3.7).

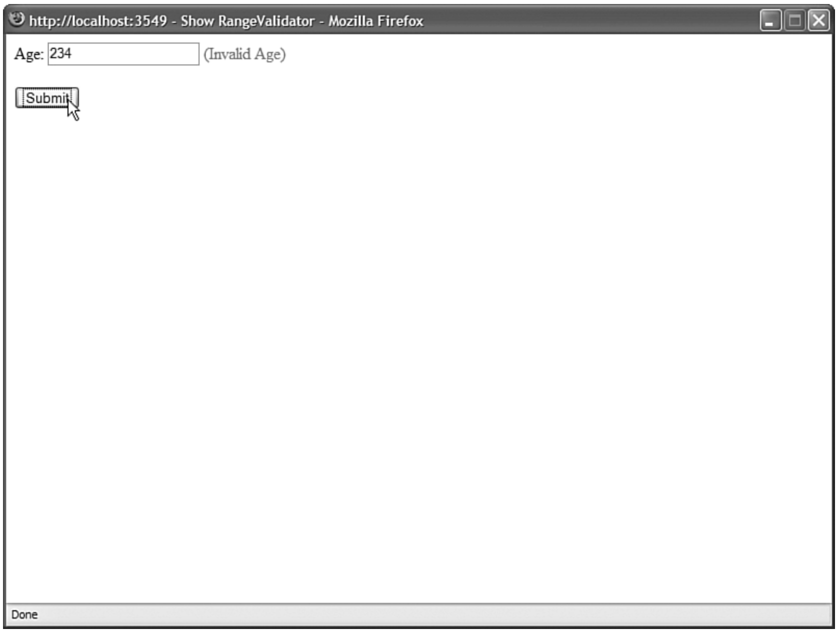

http://localhost:3549 - Show RangeValidator - Mozilla Firefox
Age: 234
(Invalid Age)
Submit
Done

FIGURE 3.7 Validating a form field against a range of values.

LISTING 3.9 `ShowRangeValidator.aspx`

```
<%@ Page Language="VB" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

LISTING 3.9 Continued

```
<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
    <title>Show RangeValidator</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>

    <asp:Label
        id="lblAge"
        Text="Age:"
        AssociatedControlID="txtAge"
        Runat="server" />
    <asp:TextBox
        id="txtAge"
        Runat="server" />
    <asp:RangeValidator
        id="reqAge"
        ControlToValidate="txtAge"
        Text="(Invalid Age)"
        MinimumValue="5"
        MaximumValue="100"
        Type="Integer"
        Runat="server" />

    <br /><br />

    <asp:Button
        id="btnSubmit"
        Text="Submit"
        Runat="server" />

    </div>
    </form>
</body>
</html>
```

If you submit the form in Listing 3.9 with an age less than 5 or greater than 100, then the validation error message is displayed. The validation message is also displayed if you enter a value that is not a number. If the value entered into the form field cannot be converted into the data type represented by the RangeValidator control's Type property, then the error message is displayed.

If you don't enter any value into the age field and submit the form, no error message is displayed. If you want to require a user to enter a value, you must associate a `RequiredFieldValidator` with the form field.

Don't forget to set the `Type` property when using the `RangeValidator` control. By default, the `Type` property has the value `String`, and the `RangeValidator` performs a string comparison to determine whether a values falls between the minimum and maximum value.

Using the `CompareValidator` Control

The `CompareValidator` control enables you to perform three different types of validation tasks. You can use the `CompareValidator` to perform a data type check. In other words, you can use the control to determine whether a user has entered the proper type of value into a form field, such as a date in a birth date field.

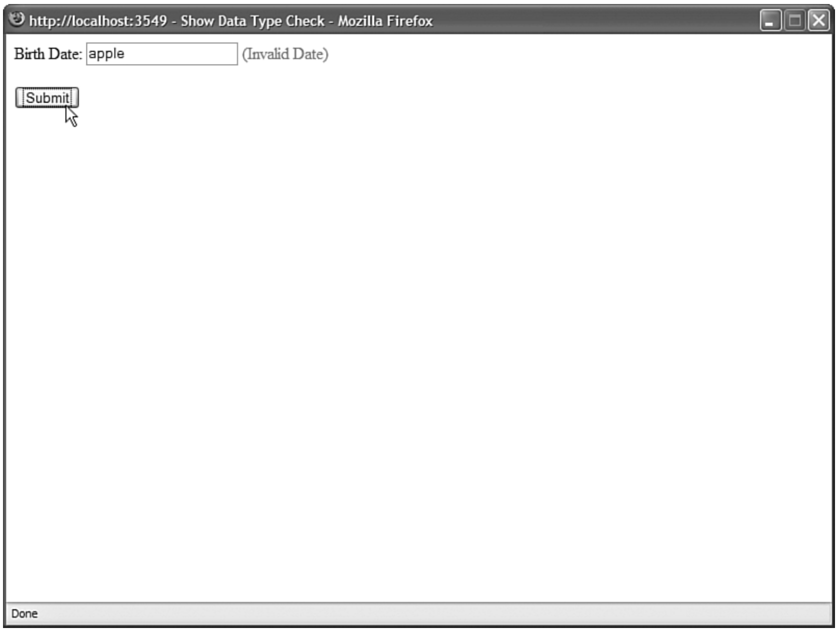
You also can use the `CompareValidator` to compare the value entered into a form field against a fixed value. For example, if you are building an auction website, you can use the `CompareValidator` to check whether a new minimum bid is greater than the previous minimum bid.

Finally, you can use the `CompareValidator` to compare the value of one form field against another. For example, you use the `CompareValidator` to check whether the value entered into the meeting start date is less than the value entered into the meeting end date.

The `CompareValidator` has six important properties:

- `ControlToValidate`—The ID of the form field being validated.
- `Text`—The error message displayed when validation fails.
- `Type`—The type of value being compared. Possible values are `String`, `Integer`, `Double`, `Date`, and `Currency`.
- `Operator`—The type of comparison to perform. Possible values are `DataTypeCheck`, `Equal`, `GreaterThan`, `GreaterThanEqual`, `LessThan`, `LessThanEqual`, and `NotEqual`.
- `ValueToCompare`—The fixed value against which to compare.
- `ControlToCompare`—The ID of a control against which to compare.

The page in Listing 3.10 illustrates how you can use the `CompareValidator` to perform a data type check. The page contains a birth date field. If you enter a value that is not a date, then the validation error message is displayed (see Figure 3.8).


http://localhost:3549 - Show Data Type Check - Mozilla Firefox
Submit
Done

Birth Date: apple (Invalid Date)

FIGURE 3.8 Performing a data type check.

LISTING 3.10 ShowDataTypeCheck.aspx

```
<%@ Page Language="VB" %>
<!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 id="Head1" runat="server">
    <title>Show Data Type Check</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>

    <asp:Label
        id="lblBirthDate"
        Text="Birth Date:"
        AssociatedControlID="txtBirthDate"
        Runat="server" />
    <asp:TextBox
        id="txtBirthDate"
        Runat="server" />
    <asp:CompareValidator
```

LISTING 3.10 Continued

```
        id="cmpBirthDate"
        Text="(Invalid Date)"
        ControlToValidate="txtBirthDate"
        Type="Date"
        Operator="DataTypeCheck"
        Runat="server" />

    <br /><br />

    <asp:Button
        id="btnSubmit"
        Text="Submit"
        Runat="server" />

    </div>
    </form>
</body>
</html>
```

Notice that the page in Listing 3.10 contains a `CompareValidator` control. Its `Type` property has the value `Date`, and its `Operator` property has the value `DataTypeCheck`. If you enter a value other than a date into the birth date field, the validation error message is displayed.

WARNING

An important limitation of the `CompareValidator` concerns how it performs a data type check. You cannot enter a long date into the form in Listing 3.10 (for example, December 25, 1966). You must enter a short date (for example, 12/25/1966). When validating currency amounts, you cannot enter the currency symbol. If these limitations concern you, you can use either the `RegularExpression` or `CustomValidator` controls to perform a more flexible data type check.

You can also use the `CompareValidator` to perform a comparison against a fixed value. For example, the page in Listing 3.11 uses a `CompareValidator` to check whether a date entered into a form field is greater than the current date (see Figure 3.9).

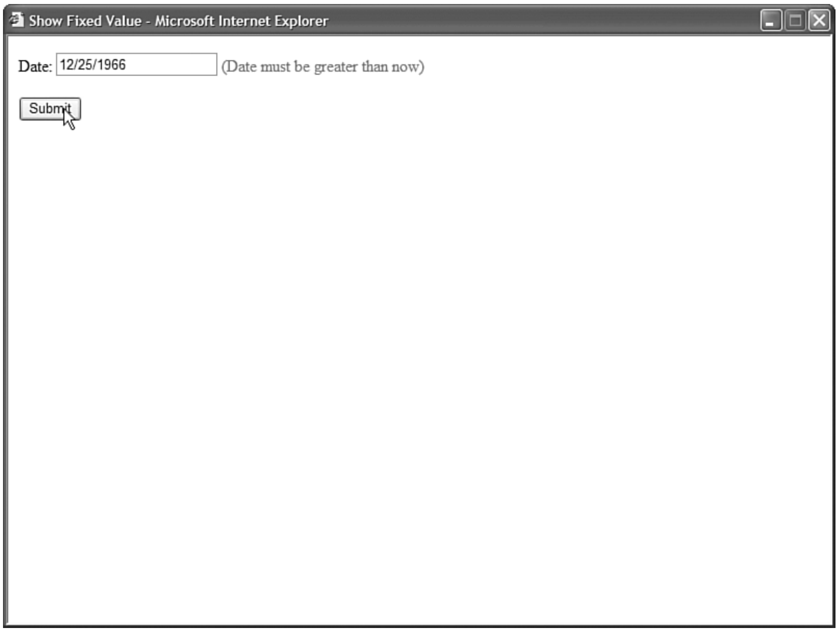
LISTING 3.11 `ShowFixedValue.aspx`

```
<%@ Page Language="VB" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<script runat="server">

    Sub Page_Load()
```

LISTING 3.11 Continued

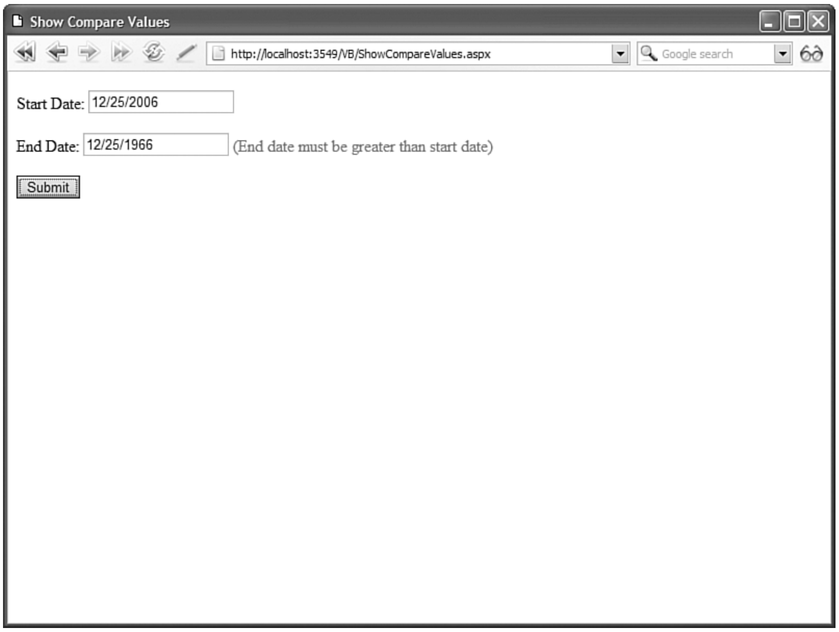
```
        cmpDate.ValueToCompare = DateTime.Now.ToString("d")
    End Sub
</script>
<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
    <title>Show Fixed Value</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>

    <asp:Label
        id="lblDate"
        Text="Date:"
        AssociatedControlID="txtDate"
        Runat="server" />
    <asp:TextBox
        id="txtDate"
        Runat="server" />
    <asp:CompareValidator
        id="cmpDate"
        Text="(Date must be greater than now)"
        ControlToValidate="txtDate"
        Type="Date"
        Operator="GreaterThan"
        Runat="server" />

    <br /><br />

    <asp:Button
        id="btnSubmit"
        Text="Submit"
        Runat="server" />

    </div>
    </form>
</body>
</html>
```

Finally, you can use a `CompareValidator` to compare the value of one form field against another form field. The page in Listing 3.12 contains a meeting start date and meeting end date field. If you enter a value into the first field that is greater than the second field, a validation error is displayed (see Figure 3.10).



Date: 12/25/1966 (Date must be greater than now)

FIGURE 3.9 Comparing a form field against a fixed value.



Start Date: 12/25/2006

End Date: 12/25/1966 (End date must be greater than start date)

FIGURE 3.10 Comparing two form fields.

LISTING 3.12 ShowCompareValues.aspx

```
<%@ Page Language="VB" %>
<!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 id="Head1" runat="server">
    <title>Show Compare Values</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>

    <asp:Label
        id="lblStartDate"
        Text="Start Date:"
        Runat="server" />
    <asp:TextBox
        id="txtStartDate"
        Runat="server" />

    <br /><br />

    <asp:Label
        id="lblEndDate"
        Text="End Date:"
        Runat="server" />
    <asp:TextBox
        id="txtEndDate"
        Runat="server" />
    <asp:CompareValidator
        id="cmpDate"
        Text="(End date must be greater than start date)"
        ControlToValidate="txtEndDate"
        ControlToCompare="txtStartDate"
        Type="Date"
        Operator="GreaterThan"
        Runat="server" />

    <br /><br />

    <asp:Button
        id="btnSubmit"
        Text="Submit"
        Runat="server" />
```

LISTING 3.12 Continued

```
    </div>
    </form>
</body>
</html>
```

Just like the `RangeValidator`, the `CompareValidator` does not display an error if you don't enter a value into the form field being validated. If you want to require that a user enter a value, then you must associate a `RequiredFieldValidator` control with the field.

Using the `RegularExpressionValidator` Control

The `RegularExpressionValidator` control enables you to compare the value of a form field against a regular expression. You can use a regular expression to represent string patterns such as email addresses, Social Security numbers, phone numbers, dates, currency amounts, and product codes.

For example, the page in Listing 3.13 enables you to validate an email address (see Figure 3.11).

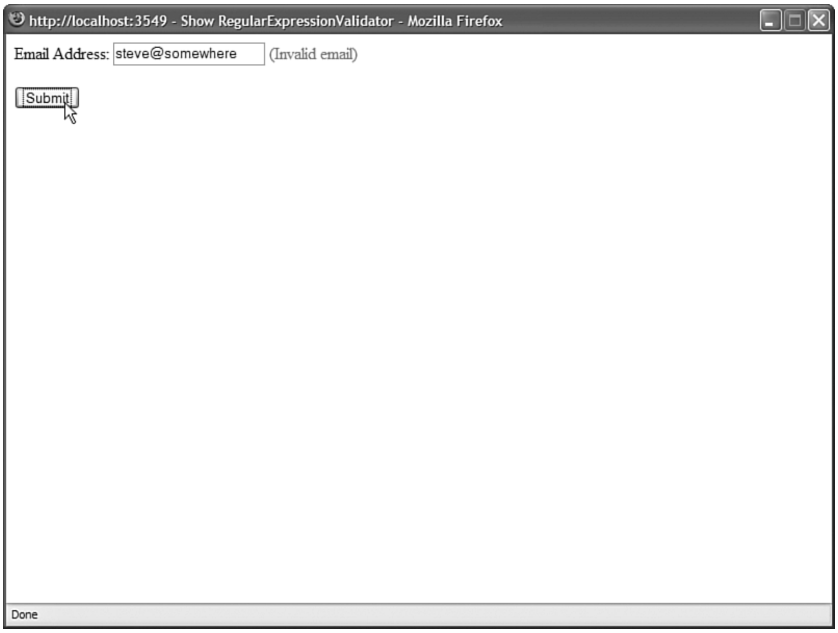

http://localhost:3549 - Show RegularExpressionValidator - Mozilla Firefox
Email Address: steve@somewhere
(Invalid email)
Submit
Done

FIGURE 3.11 Validating an email address.

LISTING 3.13 ShowRegularExpressionValidator.aspx

```
<%@ Page Language="VB" %>
<!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 id="Head1" runat="server">
    <title>Show RegularExpressionValidator</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>

    <asp:Label
        id="lblEmail"
        Text="Email Address:"
        AssociatedControlID="txtEmail"
        Runat="server" />
    <asp:TextBox
        id="txtEmail"
        Runat="server" />
    <asp:RegularExpressionValidator
        id="regEmail"
        ControlToValidate="txtEmail"
        Text="(Invalid email)"
        ValidationExpression="\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*"
        Runat="server" />

    <br /><br />

    <asp:Button
        id="btnSubmit"
        Text="Submit"
        Runat="server" />

    </div>
    </form>
</body>
</html>
```

The regular expression is assigned to the `RegularExpressionValidator` control's `ValidationExpression` property. It looks like this:

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

Regular expressions are not fun to read. This pattern matches a simple email address. The `\w` expression represents any non-whitespace character. Therefore, roughly, this regular expression matches an email address that contains non-whitespace characters, followed by an @ sign, followed by non-whitespace characters, followed by a period, followed by more non-whitespace characters.

NOTE

There are huge collections of regular expression patterns living on the Internet. The easiest way to find a good regular expression for a pattern is to simply Google it.

Just like the other validation controls, the `RegularExpressionValidator` doesn't validate a form field unless the form field contains a value. To make a form field required, you must associate a `RequiredFieldValidator` control with the form field.

VISUAL WEB DEVELOPER NOTE

If you open the property sheet for a `RegularExpressionValidator` control in Design view and select the `ValidationExpression` property, you can view a number of canned regular expressions. Visual Web Developer includes regular expressions for patterns such as email addresses, phone numbers, and Social Security numbers.

Using the `CustomValidator` Control

If none of the other validation controls perform the type of validation that you need, you can always use the `CustomValidator` control. You can associate a custom validation function with the `CustomValidator` control.

The `CustomValidator` control has three important properties:

- `ControlToValidate`—The ID of the form field being validated.
- `Text`—The error message displayed when validation fails.
- `ClientValidationFunction`—The name of a client-side function used to perform client-side validation.

The `CustomValidator` also supports one event:

- `ServerValidate`—This event is raised when the `CustomValidator` performs validation.

You associate your custom validation function with the `CustomValidator` control by handling the `ServerValidate` event.

For example, imagine that you want to validate the length of a string entered into a form field. You want to ensure that a user does not enter more than 10 characters into a multi-line TextBox control. The page in Listing 3.14 contains an event handler for a CustomValidator control's ServerValidate event, which checks the string's length.

LISTING 3.14 ShowCustomValidator.aspx

```
<%@ Page Language="VB" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<script runat="server">

    Protected Sub valComments_ServerValidate(ByVal source As Object, ByVal args
➥As System.Web.UI.WebControls.ServerValidateEventArgs)
        If args.Value.Length > 10 Then
            args.IsValid = False
        Else
            args.IsValid = True
        End If
    End Sub
</script>
<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
    <title>Show CustomValidator</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>

    <asp:Label
        id="lblComments"
        Text="Comments:"
        AssociatedControlID="txtComments"
        Runat="server" />
    <br />
    <asp:TextBox
        id="txtComments"
        TextMode="MultiLine"
        Columns="30"
        Rows="5"
        Runat="server" />
    <asp:CustomValidator
        id="valComments"
        ControlToValidate="txtComments"
        Text="(Comments must be less than 10 characters)"
        OnServerValidate="valComments_ServerValidate"
        Runat="server" />
```

LISTING 3.14 Continued

```
    <br /><br />

    <asp:Button
        id="btnSubmit"
        Text="Submit"
        Runat="server" />

    </div>
    </form>
</body>
</html>
```

The second parameter passed to the `ServerValidate` event handler is an instance of the `ServerValidateEventArgs` class. This class has two properties:

- `Value`—Represents the value of the form field being validated.
- `IsValid`—Represents whether validation fails or succeeds.
- `ValidateEmptyText`—Represents whether validation is performed when the form field being validated does not contain a value.

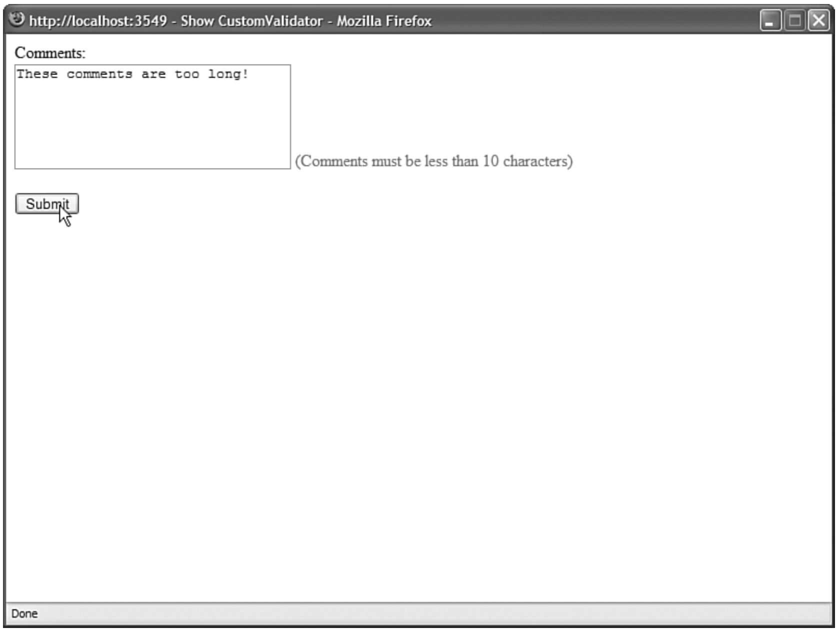
In Listing 3.14, if the string represented by the `Value` property is longer than 10 characters, then the value `False` is assigned to the `IsValid` property and validation fails. Otherwise, the value `True` is assigned to the `IsValid` property and the input field passes the validation check (see Figure 3.12).

The `ServerValidate` event handler in Listing 3.14 is a server-side function. Therefore, validation does not occur until the page is posted back to the web server. If you want to perform validation on both the client (browser) and server, then you need to supply a client-side validation function.

WARNING

If you don't associate a client validation function with a `CustomValidator` control, then the `CustomValidator` doesn't render an error message until you post the page back to the server. Because the other validation controls prevent a page from being posted if the page contains any validation errors, you won't see the error message rendered by the `CustomValidator` control until you pass every other validation check in a page.

The page in Listing 3.15 illustrates how you can associate a client-side validation function with the `CustomValidator` control. This page also checks the length of the string entered into a `TextBox` control. However, it checks the length on both the browser and server.


http://localhost:3549 - Show CustomValidator - Mozilla Firefox
Submit
Done

Comments:

These comments are too long!

(Comments must be less than 10 characters)

FIGURE 3.12 Validating field length with the `CustomValidator` control.

LISTING 3.15 `ShowCustomValidatorJS.aspx`

```
<%@ Page Language="VB" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<script runat="server">

    Protected Sub valComments_ServerValidate(ByVal source As Object,
➥ByVal args As ServerValidateEventArgs)
        If args.Value.Length > 10 Then
            args.IsValid = False
        Else
            args.IsValid = True
        End If
    End Sub
</script>
<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
    <script type="text/javascript">

    function valComments_ClientValidate(source, args)
    {
        if (args.Value.length > 10)
```

LISTING 3.15 Continued

```
            args.IsValid = false;
        else
            args.IsValid = true;
    }

    </script>
    <title>Show CustomValidator with JavaScript</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>

    <asp:Label
        id="lblComments"
        Text="Comments:"
        AssociatedControlID="txtComments"
        Runat="server" />
    <br />
    <asp:TextBox
        id="txtComments"
        TextMode="MultiLine"
        Columns="30"
        Rows="5"
        Runat="server" />
    <asp:CustomValidator
        id="valComments"
        ControlToValidate="txtComments"
        Text="(Comments must be less than 10 characters)"
        OnServerValidate="valComments_ServerValidate"
        ClientValidationFunction="valComments_ClientValidate"
        Runat="server" />

    <br /><br />

    <asp:Button
        id="btnSubmit"
        Text="Submit"
        Runat="server" />

    </div>
    </form>
</body>
</html>
```

Notice that the `CustomValidator` control in Listing 3.15 includes a `ClientValidationFunction` property. This property contains the name of a JavaScript function defined in the page's `<head>` tag.

The JavaScript validation function accepts the same two parameters as the server-side validation function. The first parameter represents the `CustomValidator` control, and the second parameter represents an object that includes both a `Value` and an `IsValid` property. The client-side function is nearly identical to the server-side function (with the important difference that it is written in JavaScript).

Unlike the `RangeValidator`, `CompareValidator`, and `RegularExpressionValidator` controls, you can validate a form field with the `CustomValidator` control even when the form field is left blank. The `CustomValidator` control includes a property named the `ValidateEmptyText` property. You can use this property to cause the `CustomValidator` control to validate a form field even when the user hasn't entered a value into the form field. For example, the page in Listing 3.16 contains a `TextBox` that requires a product code that contains exactly four characters.

LISTING 3.16 ShowValidateEmptyText.aspx

```
<%@ Page Language="VB" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<script runat="server">

    Sub valProductCode_ServerValidate(ByVal source As Object,
➥ByVal args As ServerValidateEventArgs)
        If args.Value.Length = 4 Then
            args.IsValid = True
        Else
            args.IsValid = False
        End If
    End Sub
</script>
<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
    <title>Show Validate Empty Text</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>

    <asp:Label
        id="lblProductCode"
        Text="Product Code:"
```

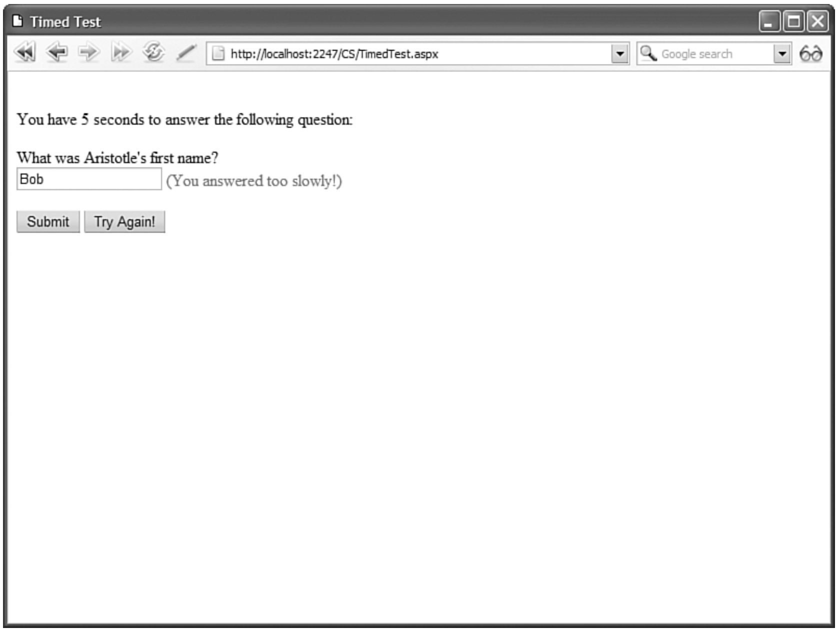
LISTING 3.16 Continued

```
        AssociatedControlID="txtProductCode"
        Runat="server" />
    <br />
    <asp:TextBox
        id="txtProductCode"
        Runat="server" />
    <asp:CustomValidator
        id="valProductCode"
        ControlToValidate="txtProductCode"
        Text="(Invalid product code)"
        ValidateEmptyText="true"
        OnServerValidate="valProductCode_ServerValidate"
        Runat="server" />

    <br /><br />

    <asp:Button
        id="btnSubmit"
        Text="Submit"
        Runat="server" />

    </div>
    </form>
</body>
</html>
```

Notice that the `CustomValidator` control in Listing 3.16 includes a `ValidateEmptyText` property which has the value `True`. If the `ValidateEmptyText` property was not included, and you submitted the form without entering any data, then no validation error would be displayed.

Finally, unlike the other validation controls, you are not required to associate the `CustomValidator` control with any form field. In other words, you don't need to include a `ControlToValidate` property.

For example, the page in Listing 3.17 contains a timed test. If you don't answer the question within five seconds, then the `CustomValidator` control displays a validation error message (see Figure 3.13).



You have 5 seconds to answer the following question:

What was Aristotle's first name?

Bob (You answered too slowly!)

FIGURE 3.13 Performing validation against no particular field.

LISTING 3.17 `TimedTest.aspx`

```
<%@ Page Language="VB" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<script runat="server">

    Sub Page_Load()
        If Not Page.IsPostBack Then
            ResetStartTime()
        End If
    End Sub

    Protected Sub btnAgain_Click(ByVal sender As Object, ByVal e
➥As System.EventArgs)
        ResetStartTime()
    End Sub

    Sub ResetStartTime()
        Session("StartTime") = DateTime.Now
    End Sub
```

LISTING 3.17 Continued

```
    Protected Sub valAnswer_ServerValidate(ByVal source As Object, ByVal args
➥As System.Web.UI.WebControls.ServerValidateEventArgs)
        Dim startTime As DateTime = CType(Session("StartTime"), DateTime)
        If startTime.AddSeconds(5) > DateTime.Now Then
            args.IsValid = True
        Else
            args.IsValid = False
        End If
    End Sub
</script>
<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
    <title>Timed Test</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>

    <p>
    You have 5 seconds to answer the following question:
    </p>

    <asp:Label
        id="lblQuestion"
        Text="What was Aristotle's first name?"
        AssociatedControlID="txtAnswer"
        Runat="server" />
    <br />
    <asp:TextBox
        id="txtAnswer"
        Runat="server" />
    <asp:CustomValidator
        id="valAnswer"
        Text="(You answered too slowly!)"
        OnServerValidate="valAnswer_ServerValidate"
        Runat="server"  />

    <br /><br />

    <asp:Button
        id="btnSubmit"
        Text="Submit"
        Runat="server" />
```

LISTING 3.17 Continued

```
    <asp:Button
        id="btnAgain"
        Text="Try Again!"
        CausesValidation="false"
        OnClick="btnAgain_Click"
        Runat="server" />

    </div>
    </form>
</body>
</html>
```

Using the `ValidationSummary` Control

The `ValidationSummary` control enables you to display a list of all the validation errors in a page in one location. This control is particularly useful when working with large forms. If a user enters the wrong value for a form field located toward the end of the page, then the user might never see the error message. If you use the `ValidationSummary` control, however, you can always display a list of errors at the top of the form.

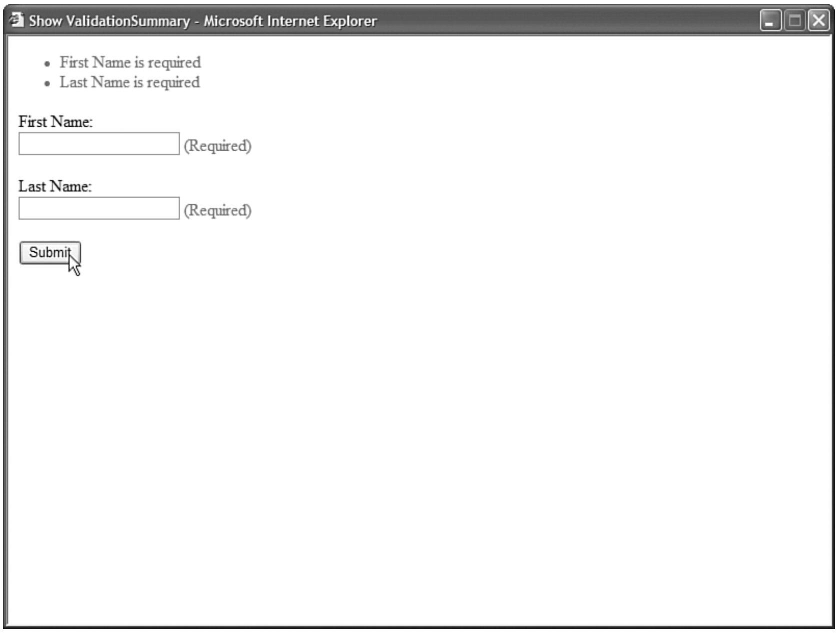
You might have noticed that each of the validation controls includes an `ErrorMessage` property. We have not been using the `ErrorMessage` property to represent the validation error message. Instead, we have used the `Text` property.

The distinction between the `ErrorMessage` and `Text` property is that any message that you assign to the `ErrorMessage` property appears in the `ValidationSummary` control, and any message that you assign to the `Text` property appears in the body of the page. Normally, you want to keep the error message for the `Text` property short (for example, `"Required!"`). The message assigned to the `ErrorMessage` property, on the other hand, should identify the form field that has the error (for example, `"First name is required!"`).

NOTE

If you don't assign a value to the `Text` property, then the value of the `ErrorMessage` property is displayed in both the `ValidationSummary` control and the body of the page.

The page in Listing 3.18 illustrates how you can use the `ValidationSummary` control to display a summary of error messages (see Figure 3.14).



- First Name is required
- Last Name is required

First Name:
(Required)

Last Name:
(Required)

FIGURE 3.14 Displaying a validation summary.

LISTING 3.18 `ShowValidationSummary.aspx`

```
<%@ Page Language="VB" %>
<!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 id="Head1" runat="server">
    <title>Show ValidationSummary</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>

    <asp:ValidationSummary
        id="ValidationSummary1"
        Runat="server" />

    <asp:Label
        id="lblFirstName"
        Text="First Name:"
        AssociatedControlID="txtFirstName"
        Runat="server" />
    <br />
```

LISTING 3.18 Continued

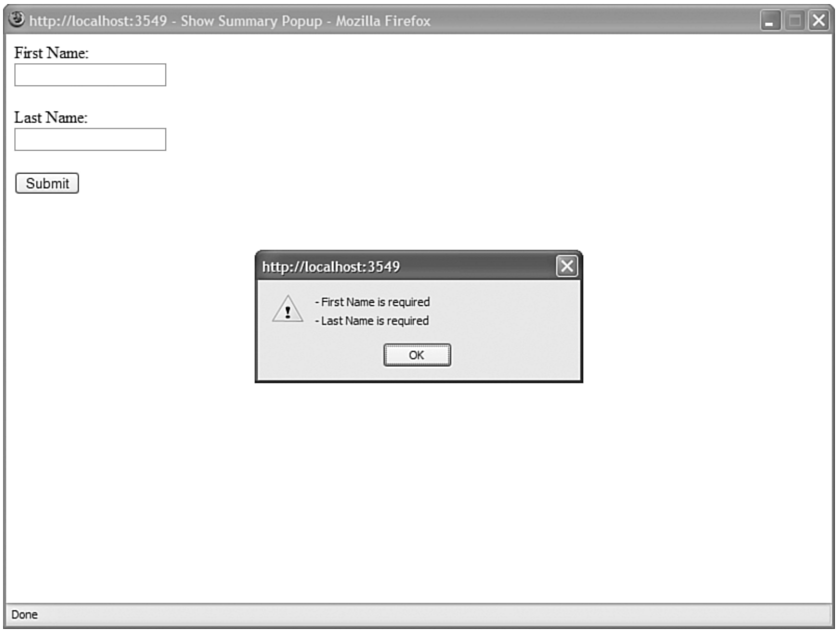
```
    <asp:TextBox
        id="txtFirstName"
        Runat="server" />
    <asp:RequiredFieldValidator
        id="reqFirstName"
        Text="(Required)"
        ErrorMessage="First Name is required"
        ControlToValidate="txtFirstName"
        Runat="server" />

    <br /><br />

    <asp:Label
        id="lblLastName"
        Text="Last Name:"
        AssociatedControlID="txtLastName"
        Runat="server" />
    <br />
    <asp:TextBox
        id="txtLastName"
        Runat="server" />
    <asp:RequiredFieldValidator
        id="reqLastName"
        Text="(Required)"
        ErrorMessage="Last Name is required"
        ControlToValidate="txtLastName"
        Runat="server" />

    <br /><br />

    <asp:Button
        id="btnSubmit"
        Text="Submit"
        Runat="server" />

    </div>
    </form>
</body>
</html>
```

If you submit the form in Listing 3.18 without entering a value for the first and last name, then validation error messages appear in both the body of the page and in the `ValidationSummary` control.

The `ValidationSummary` control supports the following properties:

- `DisplayMode`—Enables you to specify how the error messages are formatted. Possible values are `BulletList`, `List`, and `SingleParagraph`.
- `HeaderText`—Enables you to display header text above the validation summary.
- `ShowMessageBox`—Enables you to display a popup alert box.
- `ShowSummary`—Enables you to hide the validation summary in the page.

If you set the `ShowMessageBox` property to the value `True` and the `ShowSummary` property to the value `False`, then you can display the validation summary only within a popup alert box. For example, the page in Listing 3.19 displays a validation summary in an alert box (see Figure 3.15).


http://localhost:3549 - Show Summary Popup - Mozilla Firefox
First Name:
Last Name:
Submit
http://localhost:3549
OK
Done

- First Name is required
- Last Name is required

FIGURE 3.15 Displaying a validation summary in an alert box.

LISTING 3.19 `ShowSummaryPopup.aspx`

```
<%@ Page Language="VB" %>
<!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 id="Head1" runat="server">
    <title>Show Summary Popup</title>
```

LISTING 3.19 Continued

```
</head>
<body>
    <form id="form1" runat="server">
    <div>

    <asp:ValidationSummary
        id="ValidationSummary1"
        ShowMessageBox="true"
        ShowSummary="false"
        Runat="server" />

    <asp:Label
        id="lblFirstName"
        Text="First Name:"
        AssociatedControlID="txtFirstName"
        Runat="server" />
    <br />
    <asp:TextBox
        id="txtFirstName"
        Runat="server" />
    <asp:RequiredFieldValidator
        id="reqFirstName"
        ErrorMessage="First Name is required"
        ControlToValidate="txtFirstName"
        Display="None"
        Runat="server" />

    <br /><br />

    <asp:Label
        id="lblLastName"
        Text="Last Name:"
        AssociatedControlID="txtLastName"
        Runat="server" />
    <br />
    <asp:TextBox
        id="txtLastName"
        Runat="server" />
    <asp:RequiredFieldValidator
        id="reqLastName"
        ErrorMessage="Last Name is required"
        ControlToValidate="txtLastName"
        Display="None"
```

LISTING 3.19 Continued

```
        Runat="server" />

    <br /><br />

    <asp:Button
        id="btnSubmit"
        Text="Submit"
        Runat="server" />

    </div>
    </form>
</body>
</html>
```

Notice that both of the `RequiredFieldValidator` controls have their `Display` properties set to the value `None`. The validation error messages appear only in the alert box.

Creating Custom Validation Controls

In this final section, you learn how to create custom validation controls. We create two custom controls. First we create a `LengthValidator` control that enables you to validate the length of an entry in a form field. Next, we create an `AjaxValidator` control. The `AjaxValidator` control performs validation on the client by passing information back to a custom function defined on the server.

You create a new validation control by deriving a new control from the `BaseValidator` class. As its name implies, the `BaseValidator` class is the base class for all the validation controls, including the `RequiredFieldValidator` and `RegularExpressionValidator` controls.

The `BaseValidator` class is a `MustInherit` (abstract) class, which requires you to implement a single method:

- `EvaluateIsValid`—Returns true when the form field being validated is valid.

The `BaseValidator` class also includes several other methods that you can override or otherwise use. The most useful of these methods is the following:

- `GetControlValidationValue`—Enables you to retrieve the value of the control being validated.

When you create a custom validation control, you override the `EvaluateIsValid()` method and, within the `EvaluateIsValid()` method, you call `GetControlValidationValue` to get the value of the form field being validated.

Creating a `LengthValidator` Control

To illustrate the general technique for creating a custom validation control, in this section we will create an extremely simple one. It's a `LengthValidator` control, which enables you to validate the length of a form field.

The code for the `LengthValidator` control is contained in Listing 3.20.

LISTING 3.20 `LengthValidator.vb`

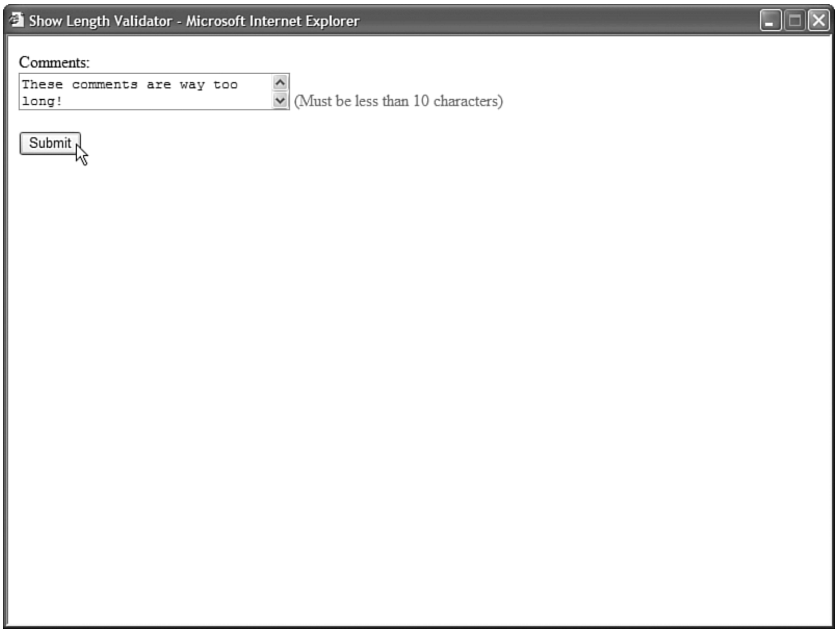
```
Imports System
Imports System.Web.UI
Imports System.Web.UI.WebControls

Namespace myControls

    ''' <summary>
    ''' Validates the length of an input field
    ''' </summary>
    Public Class LengthValidator
        Inherits BaseValidator

        Dim _maximumLength As Integer = 0

        Public Property MaximumLength() As Integer
            Get
                Return _maximumLength
            End Get
            Set(ByVal Value As Integer)
                _maximumLength = value
            End Set
        End Property

        Protected Overrides Function EvaluateIsValid() As Boolean
            Dim value As String =
➥Me.GetControlValidationValue(Me.ControlToValidate)
            If value.Length > _maximumLength Then
                Return False
            Else
                Return True
            End If
        End Function
    End Class
End Namespace
```

Listing 3.20 contains a class that inherits from the `BaseValidator` class. The new class overrides the `EvaluateIsValid` method. The value from the control being validated is retrieved with the help of the `GetControlValidationValue()` method, and the length of the value is compared against the `MaximumLength` property.

NOTE

To use the class in Listing 3.20, you need to add the class to your application's `App_Code` folder. Any class added to this special folder is automatically compiled by the ASP.NET Framework.

The page in Listing 3.21 uses the `LengthValidator` control to validate the length of a comment input field (see Figure 3.16).



These comments are way too long!

(Must be less than 10 characters)

FIGURE 3.16 Validating the length of a field with the `LengthValidator` control.

LISTING 3.21 `ShowLengthValidator.aspx`

```
<%@ Page Language="VB" %>
<%@ Register TagPrefix="custom" Namespace="myControls" %>
<!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 id="Head1" runat="server">
    <title>Show Length Validator</title>
</head>
```

LISTING 3.21 Continued

```
<body>
    <form id="form1" runat="server">
    <div>

    <asp:Label
        id="lblComments"
        Text="Comments:"
        AssociatedControlID="txtComments"
        Runat="server" />
    <br />
    <asp:TextBox
        id="txtComments"
        TextMode="MultiLine"
        Columns="30"
        Rows="2"
        Runat="server" />
    <custom:LengthValidator
        id="valComments"
        ControlToValidate="txtComments"
        Text="(Must be less than 10 characters)"
        MaximumLength="10"
        Runat="server" />

    <br /><br />

    <asp:Button
        id="btnSubmit"
        Text="Submit"
        Runat="server" />

    </div>
    </form>
</body>
</html>
```

Notice that the `LengthValidator` is registered at the top of the page with the `<%@ Register %>` directive. If you need to use the control in multiple pages in your application, then you can alternatively register the control in the `<pages>` section of your application's web configuration file.

Creating an `AjaxValidator` Control

In this section, we are going to create an extremely useful control named the `AjaxValidator` control. Like the `CustomValidator` control, the `AjaxValidator` control enables you to create a custom server-side validation function. Unlike the `CustomValidator` control, however, the `AjaxValidator` control enables you to call the custom validation function from the browser.

The `AjaxValidator` control uses AJAX (Asynchronous JavaScript and XML) to call the server-side validation function from the client. The advantage of using AJAX is that no postback to the server is apparent to the user.

For example, imagine that you are creating a website registration form and you need to validate a User Name field. You want to make sure that the User Name entered does not already exist in the database. The `AjaxValidator` enables you to call a server-side validation function from the client to check whether the User Name is unique in the database.

The code for the `AjaxValidator` control is contained in Listing 3.22.

LISTING 3.22 `AjaxValidator.vb`

```
Imports System
Imports System.Web
Imports System.Web.UI
Imports System.Web.UI.WebControls

Namespace myControls

    ''' <summary>
    ''' Enables you to perform custom validation on both the client and server
    ''' </summary>
    Public Class AjaxValidator
        Inherits BaseValidator
        Implements ICallbackEventHandler

        Public Event ServerValidate As ServerValidateEventHandler

        Dim _controlToValidateValue As String

        Protected Overrides Sub OnPreRender(ByVal e As EventArgs)
            Dim eventRef As String = Page.ClientScript.GetCallbackEventReference(
➥Me, "", "", "")

            ' Register include file
            Dim includeScript As String =
➥Page.ResolveClientUrl("~/ClientScripts/AjaxValidator.js")
            Page.ClientScript.RegisterClientScriptInclude("AjaxValidator",
➥includeScript)

            ' Register startup script
```

LISTING 3.22 Continued

```
            Dim startupScript As String =
➥String.Format("document.getElementById('{0}').evaluationfunction =
➥'AjaxValidatorEvaluateIsValid';", Me.ClientID)
            Page.ClientScript.RegisterStartupScript(Me.GetType(),
➥"AjaxValidator", startupScript, True)

            MyBase.OnPreRender(e)
        End Sub

        ''' <summary>
        ''' Only do the AJAX call on browsers that support it
        ''' </summary>
        Protected Overrides Function DetermineRenderUplevel() As Boolean
            Return Context.Request.Browser.SupportsCallback
        End Function

        ''' <summary>
        ''' Server method called by client AJAX call
        ''' </summary>
        Public Function GetCallbackResult() As String
➥Implements ICallbackEventHandler.GetCallbackResult
            Return ExecuteValidationFunction(_controlToValidateValue).ToString()
        End Function

        ''' <summary>
        ''' Return callback result to client
        ''' </summary>
        Public Sub RaiseCallbackEvent(ByVal eventArgument As String)
➥Implements ICallbackEventHandler.RaiseCallbackEvent
            _controlToValidateValue = eventArgument
        End Sub

        ''' <summary>
        ''' Server-side method for validation
        ''' </summary>
        Protected Overrides Function EvaluateIsValid() As Boolean
            Dim controlToValidateValue As String =
➥Me.GetControlValidationValue(Me.ControlToValidate)
            Return ExecuteValidationFunction(controlToValidateValue)
        End Function

        ''' <summary>
        ''' Performs the validation for both server and client
        ''' </summary>
```

LISTING 3.22 Continued

```
        Private Function ExecuteValidationFunction(ByVal controlToValidateValue
➥As String) As Boolean
            Dim args As New ServerValidateEventArgs(controlToValidateValue,
➥Me.IsValid)
            RaiseEvent ServerValidate(Me, args)
            Return args.IsValid
        End Function

    End Class

End Namespace
```

The control in Listing 3.22 inherits from the `BaseValidator` class. It also implements the `ICallbackEventHandler` interface. The `ICallbackEventHandler` interface defines two methods that are called on the server when an AJAX request is made from the client.

In the `OnPreRender()` method, a JavaScript include file and startup script are registered. The JavaScript include file contains the client-side functions that are called when the `AjaxValidator` validates a form field on the client. The startup script associates the client-side `AjaxValidatorEvaluateIsValid()` function with the `AjaxValidator` control. The client-side validation framework automatically calls this JavaScript function when performing validation.

The JavaScript functions used by the `AjaxValidator` control are contained in Listing 3.23.

LISTING 3.23 `AjaxValidator.js`

```
// Performs AJAX call back to server
function AjaxValidatorEvaluateIsValid(val)
{
    var value = ValidatorGetValue(val.controltovalidate);
    WebForm_DoCallback(val.id, value, AjaxValidatorResult, val,
AjaxValidatorError, true);
    return true;
}

// Called when result is returned from server
function AjaxValidatorResult(returnValue, context)
{
    if (returnValue == 'True')
        context.isvalid = true;
    else
        context.isvalid = false;
    ValidatorUpdateDisplay(context);
}
```

LISTING 3.23 Continued

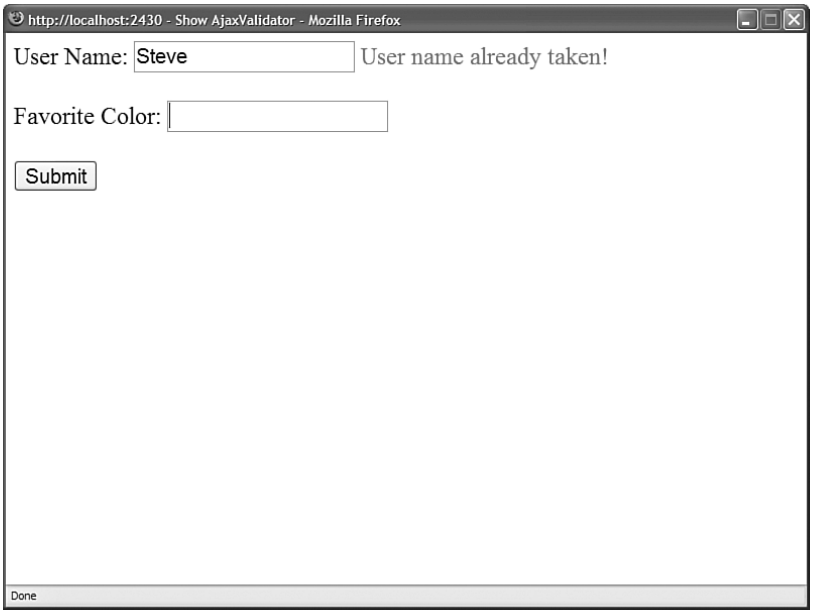
```
// If there is an error, show it
function AjaxValidatorError(message)
{
    alert('Error: ' + message);
}
```

The `AjaxValidatorEvaluateIsValid()` JavaScript method initiates an AJAX call by calling the `WebForm_DoCallback()` method. This method calls the server-side validation function associated with the `AjaxValidator` control. When the AJAX call completes, the `AjaxValidatorResult()` method is called. This method updates the display of the validation control on the client.

The page in Listing 3.24 illustrates how you can use the `AjaxValidator` control. This page handles the `AjaxValidator` control's `ServerValidate` event to associate a custom validation function with the control.

The page in Listing 3.24 contains a form that includes fields for entering a username and favorite color. When you submit the form, the values of these fields are inserted into a database table named Users.

In Listing 3.24, the validation function checks whether a username already exists in the database. If you enter a username that already exists, a validation error message is displayed. The message is displayed in the browser before you submit the form back to the server (see Figure 3.17).


http://localhost:2430 - Show AjaxValidator - Mozilla Firefox

User Name: Steve User name already taken!

Favorite Color:

FIGURE 3.17 Using the `AjaxValidator` to check whether a username is unique.

It is important to realize that you can associate any server-side validation function with the AjaxValidator. You can perform a database lookup, call a web service, or perform a complex mathematical function. Whatever function you define on the server is automatically called on the client.

LISTING 3.24 ShowAjaxValidator.aspx

```
<%@ Page Language="VB" %>
<%@ Register TagPrefix="custom" Namespace="myControls" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<%@ Import Namespace="System.Web.Configuration" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<script runat="server">

    ''' <summary>
    ''' Validation function that is called on both the client and server
    ''' </summary>
    Protected Sub AjaxValidator1_ServerValidate(ByVal source As Object,
➥ByVal args As ServerValidateEventArgs)
        If UserNameExists(args.Value) Then
            args.IsValid = False
        Else
            args.IsValid = True
        End If
    End Sub

    ''' <summary>
    ''' Returns true when user name already exists
    ''' in Users database table
    ''' </summary>
    Private Function UserNameExists(ByVal userName As String) As Boolean
        Dim conString As String =
➥WebConfigurationManager.ConnectionStrings("UsersDB").ConnectionString
        Dim con As New SqlConnection(conString)
        Dim cmd As New SqlCommand("SELECT COUNT(*) FROM Users
➥WHERE UserName=@UserName", con)
        cmd.Parameters.AddWithValue("@UserName", userName)
        Dim result As Boolean =  False
        Using con
            con.Open()
            Dim count As Integer = CType(cmd.ExecuteScalar(), Integer)
            If count > 0 Then
                result = True
            End If
        End Using
```

LISTING 3.24 Continued

```
        Return result
    End Function

    ''' <summary>
    ''' Insert new user name to Users database table
    ''' </summary>
    Protected Sub btnSubmit_Click(ByVal sender As Object, ByVal e As EventArgs)
        Dim conString As String =
➥WebConfigurationManager.ConnectionStrings("UsersDB").ConnectionString
        Dim con As New SqlConnection(conString)
        Dim cmd As New SqlCommand("INSERT Users (UserName,FavoriteColor)
➥VALUES (@UserName,@FavoriteColor)", con)
        cmd.Parameters.AddWithValue("@UserName", txtUserName.Text)
        cmd.Parameters.AddWithValue("@FavoriteColor", txtFavoriteColor.Text)
        Using con
            con.Open()
            cmd.ExecuteNonQuery()
        End Using
        txtUserName.Text = String.Empty
        txtFavoriteColor.Text = String.Empty
    End Sub
</script>
<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
    <title>Show AjaxValidator</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>

    <asp:Label
        id="lblUserName"
        Text="User Name:"
        AssociatedControlID="txtUserName"
        Runat="server" />
    <asp:TextBox
        id="txtUserName"
        Runat="server" />
    <custom:AjaxValidator
        id="AjaxValidator1"
        ControlToValidate="txtUserName"
        Text="User name already taken!"
        OnServerValidate="AjaxValidator1_ServerValidate"
```

LISTING 3.24 Continued

```
        Runat="server" />

    <br /><br />
    <asp:Label
        id="lblFavoriteColor"
        Text="Favorite Color:"
        AssociatedControlID="txtFavoriteColor"
        Runat="server" />
    <asp:TextBox
        id="txtFavoriteColor"
        Runat="server" />

    <br /><br />
    <asp:Button
        id="btnSubmit"
        Text="Submit"
        Runat="server" OnClick="btnSubmit_Click" />

    </div>
    </form>
</body>
</html>
```

Summary

In this chapter, you learned how to perform form validation with the ASP.NET 2.0 Framework. First, you were provided with an overview of all the standard validation controls. You learned how to highlight validation error messages and how to take advantage of validation groups to simulate multiple forms in a single page.

In the final section of this chapter, you learned how to create custom validation controls by deriving new controls from the `BaseValidator` control. We created both a custom `LengthValidator` and `AjaxValidator` control.