6

Validation Controls in ASP.NET

In Depth

The Validation controls are the controls used for validating the data entered in an input control, such as the TextBox control of a Web page.

When a user enters data on a Web page and submits the page to the server, the Validation controls are invoked to check the data entered by the user. If any of the data is invalid data, the Validation control displays an error message on the screen. The error message is defined as a property value of the Validation control. The data being entered is validated each time it is entered by the user, and the error message disappears only when the data is valid. Validation controls help save time and enhance the efficiency of the applications by validating the data after the page is sent to the server but before any event handlers are called. The following Validation controls are discussed in this chapter:

- The RequiredFieldValidator control
- The RangeValidator control
- The RegularExpressionValidator control
- The CompareValidator control
- The CustomValidator control
- The ValidationSummary control

Let's explore these controls in detail, starting with the BaseValidator class, which is the base class of all the Validation controls.

The BaseValidator Class

The System.Web.UI.WebControls.BaseValidator class provides basic implementation required for all the Validation controls. The inheritance hierarchy of the BaseValidator class is:

```
System.Object
   System.Web.UI.Control
         System.Web.UI.WebControls.WebControl
                 System.Web.UI.WebControls.Label
                         System.Web.UI.WebControls.BaseValidator
```

Noteworthy public properties of the BaseValidator class are given in Table 6.1:

Table 6.1: Noteworthy Public Properties of the BaseValidator Class

Property	Description
ControlToValidate	Handles an input control, such as the TextBox control, which needs to be validated
Display	Handles the behavior of the error message in a Validation control
EnableClientScript	Handles a value indicating whether or not client-side validation is enabled
Enabled	Handles a value that indicates whether or not the Validation control is enabled or not
ErrorMessage	Handles the text for the error message displayed in a ValidationSummary control when validation fails
ForeColor	Handles the color of the message displayed when validation fails
IsValid	Handles a value that indicates if the associated input control passes validation or not
SetFocusOnError	Handles a value that indicates if the focus is set on the control or not, specified by the ControlToValidate property when validation fails
Text	Handles the text displayed in the Validation control when the validation fails
ValidationGroup	Handles the name of the validation group to which this Validation control belongs

NOTE

The ValidationGroup property, when set, validates only the Validation controls within the specified group when the control is posted back to the server

Noteworthy public methods of the `BaseValidator` class are listed in Table 6.2:

Table 6.2: Noteworthy Public Methods of the BaseValidator Class

Method	Description
`Validate()`	Helps in performing validation on the associated input control and updates the `IsValid` property
`GetValidationProperty()`	Helps in determining the validation property of a control, if it exists

Now, let's discuss the Validation controls one by one in detail.

The RequiredFieldValidator Control

The `RequiredFieldValidator` control is the simplest control, and is used to ensure that the user has entered the data into the input control, such as `TextBox` to which it is bound. For example, if you are entering data, for buying a T-shirt, in an input control, it is necessary to enter the size of the T-shirt that you want to buy. If you do not enter a value, the Validation control displays an error message. The inheritance hierarchy for the `RequiredFieldValidator` class is:

```
System.Object
   System.Web.UI.Control
         System.Web.UI.WebControls.WebControl
                 System.Web.UI.WebControls.Label
                         System.Web.UI.WebControls.BaseValidator
                                 System.Web.UI.WebControls.RequiredFieldValidator
```

The `RequiredFieldValidator` class has no non-inherited methods or events and it inherits the properties and methods of the `BaseValidator` class that are listed in Tables 6.1 and 6.2. Noteworthy public property of the `RequiredFieldValidator` class is listed in Table 6.3:

Table 6.3: Noteworthy Public Property of the RequiredFieldValidator Class

Property	Description
`InitialValue`	Handles the initial value of the associated input control

The RangeValidator Control

The `RangeValidator` control checks whether or not the value of an input control is inside a specified range of values. It has the following three properties:

- **ControlToValidate**—Contains the input control to validate
- **MinimumValue**—Holds the minimum value of the valid range
- **MaximumValue**—Holds the maximum value of the valid range

If one of these properties is set, then the other property must also be set.

Do not forget to set the `Type` property to the data type of the values. The following data types can be used for the values:

- **String**—A string data type
- **Integer**—An integer data type
- **Double**—A double data type
- **Date**—A date data type
- **Currency**—A currency data type

The inheritance hierarchy for the RangeValidator class is:

```
System.Object
  System.Web.UI.Control
        System.Web.UI.WebControls.WebControl
              System.Web.UI.WebControls.Label
                    System.Web.UI.WebControls.BaseValidator
                          System.Web.UI.WebControls.BaseCompareValidator
                                System.Web.UI.WebControls.RangeValidator
```

Noteworthy public properties of the `RangeValidator` class are listed in Table 6.4:

Table 6.4: Noteworthy Public Properties of the RangeValidator Class

Property	Description
`MaximumValue`	Obtains or sets the maximum value of the validation range for the `RangeValidator` control
`MinimumValue`	Obtains or sets the minimum value of the validation range for the `RangeValidator` control

The RegularExpressionValidator Control

Regular expressions are used to check whether or not the text matches a certain pattern. The `RegularExpressionValidator` control validates whether the value in the input control (`text box`) matches the format specified by the regular expression, such as a US telephone number or a date format. The `RegularExpressionValidator` control exists within the `System.Web.UI.WebControls` namespace.

In general, regular expressions are made up of text with an embedded code listing that starts with a backslash `(\)` as well as other control code listings. The following is an example:

```
\b[A-Za-z]+\b
```

The code for a word boundary (where a word ends or starts) is `\b` and a `character` class is a set of characters surrounded by `[` and `]`. This allows you to specify what characters you want to accept. Therefore, this regular expression matches a word consisting of uppercase or lowercase letters only. The `+` sign matches the previous character or the subexpression one or more times, so we are matching one or more uppercase and/or lowercase letters here.

NOTE

The + sign in the RegularExpressionValidator Control helps in matching the previous character or the subexpression one or more times. For example, the expression zo+ will match zo and zoo but not z.

For example, the regular expression to determine if the text matches a valid `e-mail address` is:

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

In the preceding expression, `\w` stands for a word character (such as letters, and underscores) and `*` sign matches the previous character or subexpression zero or more times. For example, the expression zo* will match z and zoo.

ASP.NET 4.0 includes some pre-built regular expressions that you can use to match well-known sequences of characters, such as a French phone number, P.R.C. social security number, US social security number, e-mail address, and Internet URL address. Regular expressions are complex to understand, as they require in-depth knowledge of the character relations required during custom expression definition.

The inheritance hierarchy for the `RegularExpressionValidator` class is:

```
System.Object
  System.Web.UI.Control
        System.Web.UI.WebControls.WebControl
              System.Web.UI.WebControls.Label
                    System.Web.UI.WebControls.BaseValidator
                          System.Web.UI.WebControls.RegularExpressionValidator
```

The class has no non-inherited methods or events. Noteworthy public property of the `RegularExpressionValidator` class is listed in Table 6.5:

Table 6.5: Noteworthy Public Property of the RegularExpressionValidator Class	
Property	**Description**
`ValidationExpression`	Obtains or sets the regular expression that you want to match data against for validation

The CompareValidator Control

The CompareValidator control is used to compare the value entered by a user into one input control with the value entered into another input control or with an already existing value. The `CompareValidator` control exists within the `System.Web.UI.WebControls` namespace. Table 6.6 lists the various `Operator` property types, specifying the different types of comparisons:

Table 6.6: Operator Properties of the CompareValidator Control	
Type	**Description**
`Equal`	Checks whether the compared values are equal
`NotEqual`	Checks that controls are not equal to each other
`GreaterThan`	Checks for a greater than relationship
`GreaterThanEqual`	Checks for a greater than or equal to relationship
`LessThan`	Checks for a less than relationship
`LessThanEqual`	Checks for a less than or equal to relationship
`DataTypeCheck`	Compares data types between the value entered into the data-entry control being validated and the data type specified by the Type property

The `Type` property is used to specify the data type of both the comparison values, where `String` is the default type. Both values are automatically converted to the String data type before the comparison operation is performed. The different data types that can be used are as follows:

- **String**—A string data type
- **Integer**—An integer data type
- **Double**—A double data type
- **Date**—A date data type
- **Currency**—A currency data type

Noteworthy public properties of the `CompareValidator` class are given in Table 6.7:

Table 6.7: Noteworthy Public Properties of the CompareValidator Class	
Property	**Description**
`ControlToCompare`	Obtains or sets the data entry control, which has to be compared with the data-entry control being validated
`Operator`	Obtains or sets the comparison operation to perform
`ValueToCompare`	Obtains or sets a constant value to compare with the value entered by a user in the data entry control being validated

The CustomValidator Control

The `CustomValidator` control is used to customize and implement data validation as per your requirement. Occasionally, you might need to validate your data in a way that is not possible directly with the help of existing Validation controls. For example, if you want to find out whether a number is odd or even, you cannot use an existing Validation control for doing that because this functionality is not included in any of the `Validation` controls. To solve this problem, you need a Validation control that can be customized to solve this problem. The

CustomValidator control exists within the System.Web.UI.WebControls namespace. The inheritance hierarchy for the CustomValidator class is:

```
System.Object
  System.Web.UI.Control
        System.Web.UI.WebControls.WebControl
              System.Web.UI.WebControls.Label
                    System.Web.UI.WebControls.BaseValidator
                          System.Web.UI.WebControls.CustomValidator
```

The CustomValidator control checks whether or not the input you have given, such as prime, even, or odd number, matches a given condition. Noteworthy public properties of the CustomValidator class are listed in Table 6.8:

Table 6.8: Noteworthy Public Properties of the CustomValidator Class

Property	Description
ClientValidationFunction	Obtains or sets the name of the custom client-side script function used for validation
ValidateEmptyText	Obtains or sets a Boolean value indicating whether empty text should be validated or not

To use the CustomValidator control in a Web application, first create a function using a scripting language, such as JavaScript or VBScript (both are supported by Internet Explorer), in the source file of the Web page. You can then set the ClientValidationFunction property of the CustomValidator control to the name of the function you have created. The CustomValidator control validates the data entered by the user in the data entry control at the runtime based on the code written inside this function.

A noteworthy public event of the CustomValidator class is listed in Table 6.9:

Table 6.9: Noteworthy Public Event of the CustomValidator Class

Event	Description
ServerValidate	Occurs when validation takes place on the server

The ValidationSummary Control

The ValidationSummary control collects all the Validation control error messages (known as summary) and displays it collectively on the screen. The display of the summary, which can be a list, a bulleted list, or a single paragraph, can be set by using the DisplayMode property. You can also specify whether the summary should be displayed in the Web page or in a message box by setting the ShowSummary and ShowMessageBox properties, respectively. The ValidationSummary control exists within the System.Web.UI.WebControls namespace.

The inheritance hierarchy for the ValidationSummary class is:

```
System.Object
  System.Web.UI.Control
        System.Web.UI.WebControls.WebControl
              System.Web.UI.WebControls.ValidationSummary
```

Noteworthy public properties of the ValidationSummary class are listed in Table 6.10:

Table 6.10: Noteworthy Public Properties of the ValidationSummary Class

Property	Description
DisplayMode	Obtains or sets the display mode of the ValidationSummary control
EnableClientScript	Obtains or sets a value indicating whether the ValidationSummary control updates itself using the client-side script
ForeColor	Obtains or sets the foreground color of the control
HeaderText	Obtains or sets the header text displayed at the top of the summary

Table 6.10: Noteworthy Public Properties of the ValidationSummary Class	
Property	**Description**
`ShowMessageBox`	Obtains or sets a value showing whether the validation summary is displayed in a message box
`ShowSummary`	Obtains or sets a value showing whether the validation summary is displayed in line
`ValidationGroup`	Obtains or sets the group of controls for which the `ValidationSummary` object displays validation messages

While using a `ValidationSummary` control in a Web application, if you want to display all the error messages on the same Web page, you must set the `ShowSummary` property of the `ValidationSummary` control to `True`. In case you want to display the error messages in a message box, set the `ShowMessageBox` property of the `ValidationSummary` control to `True`. By setting both these properties to `True`, you can display the error messages in both ways. You can also set the mode in which you want to display the error messages, by setting the `DisplayMode` property of the `ValidationSummary` control. The possible values for this property are `List`, `BulletList`, and `SingleParagraph`.

Immediate Solutions

Using the RequiredFieldValidator Control

To add a `RequiredFieldValidator` control on a Web page (Default.aspx), add the following code to the source code of the Web page:

```
<asp:RequiredFieldValidator ID="RequiredFieldValidator1" runat="server"
  ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValidator>
```

Now, let's create an application named RequiredFieldValidatorCS (also available on CD-ROM). You can add a `RequiredFieldValidator` control to a Web page either by dragging and dropping it from the Standard tab of the Toolbox or by double-clicking this control in the Toolbox.

NOTE

The applications in this chapter are created using the ASP.NET Empty Web Site template in the New Web Site dialog box. After creating the application, you need to add the Default.aspx page to the application.

In this example, we are adding two `RequiredFieldValidator` controls on a Web page for validating two `TextBox` controls. The `RequiredFieldValidator` controls ensure that the two `TextBox` controls are not kept empty by the user at runtime. Listing 6.1 shows the Default.aspx page for the `RequiredFieldValidator` control:

Listing 6.1: Showing the Code for the `Default.aspx` Page

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"
 Inherits="_Default" %>

<!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">
    <title>RequiredFieldValidator Control Example</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
    <asp:Label ID="Label1" runat="server"
         Text="RequiredField Validator Control Example" Font-Bold="True"
         Font-Size="Medium" Font-Underline="True"></asp:Label>
         <br />
         <br />
         <asp:Label ID="Label2" runat="server" Font-Bold="True" Font-Size="Small"
         Text="User name:"></asp:Label>
           
         <asp:TextBox ID="TextBox1" runat="server" Height="22px"
                Width="175px"></asp:TextBox>
         <asp:RequiredFieldValidator ID="RequiredFieldValidator1" runat="server"
         ControlToValidate="TextBox1" ErrorMessage="User name can not be
                empty"></asp:RequiredFieldValidator>
         <br />
         <br />

         <asp:Label ID="Label3" runat="server" Font-Bold="True" Font-Size="Small"
         Text="Password:"></asp:Label>
             
         <asp:TextBox ID="TextBox2" runat="server" Height="22px" TextMode="Password"
         Width="175px"></asp:TextBox>
         <asp:RequiredFieldValidator ID="RequiredFieldValidator2" runat="server"
```

```
            ControlToValidate="TextBox2" ErrorMessage="Password can not be
                    empty"></asp:RequiredFieldValidator>
            <br />
            <br />
            <br />

                         
                                
                        
            <asp:Button ID="Button1" runat="server" BackColor="Black" Font-Bold="True"
            Font-Size="Medium" ForeColor="White" Text="Submit" />

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

Now, run the application by pressing the F5 key. If you click the Submit button without entering data in the two `TextBox` controls, both the `RequiredFieldValidator` controls display error messages, as shown in Figure 6.1:

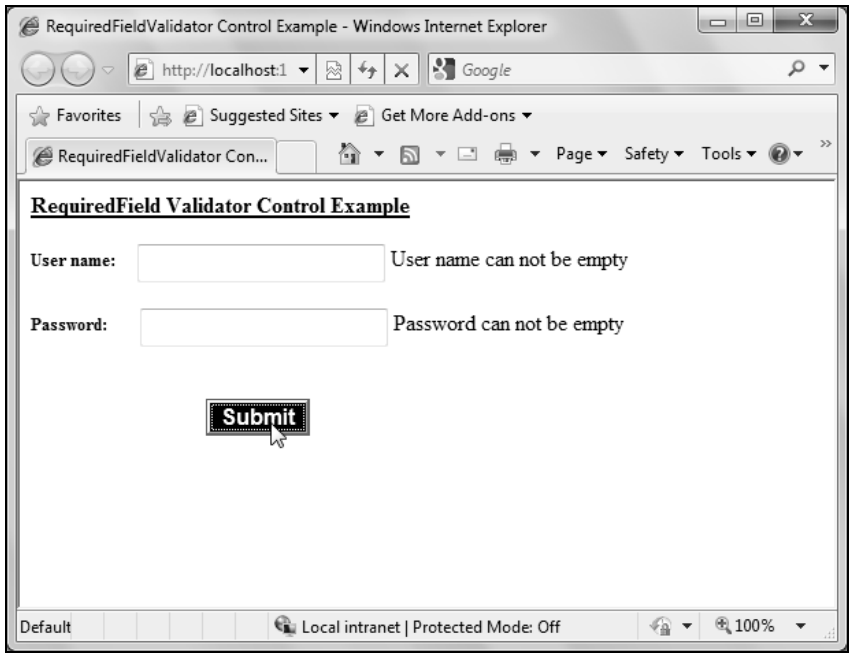

Figure 6.1: Showing Error Messages in the RequiredFieldValidator Controls

Using the RangeValidator Control

When you add a `RangeValidator` control on a Web page (Default.aspx), it adds the following code to the source code of the Web page:

```
    <asp:RangeValidator ID="RangeValidator1" runat="server"
      ErrorMessage="RangeValidator"></asp:RangeValidator>
```

Now, let's create an application named RangeValidatorCS (also available on CD-ROM). In this example, we are adding two `RangeValidator` controls on a Web page for validating two `TextBox` controls. The first `RangeValidator` control ensures that the age entered in the first `TextBox` control is in the range 20-40 yrs. The

second `RangeValidator` control ensures that the date of birth is in the range 1988/1/1 to 2028/1/1. Listing 6.2 shows the `Default.aspx` page for the `RangeValidator` control:

Listing 6.2: Showing the Code for the `Default.aspx` Page

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"
 Inherits="_Default" %>

<!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">
    <title>RangeValidator Control Example</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
    <asp:Label ID="Label1" runat="server" Text="RangeValidator Control Example"
         Font-Bold="True" Font-Size="Medium" Font-Underline="True"></asp:Label>
         <br />
         <br />
         <br />
         <asp:Label ID="Label2" runat="server" Font-Bold="True" Font-Size="Small"
         Text="Your Age:"></asp:Label>
           
         <asp:TextBox ID="TextBox1" runat="server" Width="172px"></asp:TextBox>
         <asp:RangeValidator ID="RangeValidator1" runat="server"
         ControlToValidate="TextBox1"
         ErrorMessage="Invalid Age. Please enter the age between 20 to 40."
         MaximumValue="40" MinimumValue="20" Type="Integer"></asp:RangeValidator>
          <br />
         <asp:RequiredFieldValidator ID="RequiredFieldValidator1" runat="server"
         ControlToValidate="TextBox1" ErrorMessage="This field can not be
                 blank."></asp:RequiredFieldValidator>
         <br />
         <br />
         <asp:Label ID="Label3" runat="server" Font-Bold="True" Font-Size="Small"
         Text="Date of Birth (YYYY/MM/DD format) :"></asp:Label>
            
         <asp:TextBox ID="TextBox2" runat="server" Width="172px"></asp:TextBox>
         <asp:RangeValidator ID="RangeValidator2" runat="server"
         ControlToValidate="TextBox2"
         ErrorMessage="Invalid DOB. It must be between 1988/1/1 to 2028/1/1."
         MaximumValue="2028/1/1"                                    MinimumValue="1988/1/1"
         Type="Date"></asp:RangeValidator>
         <br />
         <asp:RequiredFieldValidator ID="RequiredFieldValidator2" runat="server"
         ControlToValidate="TextBox2" ErrorMessage="This field can not be
                 blank."></asp:RequiredFieldValidator>
         <br />
         <br />
                      
                      
                    
         <asp:Button ID="Button1" runat="server" BackColor="Black" Font-Bold="True"
         Font-Size="Small" ForeColor="White" Text="Submit" />

    </div>
    </form>
```

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

Now, execute the application by pressing the F5 key. Enter values in the text boxes and click the Submit button. If the values are outside the specified range in the `TextBox` controls the `RangeValidator` controls display error messages, as shown in Figure 6.2:

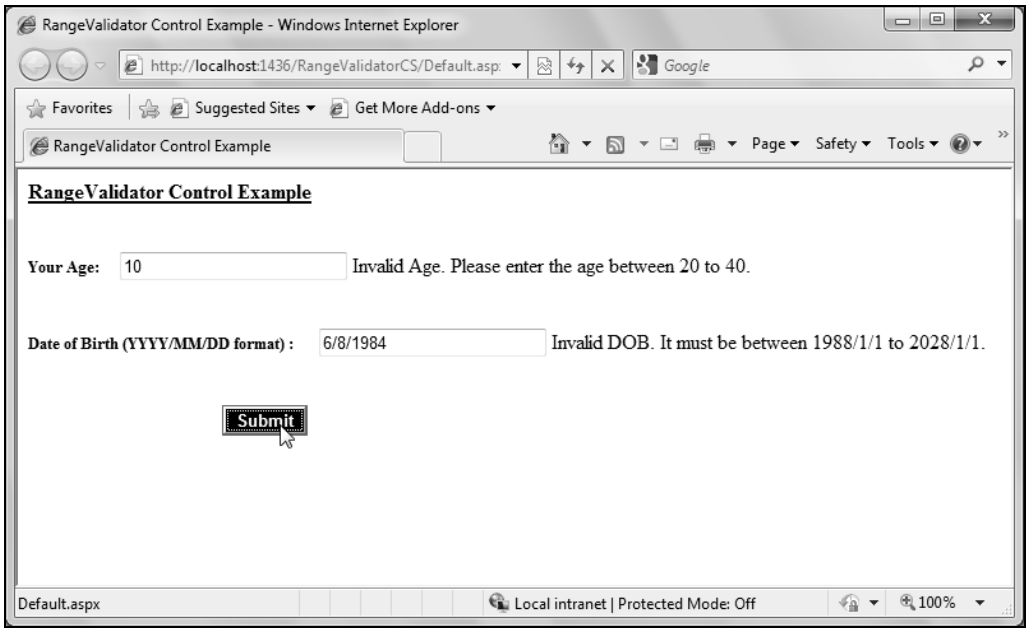

Figure 6.2: Showing Error Messages in the RangeValidator Controls

NOTE

It is mandatory to use a RequiredFieldValidator control with the RangeValidator, RegularExpressionValidator, CompareValidator, and CustomValidator controls because the latter do not work on blank input controls.

Using the RegularExpressionValidator Control

When you add a RegularExpressionValidator control on a Web page (Default.aspx), it adds the following code to the source code of the Web page:

```
<asp:RegularExpressionValidator ID="RegularExpressionValidator1" runat="server"
  ErrorMessage="RegularExpressionValidator"></asp:RegularExpressionValidator>
```

Now, let's create an application named RegularExpressionValidatorCS (also available on CD-ROM). In this example, we are taking two `RegularExpressionValidator` controls on a Web page for validating two `TextBox` controls. It ensures that the data (URL and email address) entered in both the `TextBox` controls is valid. Listing 6.3 shows the Default.aspx page for the RegularExpressionValidator control:

Listing 6.3: Showing the Code for the `Default.aspx` Page

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"
 Inherits="_Default" %>

<!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">
    <title>RegularExpressionValidator Control Example</title>
</head>
```

```
    <body>
        <form id="form1" runat="server">
        <div>
        <asp:Label ID="Label1" runat="server" Font-Bold="True" Font-Size="Medium"
             Font-Underline="True" Text="RegularExpressionValidator Control
                     Example"></asp:Label>
             <br />
             <br />
             <br />
             <asp:Label ID="Label2" runat="server" Font-Bold="True" Font-Size="Small"
             Text="Enter URL:"></asp:Label>
                         
                           
             <asp:TextBox ID="TextBox1" runat="server" Width="177px"></asp:TextBox>
             <asp:RegularExpressionValidator ID="RegularExpressionValidator1" runat="server"
             ControlToValidate="TextBox1" ErrorMessage="Not a Valid URL."
             ValidationExpression="http(s)?://([\w-]+\.)+[\w-]+(/[\w-
             ./?%&amp;=]*)?"></asp:RegularExpressionValidator>
             <br />
             <asp:RequiredFieldValidator ID="RequiredFieldValidator1" runat="server"
             ControlToValidate="TextBox1" ErrorMessage="This field can not be
                     blank."></asp:RequiredFieldValidator>
             <br />
             <br />
             <asp:Label ID="Label3" runat="server" Font-Bold="True" Font-Size="Small"
             Text="Enter E-mail address:"></asp:Label>
               
             <asp:TextBox ID="TextBox2" runat="server" Width="177px"></asp:TextBox>
             <asp:RegularExpressionValidator ID="RegularExpressionValidator2" runat="server"
             ControlToValidate="TextBox2" ErrorMessage="Not a Valid E-mail address."
             ValidationExpression="\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-
             .]\w+)*"></asp:RegularExpressionValidator>
             <br />
             <asp:RequiredFieldValidator ID="RequiredFieldValidator2" runat="server"
             ControlToValidate="TextBox2" ErrorMessage="This field can not be
                     blank."></asp:RequiredFieldValidator>
             <br />
             <br />
             <br />
                         
                          
                      
             <asp:Button ID="Button1" runat="server" BackColor="Black" Font-Bold="True"
             Font-Size="Small" ForeColor="White" Text="Submit" Width="60px" />

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

Now, press the F5 key to execute the application and enter the values in the `TextBox` controls and click the Submit button. If the values are not in the specified format then the `RegularExpressionValidator` controls display error messages, as shown in Figure 6.3:

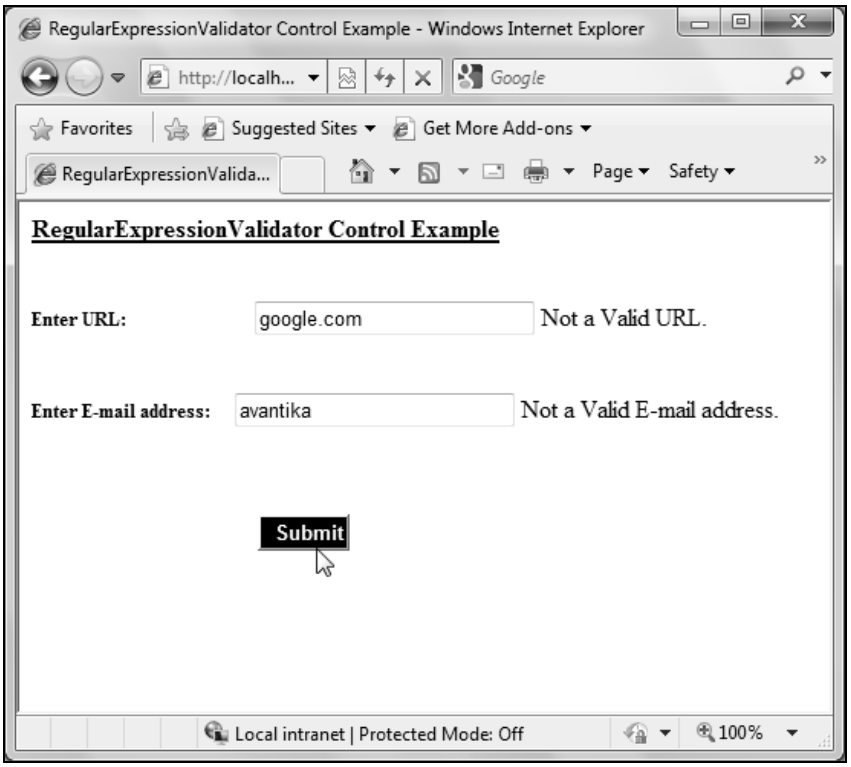

Figure 6.3: Displaying Error Message in the RegularExpressionValidator Controls

Using the CompareValidator Control

When you add a CompareValidator control on a Web page (Default.aspx), it adds the following code to the source code of the Web page:

```
<asp:CompareValidator ID="CompareValidator1" runat="server"
 ErrorMessage="CompareValidator"></asp:CompareValidator>
```

Now, let's create an application named CompareValidatorCS (available in the CD-ROM as well).

In this example, we take a `CompareValidator` control on a Web page, which validates the third `TextBox` control. It ensures that the data (password) entered in the third `TextBox` control is exactly the same as the data (`password`) entered in the second `TextBox` control. Listing 6.4 shows the Default.aspx page for the CompareValidator control:

Listing 6.4: Showing the Code for the `Default.aspx` Page

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"
 Inherits="_Default" %>

<!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">
    <title>CompareValidator Control Example</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
```

```
        <asp:Label ID="Label1" runat="server" Font-Bold="True" Font-Size="Medium"
             Font-Underline="True" Text="CompareValidator Control Example"></asp:Label>
             <br />
             <br />
             <br />
             <asp:Label ID="Label2" runat="server" Font-Bold="True" Font-Size="Small"
             Text="User Name:"></asp:Label>
                       
             <asp:TextBox ID="TextBox1" runat="server" Width="173px"></asp:TextBox>
             <asp:RequiredFieldValidator ID="RequiredFieldValidator1" runat="server"
             ControlToValidate="TextBox1" ErrorMessage="This Field can not be
                     blank."></asp:RequiredFieldValidator>
             <br />
             <br />
             <asp:Label ID="Label3" runat="server" Font-Bold="True" Font-Size="Small"
             Text="Password:"></asp:Label>
                         
             <asp:TextBox ID="TextBox2" runat="server" TextMode="Password"
                     Width="173px"></asp:TextBox>
             <asp:RequiredFieldValidator ID="RequiredFieldValidator2" runat="server"
             ControlToValidate="TextBox2" ErrorMessage="This Field can not be
                     blank."></asp:RequiredFieldValidator>
             <br />
             <br />
             <asp:Label ID="Label4" runat="server" Font-Bold="True" Font-Size="Small"
             Text="Re-type Password:"></asp:Label>
              <asp:TextBox ID="TextBox3" runat="server" TextMode="Password"
             Width="173px"></asp:TextBox>
             <asp:CompareValidator ID="CompareValidator1" runat="server"
             ControlToCompare="TextBox2" ControlToValidate="TextBox3"
             ErrorMessage="Password does not match."></asp:CompareValidator>
             <br />
             <br />
             <asp:RequiredFieldValidator ID="RequiredFieldValidator3" runat="server"
             ControlToValidate="TextBox3" ErrorMessage="This Field can not be
                     blank."></asp:RequiredFieldValidator>
             <br />
             <br />
                        
                         
                       
             <asp:Button ID="Button1" runat="server" BackColor="Black" Font-Bold="True"
             Font-Size="Small" ForeColor="White" Text="Submit" />

        </div>
        </form>

    </body>
    </html>
```

Now, run the application, and enter avantika in the first `TextBox` control, password@12 in the second `TextBox` control, password in the third `TextBox` control, and then click the Submit button. The `CompareValidator` control displays an error message, as shown in Figure 6.4:

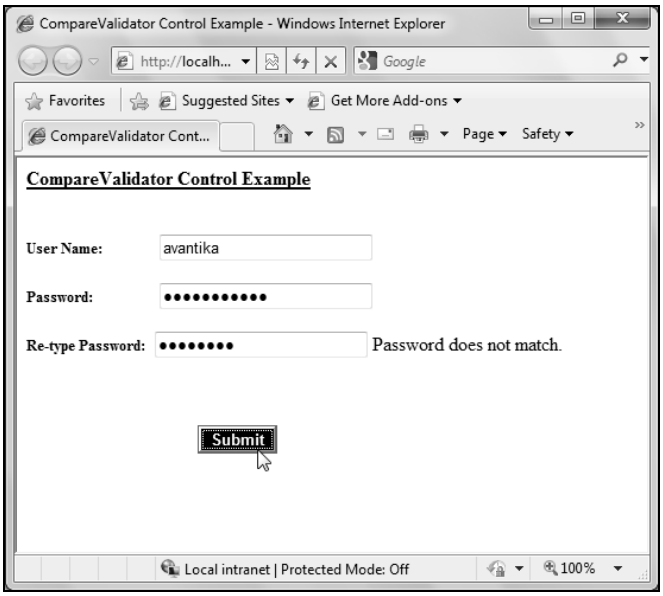

Figure 6.4: Displaying Error Message in the CompareValidator Control

Using the CustomValidator Control

When you add a CustomValidator control on a Web page (Default.aspx), it adds the following code to the source code of the Web page:

```
<asp:CustomValidator ID="CustomValidator1" runat="server"
 ErrorMessage="CustomValidator"></asp:CustomValidator>
```

Now, let's create an application named CustomValidatorCS (also available on the CD-ROM). In this example, we are taking a `CustomValidator` control on a Web page, which validates the `TextBox` control and ensures that the user enters the correct data (username). Listing 6.5 shows the Default.aspx page for the CustomValidator control:

Listing 6.5: Showing the Code for the `Default.aspx` Page

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"
 Inherits="_Default" %>
<!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">
    <title>CustomValidator Control Example</title>
</head>
<body>
    <form id="form1" runat="server">
    <script language="vbscript" type="text/vbscript">
         Sub Validate (source, arguments)
                 If (arguments.Value = "testuser")Then
                         arguments.IsValid = True
                 Else
                         arguments.IsValid = False
                 End IF
         End Sub
         </script>
    <div>
    <asp:Label ID="Label1" runat="server" Font-Bold="True" Font-Size="Medium"
```

```
            Font-Underline="True" Text="CustomValidator Control Example"></asp:Label>
            <br />
            <br />
            <br />
            <asp:Label ID="Label2" runat="server" Font-Bold="True" Font-Size="Small"
            Text="Enter User Name:"></asp:Label>
              
            <asp:TextBox ID="TextBox1" runat="server" Width="179px"></asp:TextBox>
            <asp:RequiredFieldValidator ID="RequiredFieldValidator1" runat="server"
            ControlToValidate="TextBox1" ErrorMessage="This field can not be
            blank."></asp:RequiredFieldValidator>
            <br />
            <asp:CustomValidator ID="CustomValidator1" runat="server"
            ErrorMessage="Invalid entry. Correct username is 'testuser'."
            ClientValidationFunction="Validate"
                    ControlToValidate="TextBox1"></asp:CustomValidator>
            <br />
            <br />
            <br />
                        
                                
            <asp:Button ID="Button1" runat="server" BackColor="Black" Font-Bold="True"
            Font-Size="Small" ForeColor="White" Text="Submit" />
        </div>
        </form>
    </body>
    </html>
```

In the source file, `Default.aspx`, we have created a procedure named Validate in the `<script>` tag. The string that the user enters should be same to the string defined, testuser, in the script. Now, run the application and enter avantika in the Enter user name text box and click the Submit button. The `CustomValidator` control displays an error message, as shown in Figure 6.5:

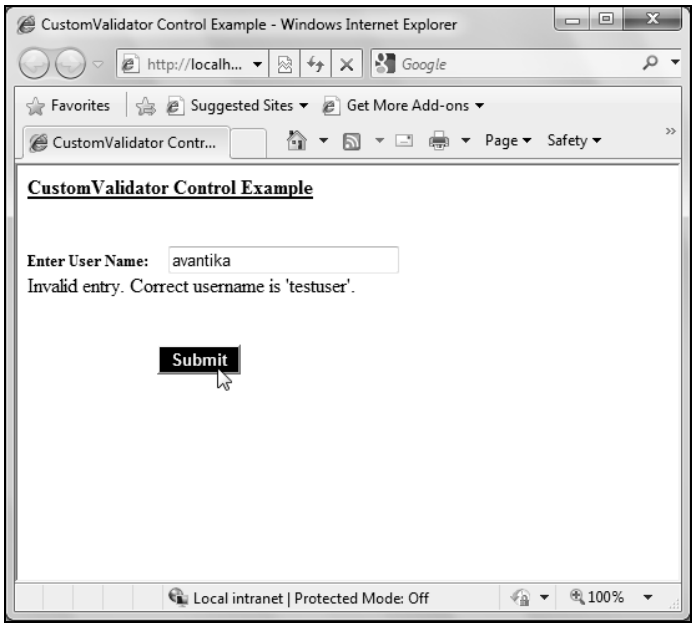

Figure 6.5: Displaying Error Message in the CustomValidator Control

Using the ValidationSummary Control

When you add a ValidationSummary control to a Web page (Default.aspx), it adds the following code to the source code of the Web page:

```
<asp:ValidationSummary ID="ValidationSummary1" runat="server" />
```

Now, let's create an application named ValidationSummaryCS (also available on the CD-ROM). In this example, we are taking two `RequiredFieldValidator` controls and one `ValidationSummary` control on a Web page. The `ValidationSummary` control displays a summary of the error messages from all the other Validation controls. Listing 6.6 shows the Default.aspx page for the ValidationSummary control:

Listing 6.6: Showing the Code for the `Default.aspx` Page

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"
 Inherits="_Default" %>

<!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">
    <title>ValidationSummary Control Example</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
    <asp:Label ID="Label1" runat="server" Font-Bold="True" Font-Size="Medium"
         Font-Underline="True" Text="ValidationSummary Control Example"></asp:Label>
         <br />
         <br />
         <asp:Label ID="Label2" runat="server" Text="First Name:  "></asp:Label>
         <asp:TextBox ID="TextBox1" runat="server" Width="174px"></asp:TextBox>
         <asp:RequiredFieldValidator ID="RequiredFieldValidator1" runat="server"
         ControlToValidate="TextBox1" Display="None"
         ErrorMessage="First Name field can not be blank."></asp:RequiredFieldValidator>
         <br />
         <br />
         <asp:Label ID="Label3" runat="server" Text="Last Name:  "></asp:Label>
         <asp:TextBox ID="TextBox2" runat="server" Width="174px"></asp:TextBox>
         <asp:RequiredFieldValidator ID="RequiredFieldValidator2" runat="server"
         ControlToValidate="TextBox2" Display="None"
         ErrorMessage="Last Name field can not be blank."></asp:RequiredFieldValidator>
         <br />
         <br />
         <asp:ValidationSummary ID="ValidationSummary1" runat="server" Height="65px"
         Width="439px" />
         <br />
         <asp:Button ID="Button1" runat="server" BackColor="Black" Font-Bold="True"
         Font-Size="Small" ForeColor="White" Text="Submit" />
    </div>
    </form>
</body>
</html>
```

Now, run the application and click the Submit button without entering any value in both the `TextBox` controls. Notice that errors related to both the `RequiredFieldValidator` controls in the `ValidationSummary` control appears, as shown in Figure 6.6:

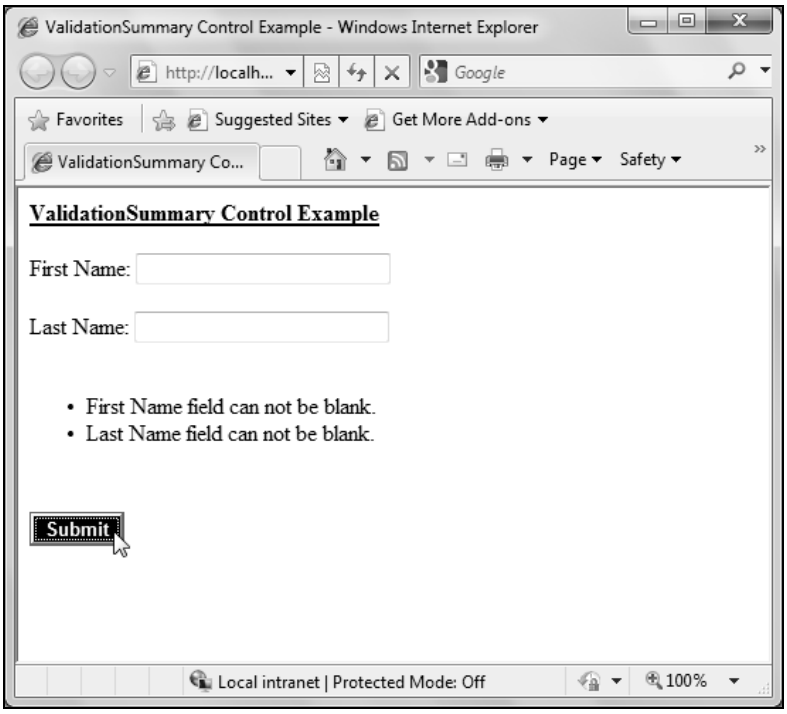

Figure 6.6: Showing Error Messages in the ValidationSummary Control

Summary

In this chapter, we have learned about the Validation controls, for example, `RequiredFieldValidator`, `CustomValidator`, and the rest. These controls are used to validate data entered by the user at the client-end. These Validation controls, in turn, helps in reducing the load on the server by eliminating the need to send every piece of data to the server for validation. The associated control property for each control needs to be set for its association with standard controls. At the time of designing a website, all the Validation controls require an expression property. This property helps validate the data entered by a user in the associated control.

In the next chapter, you learn about developing the ASP.NET AJAX Applications.