Tutorial 08 – Server Control Form Validation

There are validation controls for specific types of validation, such as range checking or pattern matching. The following table lists the validation controls.

Type of validation	**Control to use**	**Description**
Required entry	RequiredFieldValidator	Ensures that the user does not skip an entry.
Comparison to a value	CompareValidator	Compares a user's entry against a constant value, against the value of another control (using a comparison operator such as less than, equal, or greater than), or for a specific data type.
Range checking	RangeValidator	Checks that a user's entry is between specified lower and upper boundaries. You can check ranges within pairs of numbers, alphabetic characters, and dates.
Pattern matching	RegularExpressionValidator	Checks that the entry matches a pattern defined by a regular expression. This type of validation enables you to check for predictable sequences of characters, such as those in e-mail addresses, telephone numbers, postal codes, and so on.
User-defined	CustomValidator	Checks the user's entry using validation logic that you write yourself. This type of validation enables you to check for values derived at run time.
Summary	ValidationSummary	This control is used along with other validation controls to display a summary message that lists all errors discovered on the page.

You can attach more than one validation control to an input control. For example, you might specify that a control is required and that it also contains a specific range of values.

Using the RequiredFieldValidator Control

The most basic type of ASP.NET validation control is the required field validator. As its name implies, it requires the user to enter some value — any value will do — into an input field. RequiredFieldValidator controls are used most often with TextBox controls, but you can use them with other types of input controls as well.

Besides the ubiquitous ID and Runat attributes, a RequiredFieldValidator control has two important attributes you should always set:

- § ControlToValidate: Provides the ID of the input control that the required field validator should be associated with.
- § ErrorMessage: Provides the text that the validator control will display if the user doesn't enter any data into the associated input control.

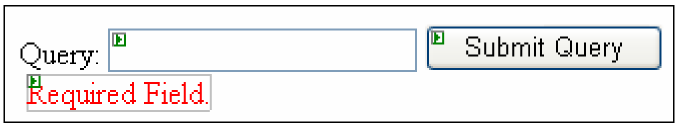
For example, here's a typical RequiredFieldValidator control:

```
<asp:RequiredFieldValidator
      ID="RequiredFieldValidator1"
      runat="server"
      ControlToValidate="TextBox1"
      ErrorMessage="Required Field.">
</asp:RequiredFieldValidator>
```

Here the required field validator is associated with the control named TextBox1 and the error message is "Required field.".

You should place the RequiredFieldValidator control where you want the error message to be displayed if the user fails to enter a value.

Example



1. Add a TextBox control to the page.
2. Add a Button control to the page and set the following properties:

```
      <asp:Button ID="Button1" runat="server" Text="Submit Query" />
```

3. Add a RequiredFieldValidator control to the page and set the following properties:

```
      <asp:RequiredFieldValidator ID="RequiredFieldValidator1"
            runat="server"
            ControlToValidate="TextBox1"
            ErrorMessage="Required Field.">
      </asp:RequiredFieldValidator>
```

4. Click the Start Debugging icon or press F5 to debug the codes and preview the web form in browser.

Using the CompareValidator Control

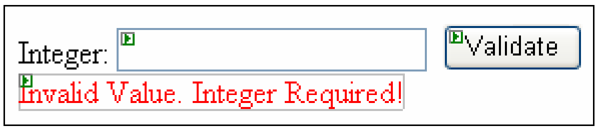
Next to the RequiredFieldValidator control, the CompareValidator control is probably the validator you'll use most. It performs several basic types of comparison checks on an input field. It can perform the following basic types of comparisons:

- **Constant comparisons**, in which the value entered by the user is compared with a constant value. For example, you can ensure that a numeric value is greater than zero.
- **Comparisons with other controls**, in which the value entered by the user is compared with the value entered for another control. For example, if a page has fields that let the user enter a start date and an end date, you can use a CompareValidator control to make sure that the end date does not occur before the start date.
- **Type comparisons**, which let you verify that the user has entered the correct type of data. This is one of the most useful types of comparisons performed by the CompareValidator control.

The following table lists the properties you're most likely to set for the CompareValidator control:

Property	**Description**
ControlToValidate	The error message displayed if the control fails the validation.
ValueToCompare	The value that the control should be compared to.
ControlToCompare	The ID of another control that will supply the value to compare.
Operator	The comparison operation to perform. You can specify Equal, NotEqual, LessThan, LessThanEqual, GreaterThan, GreaterThan, Equal, or DataTypeCheck.
Type	The data type. You can specify String, Integer, Double, Date, or Currency.

Example 1



1. Add a TextBox control to the page.
2. Add a Button control to the page and set the following properties:

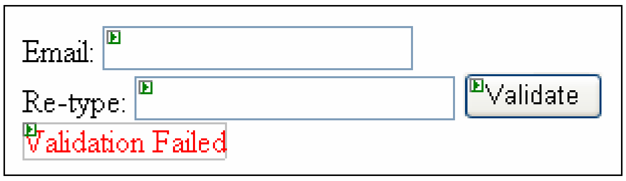
```
<asp:Button ID="Button1" runat="server" Text="Validate" />
```

3. Add a CompareValidator control to the page and set the following properties:

```
<asp:CompareValidator ID="CompareValidator1" runat="server"
      ControlToValidate="TextBox1"
      ErrorMessage="Invalid Value. Integer Required!"
      Operator="DataTypeCheck"
      Type="Integer">
</asp:CompareValidator>
```

4. Click the Start Debugging icon or press F5 to debug the codes and preview the web form in browser.

Example 2


Email:

Re-type:

Validation Failed

1. Add two TextBox controls to the page.
2. Add a Button control to the page and set the following properties:

```
<asp:Button ID="Button1" runat="server" Text="Validate" />
```

3. Add a CompareValidator control to the page and set the following properties:

```
<asp:CompareValidator ID="CompareValidator1" runat="server"
      ControlToCompare="TextBox2"
      ControlToValidate="TextBox1"
      ErrorMessage="Validation Failed"
      EnableClientScript="False">
</asp:CompareValidator>
```

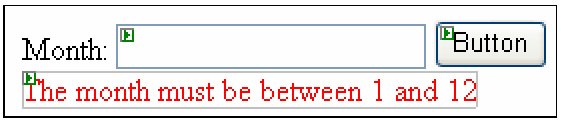
4. Click the Start Debugging icon or press F5 to debug the codes and preview the web form in browser.

Using the RangeValidator Control

The RangeValidator control is similar to the CompareValidator control, but instead of doing a single comparison check, it does two checks to make sure the value entered by the user falls within a particular range of values. Here's a table that sums up the properties you can use with the RangeValidator control:

Property	**Description**
ControlToValidate	The ID of the input control to validate.
ErrorMessage	The error message displayed if the control fails the validation.
MinimumValue	The smallest acceptable value.
MaximumValue	The largest acceptable value.
Type	The data type. You can specify String, Integer, Double, Date, or Currency.

Example



1. Add a TextBox control to the page.
2. Add a Button control to the page and set the following properties:

```
<asp:Button ID="Button1" runat="server" Text="Validate" />
```

3. Add a RangeValidator control to the page and set the following properties:

```
<asp:RangeValidator ID="RangeValidator1" runat="server"
      ControlToValidate="TextBox1"
      ErrorMessage="The month must be between 1 and 12"
      MaximumValue="12"
      MinimumValue="1"
      Type="Integer">
</asp:RangeValidator>
```

4. Click the Start Debugging icon or press F5 to debug the codes and preview the web form in browser.

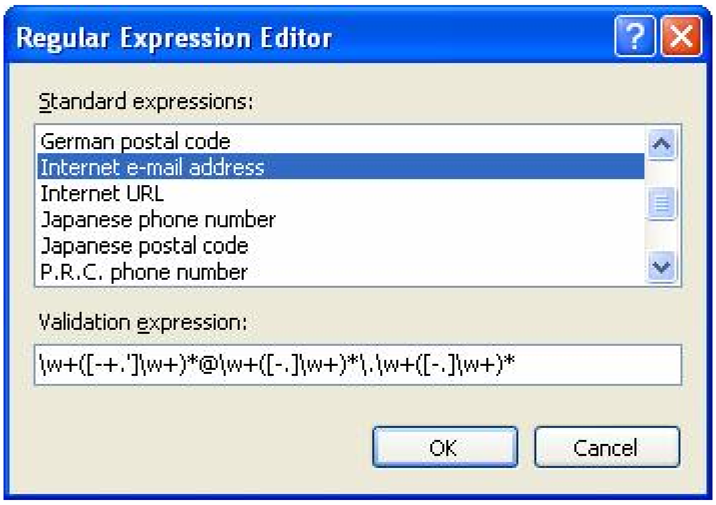
Using the RegularExpressionValidator

Many types of data-entry fields follow standard patterns. For example, phone numbers follow the pattern (nnn) nnn-nnnn, and Zip codes are either nnnnn or nnnnn-nnnn.

The RegularExpressionValidator control is designed to let you validate data against patterns like this. The term regular expression refers to a somewhat standardized language used to define these patterns. Regular expressions can be extremely powerful, and also extremely confusing.

You use the ValidationExpression property to specify the regular expression you want to use for the validation.

Fortunately, Visual Studio provides several predefined regular expressions you can use with the RegularExpressionValidator control. There are pre-defined expressions. To use one of these expressions, select a RegularExpressionValidator control in the Web designer, and then Select the ValidationExpression property in the Properties window. Doing so brings up the dialog box shown in Figure below, from which you can choose the expression you want to use.



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

Example

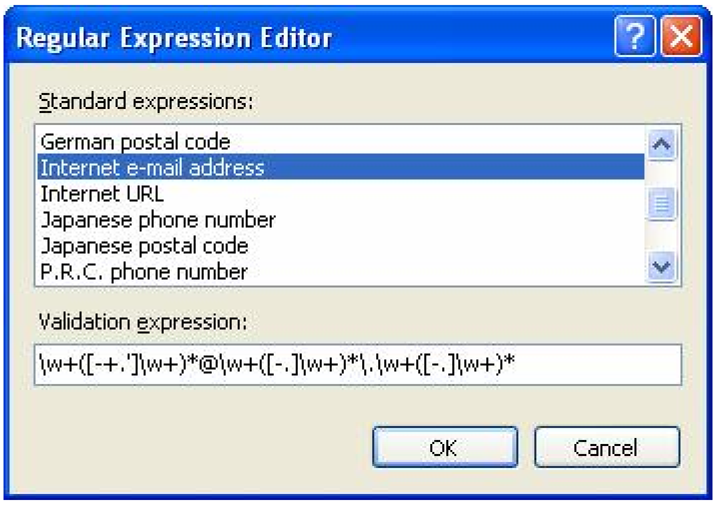
1. Add a TextBox control to the page.
2. Add a Button control to the page and set the following properties:

```
        <asp:Button ID="Button1" runat="server" Text="Validate" />
```

3. Add a RegularExpressionValidator control to the page and set the following properties,

```
<asp:RegularExpressionValidator ID="RegularExpressionValidator1"
      runat="server" ErrorMessage="RegularExpressionValidator"
      ControlToValidate="TextBox1"
      ValidationExpression="\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*">
</asp:RegularExpressionValidator>
```

You may select the ValidationExpression property in the Properties window. to brings up the dialog box shown in Figure below, from there you can choose the Internet e-mail address.



Standard expressions:

- German postal code
- Internet e-mail address
- Internet URL
- Japanese phone number
- Japanese postal code
- P.R.C. phone number

Validation expression:

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

4. Click the Start Debugging icon or press F5 to debug the codes and preview the web form in browser.

Using a CustomValidator Control

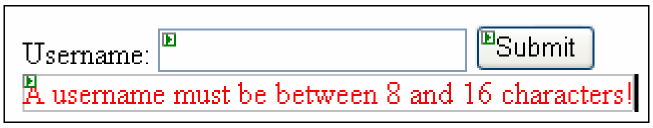
If ASP.NET doesn't provide a validator control that offers the type of validation you need, you can tool up your own by using a CustomValidator control. The .aspx markup for a CustomValidator control is similar to the markup for other validator controls. For example:

```
<asp:CustomValidator ID="CustomValidator1" runat="server"
      ControlToValidate="TextBox1"
      ErrorMessage="A username must be between 8 and 16 characters!"
      OnServerValidate="CustomValidator1_ServerValidate">
</asp:CustomValidator>
```

Notice that the CustomValidator control includes an OnServerValidate attribute that specifies the name of the method to execute when the custom validator is called upon to validate the data. This method is passed a parameter named args, which contains two properties you can use to perform your own validation routine:

- **IsValid**: Your routine should set this property to indicate whether the value entered by the user is valid.
- **Value**: This is the value entered by the user.

Example


Username:
Submit
A username must be between 8 and 16 characters!

1. Add a TextBox control to the page.
2. Add a Button control to the page and set the following properties:

```
      <asp:Button ID="Button1" runat="server" Text="Submit" />
```

3. Add a CustomValidator control to the page and set the following properties:

```
<asp:CustomValidator ID="CustomValidator1" runat="server"
      ControlToValidate="TextBox1"
      ErrorMessage="A username must be between 8 and 16 characters!"
      OnServerValidate="CustomValidator1_ServerValidate">
</asp:CustomValidator>
```

4. Double click the CustomValidator and type the following code.

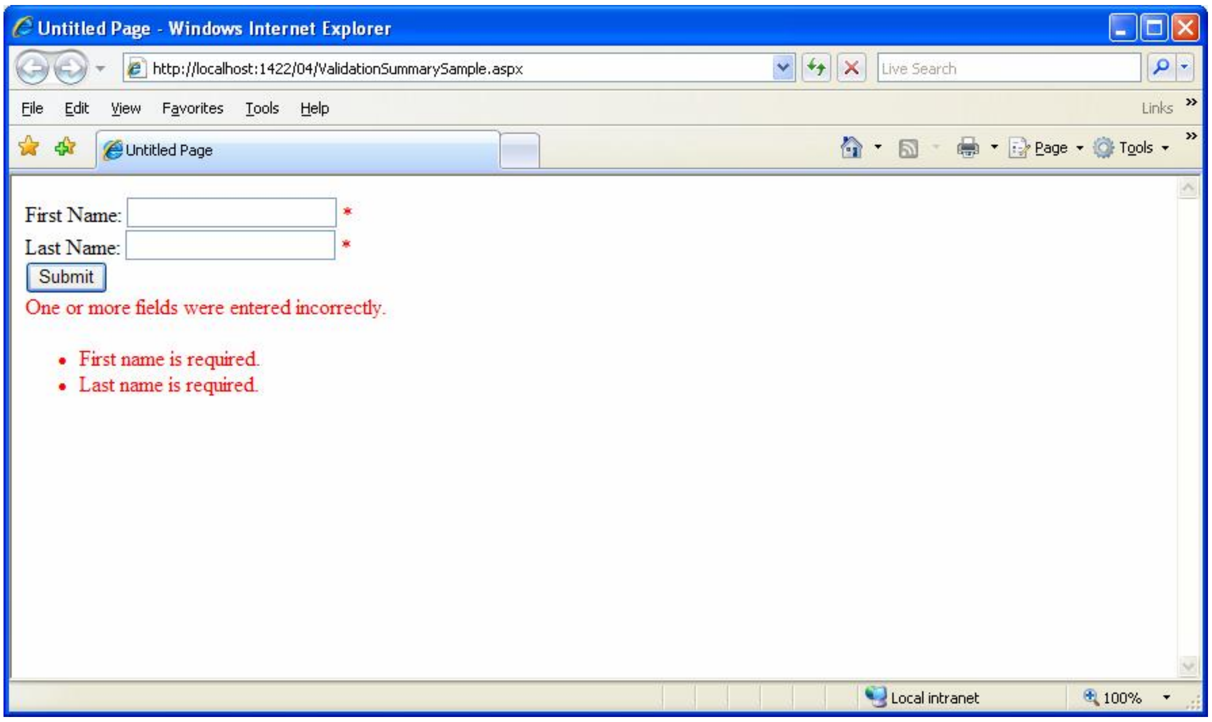
```
protected void CustomValidator1_ServerValidate(object source,
ServerValidateEventArgs args)
{
      if (args.Value.Length < 8 || args.Value.Length > 16)
            args.IsValid = false;
      else
            args.IsValid = true;
}
```

5. Click the Start Debugging icon or press F5 to debug the codes and preview the web form in browser.

Using the ValidationSummary Control

The ValidationSummary control is designed to let you display validation error messages at the top or bottom of the page, rather than intermixed with the input controls. The ValidationSummary control can also display a simple message to indicate that at least one validation error has occurred on the page. Or, if you prefer, it can show the individual error messages for each validation error as a list, a bullet list, or a single paragraph. It can even display a pop-up message box that lists the errors.

When you use a ValidationSummary control on a page, you usually want the individual validation error messages to appear in the validation summary at the top or bottom of the page, rather than intermixed with the input controls. It's common practice simply to highlight each control that has a validation error with an asterisk, as shown in figure below.


Untitled Page - Windows Internet Explorer
http://localhost:1422/04/ValidationSummarySample.aspx
Live Search
File
Edit
View
Favorites
Tools
Help
Links
Untitled Page
Page
Tools
Local intranet
100%

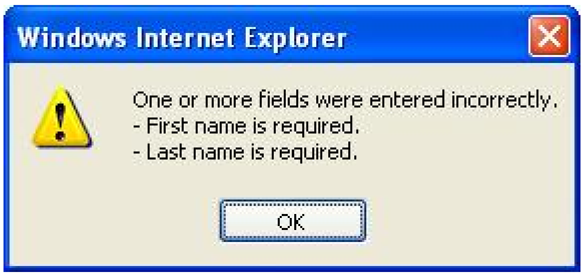
First Name: *
Last Name: *
Submit
One or more fields were entered incorrectly.

- First name is required.
- Last name is required.

The following table lists a few additional properties you can specify for the ValidationSummary control:

Property	**Description**
ShowSummary	Specifies whether the error messages for each invalid validator should be displayed. The default is True.
DisplayMode	Specifies how the error messages should be displayed. The options are List, BulletList, or SingleParagraph.
HeaderText	Specifies a text message to be displayed above the summary.
ShowMessageBox	Specifies whether a pop-up message box should be displayed.

If you specify ShowMessageBox="True", a pop-up message box will be displayed if there is a validation error, as shown in figure below. The user must close this dialog box before correcting the entry error.



One or more fields were entered incorrectly.
- First name is required.
- Last name is required.

Example

1. Add two TextBox controls to the page.

2. Add a Button control to the page and set the following properties:

```
      <asp:Button ID="Button1" runat="server" Text="Submit" />
```

3. Add two RequiredFieldValidator controls to the page and set the following properties:

```
<asp:RequiredFieldValidator ID="RequiredFieldValidator1" runat="server"
      ControlToValidate="TextBox1"
      ErrorMessage="First name is required.">
      *
</asp:RequiredFieldValidator>
```

```
<asp:RequiredFieldValidator ID="RequiredFieldValidator2" runat="server"
      ControlToValidate="TextBox2"
      ErrorMessage="Last name is required.">
      *
</asp:RequiredFieldValidator>
```

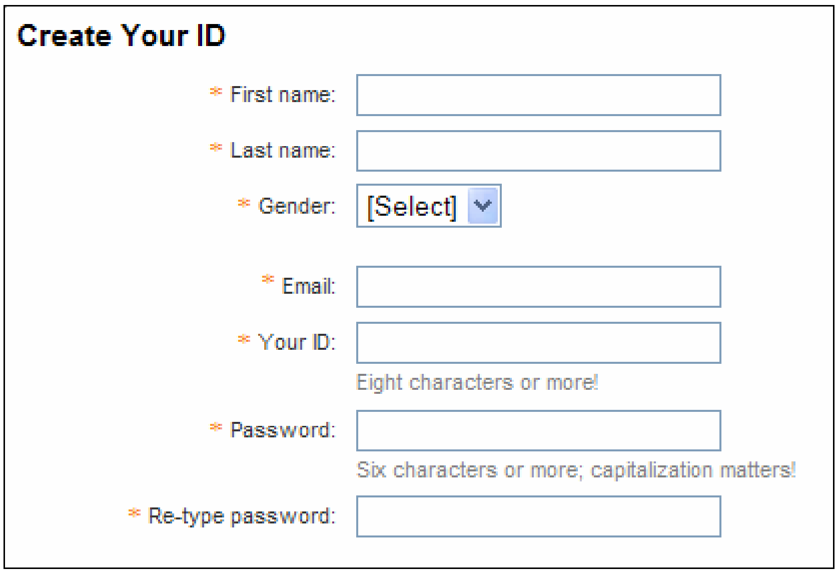
4. Add a ValidatorSummary control to the page and set the following properties:

```
<asp:ValidationSummary ID="ValidationSummary1" runat="server"
HeaderText="One or more fields were entered incorrectly." />
```

5. Click the Start Debugging icon or press F5 to debug the codes and preview the web form in browser.

Exercise

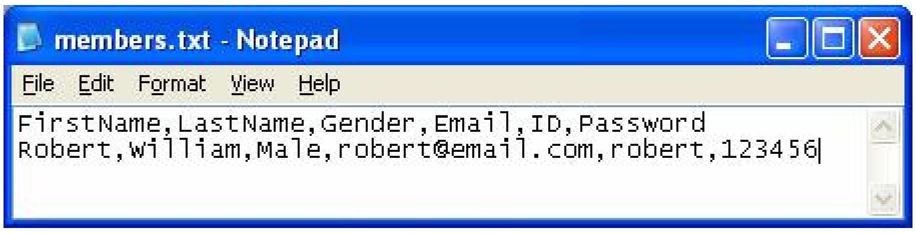
1. Create a member registration form as shown as figure below and use appropriate validation controls to validate the web controls.



Create Your ID

* First name:	
* Last name:	
* Gender:	[Select]
* Email:	
* Your ID:	Eight characters or more!
* Password:	Six characters or more; capitalization matters!
* Re-type password:	

2. Add a button to the Web Form and assign an action to write the inputs to a text file named "`members.txt`" as comma separated values.



```
FirstName,LastName,Gender,Email,ID,Password
Robert,william,Male,robert@email.com,robert,123456
```

3. Create a Web Form to read the "`members.txt`" and display the values in organized format on the Web Form.