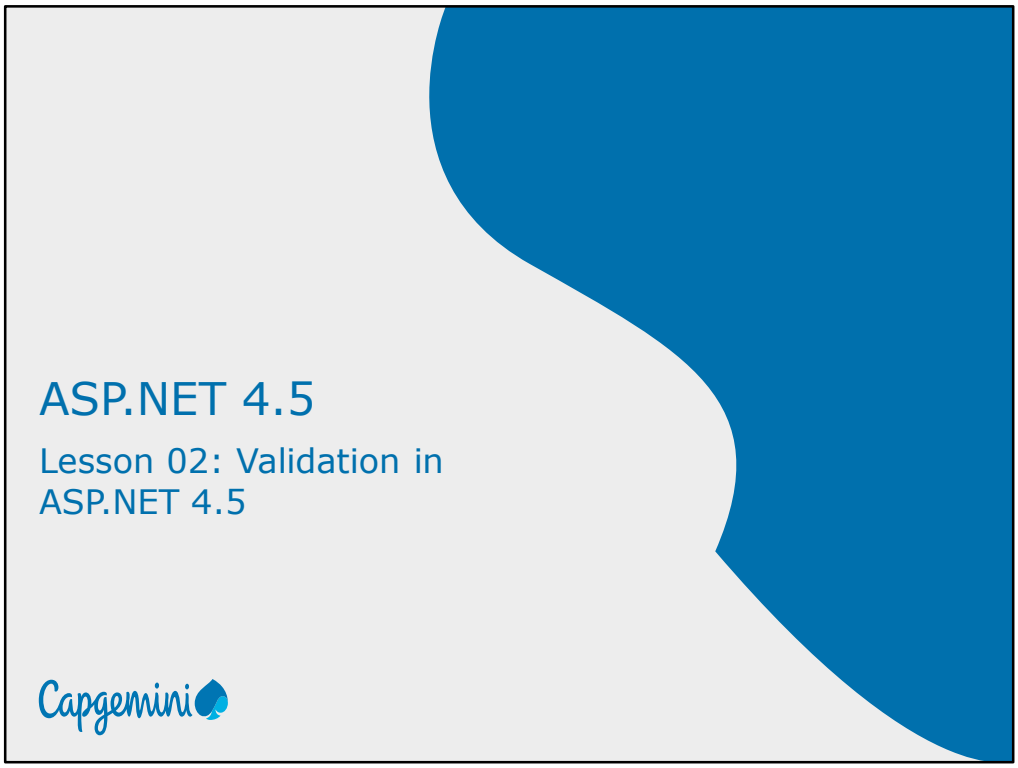

ASP.NET 4.5
Lesson 02: Validation in ASP.NET 4.5
Capgemini

Lesson Objectives

- After completing this Lesson you will understand:
  - Validation controls provided in ASP.NET
  - Validation Group in ASP.NET

Copyright © Capgemini 2018. All Rights Reserved

2.1 Validation

Overview

- Validation:
  - Set of rules you apply to the data you collect.
- Following validations are possible in Web applications:
  - Client-side
  - Server-side

Copyright © Capgemini 2018. All Rights Reserved

Validation is a set of rules that you apply to the data you collect. These rules can be many or few depending on the application requirements. The data you collect for validation comes in Web forms in your application. Validation is web applications can be done at the client-side or server-side.

Client Side Validation: Quick and responsive for the end user. Bu using the client side validation the user can be prompted with messages without having to visit the server to detect the possible problems with values. But client side validation is insecure.

Server Side Validation: More secure as compared to client side validation because they cannot be bypassed. But it is slow since data has to be sent to the server for validation.

In this module we would be looking at the validation mechanism in ASP.Net.

2.2: Validation Controls

Overview

- Server controls for validation
- To ensure the correctness of data being entered
- Eliminates developers' dilemma of where validation can be implemented
- Performs browser detection when generating the page and make a decision based on information it gathers

Copyright © Capgemini 2018. All Rights Reserved

ASP.Net validation controls (also known as validators) simplify the task of ensuring that data is entered correctly on forms. They are simply server controls introduced in ASP.Net and they are rather smart. You can associate the validation controls with any of the form controls included in the ASP.NET Framework. In classic ASP developers often faced a dilemma of where the validation code should exist. The validation controls in ASP.Net eliminates this dilemma because it performs browser detection when generating the ASP.Net and makes decision based on the information it gathers.

If the user has a browser that supports DHTML and client-side validation is enabled, validation controls can perform their validation using client script. Client-side validation is handy whenever you want to avoid a round trip to the server for server-side validation. Regardless of whether client-side validation is performed, server-side validation will be done only to ensure that validation takes place, even when the user's browser does not support DHTML. ASP.Net provides six validation controls:
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.

2.2: Validation Controls

Validation Controls

- The validation controls are found in the System.Web.UI.WebControls namespace and inherit from the
- BaseValidator class.
- This class defines the basic functionality for a validation control.

Copyright © Capgemini 2018. All Rights Reserved

ControlToValidate Identifies the control that this validator will check. Each validator can verify the value in one input control. However, it's perfectly reasonable to "stack" validators—in other words, attach several validators to one input control to perform more than one type of error checking.
ErrorMessage and ForeColor
If validation fails, the validator control can display a text message (set by the ErrorMessage property). By changing the ForeColor, you can make this message stand out in angry red lettering.
Display Allows you to configure whether this error message will be inserted into the page dynamically when it's needed (Dynamic) or whether an appropriate space will be reserved for the message (Static).
Dynamic is useful when you're placing several validators next to each other. That way, the space will expand to fit the currently active error indicators, and you won't be left with any unseemly whitespace.
Static is useful when the validator is in a table and you don't want the width of the cell to collapse when no message is displayed.
Finally, you can also choose None to hide the error message altogether.
IsValid After validation is performed, this returns true or false depending on whether it succeeded or failed. Generally, you'll check the state of the entire page by looking at its IsValid property instead to find out if all the validation controls succeeded.
Enabled When set to false, automatic validation will not be performed for this control when the page is submitted.
EnableClientScript If set to true, ASP.NET will add JavaScript and DHTML code to allow client-side validation on browsers that support it.

2.2.1: RequiredFieldValidator

RequiredFieldValidator

- Makes a user to enter value into a form field mandatorily before submitting the form.
- Example:

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

Copyright © Capgemini 2018. All Rights Reserved

The RequiredFieldValidator control simply checks if something was entered into the HTML form. It is a simple control but is the most frequently used. The validation control has to be simply placed on your form and specify the properties.

ControlToValidate: This property is used in specify which control on the ASP.Net form has to be validated by this control.
Text: Error indicator like #,* etc.
ErrorMessage: The error message to be displayed if the validation fails.

2.2.2: RangeValidator

Range Validator

- Checks whether a form field value falls between a certain minimum and maximum value

Example:

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

Copyright © Capgemini 2018. All Rights Reserved

The RangeValidator control ensures that end-user has entered values within the specified range i.e they fall within a minimum and maximum values.

This control can be used for checking range of numbers, characters and also items such as calendar dates. The properties for this control are:
ControlToValidate: The ASP.Net form element being validated.
Text – The error indicator like #,* etc.
ErrorMessage: The error message to be displayed if the 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.

2.2.3: CompareValidator

Compare Validator

- Allows you to compare form elements
- Compares form values with constants

Example:

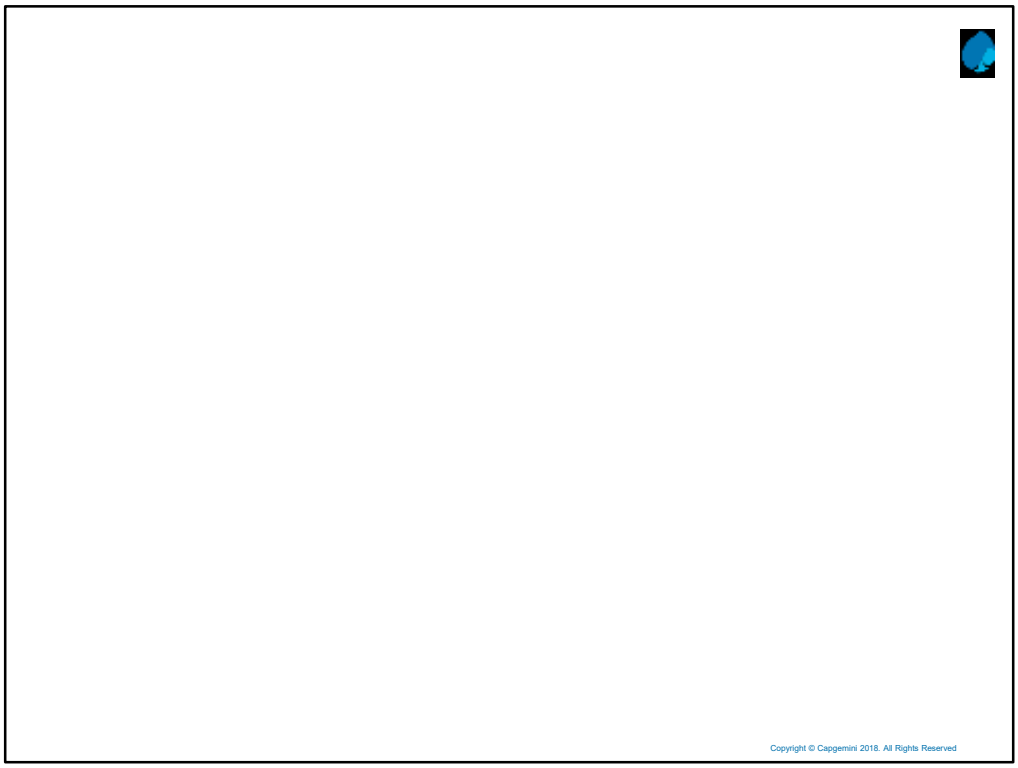
```
<asp:TextBox id="txtBirthDate" Runat="server" />
<asp:CompareValidator id="cmpBirthDate"
Text="(Invalid Date)" ControlToValidate="txtBirthDate"
Type="Date" Operator="DataTypeCheck" Runat="server"
/>
```

Copyright © Capgemini 2018. All Rights Reserved

The CompareValidator control enables you to perform 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.



Important properties for this control are:
ControlToValidate: The form element being validated.
Text: The error indicator like #,* etc.
ErrorMessage: Error message to be displayed if the 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.

2.2.4: RegularExpressionValidator

Regular Expression Validator

- Control offers flexibility to check user input against a specified pattern defined in your regular expression

Example:

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

Copyright © Capgemini 2018. All Rights Reserved

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.

The properties for this control are:
ControlToValidate: Form element that is to be validated.
Text: The error indicator like #,* etc.
ErrorMessage: The error message to be displayed if the validation fails.
ValidationExpression: The regular expression pattern to compare with the input value.

2.3: Demo on Validation Controls

Validation Controls

- Understanding Validation Controls

Copyright © Capgemini 2018. All Rights Reserved

2.4: CustomValidator

Overview

- Allows you to build your own client side or server side validation to apply to your web form

Example:

```
<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" /> <br /><br />
```

Copyright © Capgemini 2018. All Rights Reserved

We have seen a wide variety of Validation controls provided by ASP.NET. These controls address many of the validation rules that the developers would want to apply to your web forms. But in some situations these controls may not work and some customization may be required. This is where you would used CustomValidator.
This validator control allows you to build your own client side or server side validations that can be easily applied to your webforms.

The CustomValidator control has three important properties:
ControlToValidate: The form field being validated.
Text: The error indicator like #,* etc..
ErrorMessage: The error message to be displayed if the 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. The function can be a typical JavaScript function for the client-side custom validator.

2.5: Demo on CustomValidator

Custom Validator

- Understanding CustomValidator in ASP.NET

Copyright © Capgemini 2018. All Rights Reserved

2.6: Validation Summary

Overview

- Reporting control used by other validation controls on the page
- Consolidates error reporting for all validation errors occurred on the page
- Error messages can be displayed in a list, bulleted list or paragraph

Copyright © Capgemini 2018. All Rights Reserved

The ValidationSummary control does not perform validations on the content input into your webforms. This control is a reporting control which is used to consolidate errors for all the validation errors occurred on your page instead of leaving this up to each and every individual validation control.

This facility is very handy when the form is quite comprehensive. It becomes quite user-friendly to present all possible validation errors together in a single and identifiable manner. These error messages can be displayed in a list, bulleted list or a paragraph.

You might have noticed that each of the validation controls includes an ErrorMessage property. We do not use 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!").

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.

2.7: Demo

Validation Summary Control

- Understanding ValidationSummary control

Copyright © Capgemini 2018. All Rights Reserved

2.7.1 Common Properties

Common Properties

Validator	Properties
RequiredFieldValidator	None ,required
RangeValidator	MaximumValue, MinimumValue, Type
CompareValidator	ControlToCompare, Operator, Type, ValueToCompare
RegularExpressionValidator	ValidationExpression
CustomValidator	ClientValidationFunction, ValidateEmptyText, ServerValidate event

Copyright © Capgemini 2018. All Rights Reserved

2.8: Validation Group

Overview

- Many forms can exist on a web page and have different validations for controls on each form
- Different validation controls can be applied to each control
  - However, on from submission all validations, unrelated to the submitted form, are fired
- ASP.Net2.0 provides with Validation Group that enables you to separate the validation controls into separate groups

Copyright © Capgemini 2018. All Rights Reserved

There could be lot of instances wherein many forms are placed on a single web page. This is required because different button clicks should do behave differently on the server-side. But handling validation for each form in such a situation is an issue.

In the earlier version of ASP.Net, if you have multiple forms in an .aspx page, submission of that page to the server causes validation of all the controls in the page, including the form which is not required to submit values.

For example, if one form on the page accepts username and password to login into the members section and another form allows new user to sign up as member. Each form has its own set of validation controls. The problem is when a user submits one of the forms. You do not want the validation for username & password form to fire when a new user is trying to sign-up and vice-versa.

ASP.Net 2.0 provides you with a ValidationGroup that enables you to separate the validation controls in separate groups. You can assign a group name to a set of validators to ensure that validation occurs only for controls in the specified group. This enables you to have multiple control groups that are validated separately on the same page.

2.8.1: Validation Group

Overview (contd..)

- A group name can be assigned to a set of validators
  - Ensures that validation occurs only for controls in the specified group
  - Allows you to have multiple control groups validated separately on the same page
- Specify the Validation Group Property for the Button and corresponding Validator Controls

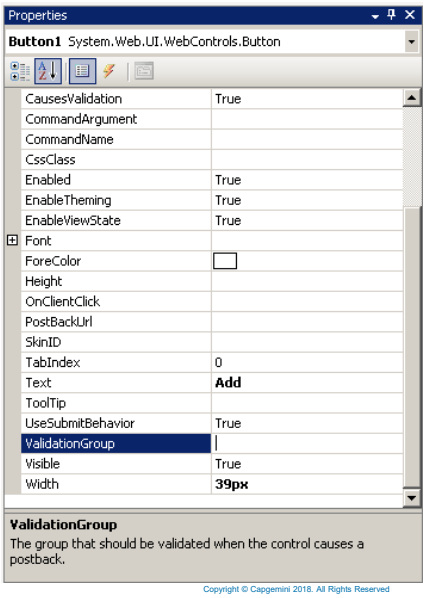
Copyright © Capgemini 2018. All Rights Reserved

The ValidationGroup property is used when the user wants to perform separate validation scenarios on the same page. Set the group name on validator controls and on the button or other postback control that causes validation.

2.8.1:2: Setting the Validation Group property

Set Validation Group Property

- Similarly, set it for Validation controls
- Name of the Validation Group should be same for the button and Validation Control which are part of the same group



Copyright © Capgemini 2018. All Rights Reserved

After the Validation control Property has been set, the source code reflects it as follows:

```
<asp:TextBox ID="SearchTerm" runat="server" />
<asp:Button ID="Button2" runat="server" Text="Search"
PostBackUrl="Search_cs.aspx" ValidationGroup="Search" />
<asp:RequiredFieldValidator ID="RequiredFieldValidator1"
SetFocusOnError="true" Runat="server" ErrorMessage="Search is
empty!!" ControlToValidate="SearchTerm" ValidationGroup="Search" />
```

2.9: Demo:

ValidationGroup

- Understanding ValidationGroup control

Copyright © Capgemini 2018. All Rights Reserved

2.10: Html 5 Validation

Html 5 Validation

- ASP.NET does support one HTML5 validation feature: the enhanced type attribute, which allows you to create text boxes that are intended for specific types of data (such as e-mail addresses or numeric values).
- You can set the type attribute through the TextBox.TextMode property.
  - <asp:TextBox id="txtAge" runat="server" TextMode="Number" />

Copyright © Capgemini 2018. All Rights Reserved

Note:For the vast majority of ASP.NET web forms, the best solution is to use ASP.NET validators.

2.11: Unobtrusive Validation in Asp.Net 4.5

Unobtrusive Validation in Asp.Net 4.5

- You can now configure the built-in validator controls to use unobtrusive JavaScript for client-side validation logic.
- This significantly reduces the amount of JavaScript rendered inline in the page markup and reduces the overall page size.
- You can configure unobtrusive JavaScript for validator controls in any of these ways:
  - In Web.config file
  - In Global.asax
  - In Page class

Copyright © Capgemini 2018. All Rights Reserved

The first method is by using the Web.Config file.
Step 1
Write the following code in your Web.Config file:

```
<configuration>
<appSettings>
<add key="ValidationSettings:UnobtrusiveValidationMode" value="None"></add>
</appSettings>
<system.web>
<compilation debug="true" targetFramework="4.5" />
<httpRuntime targetFramework="4.5" />
</system.web>
</configuration>
```

Step 2
Now write the following code in your WebForm.aspx Page:

```
<asp:TextBox runat="server" ID="txt" />
<asp:RequiredFieldValidator ID="RequiredFieldValidator1" ErrorMessage="txt is
required" ControlToValidate="txt" runat="server" Text="Text is Required"
Display="Dynamic" />
<asp:Button ID="Button1" Text="Send info" runat="server" />
```

Step 3
Run your application

2.11: Demo

Demo

- Open an Asp.Net 4.5 Application have a look at the Javascript Generated when you place a Textbox and Required Field Validator
- Open a Asp.Net 4.5 Application and see the code generated

Copyright © Capgemini 2018. All Rights Reserved

Unobtrusive Validation in Asp.Net 4.5 (Contd..)
The second method is by using a Global.asax file.
Step 1:In the Global.asax file, first add the namespace "using System.Web.UI;".
After adding the namespace write the following code in the Application_Start method:

```
protected void Application_Start(object sender, EventArgs e)
{
ValidationSettings.UnobtrusiveValidationMode = UnobtrusiveValidationMode.None;
}
```

Step 2 :Now write the following code in your WebForm.aspx page:

```
<asp:TextBox runat="server" ID="txt" />
<asp:RequiredFieldValidator ID="RequiredFieldValidator1" ErrorMessage="txt is required"
ControlToValidate="txt" runat="server" Text="Text is Required" Display="Dynamic" />
<asp:Button ID="Button1" Text="Send info" runat="server" /> Step 3
```

Again debugging your Web Application you will again get the same output as you got in the first method

The third method is by simply writing the code on each page inside the Page_Load event.
Step 1

```
protected void Page_Load(object sender, EventArgs e)
{
this.UnobtrusiveValidationMode = System.Web.UI.UnobtrusiveValidationMode.None;
} Step 2 :Now write the following code in your WebForm.aspx page:
<asp:TextBox runat="server" ID="txt" />
<asp:RequiredFieldValidator ID="RequiredFieldValidator1" ErrorMessage="txt is required"
ControlToValidate="txt" runat="server" Text="Text is Required" Display="Dynamic" />
<asp:Button ID="Button1" Text="Send info" runat="server" /> Step 3
```

Again debug your Web Application and you will again get the same output as you got in the first method.

2.12: Validation

Best Practices & Guidelines

- Use Client Side validations to reduce round trips to process user requests
- Code validation for HTML controls with client-side JavaScript

Copyright © Capgemini 2018. All Rights Reserved

The ValidationGroup property is used when the user wants to perform separate validation scenarios on the same page. Set the group name on validator controls and on the button or other postback control that causes validation.

Summary

- In this module, you have learnt:
  - Validation controls provided in ASP.NET
  - Validation Group in ASP.NET

Copyright © Capgemini 2018. All Rights Reserved

Review - Question

- Question 1: Following is not a Validation Control:
  - RequiredFieldValidator
  - RangeValidator
  - CheckboxValidator
  - CompareValidator
- Question 2: Which of the following controls ensures that the user does not skip any entry?
  - Text Box
  - RequiredFieldValidator
  - RangeValidator
  - Button
- Question 3: Server control which checks an input value for a given range is:
  - CompareValidator
  - ChkRangeValue
  - RangeValidator
  - None

Copyright © Capgemini 2018. All Rights Reserved