Agenda

- *BaseValidator*
- *RequiredFieldValidator*
- *CompareValidator*
- *RangeValidator*
- *RegularExpressionValidator*
- *CustomValidator*
- *CausesValidation Property*
- *Grouping – ValidationGroup Property*
- *Paga.Validators and Page.IsValid*

Base Validator

BaseValidator is the parent of all the validation controls.

Validation Controls inherited from **BaseValidator** control are:

1. RequiredFieldValidator
2. RangeValidator
3. CompareValidator
4. RegularExpressionValidator
5. CustomValidator

Properties of BaseValidator:

ControlToValidate, ErrorMessage, Text, Display (Static/Dynamic/None), SetFocusOnError, ToolTip, EnableClientScript (to disable client side validations), ValidationGroup.

ValidationSummary

Is used to display the collection of error message on the page at a given location.

It shows the ErrorMessage property of all the controls whose validation has failed.

Properties: DisplayMode, ShowMessageBox, ShowSummary: - any one of these two properties must be set to true.

Sample:

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

RequiredFieldValidator

To ensure that the user does not skip an entry in the control with which is it associated.

Properties: InitialValue

Required field validator for a textbox

```
<asp:TextBox ID="txtName" runat="server" />
<asp:RequiredFieldValidator ID="rfvName" runat="server" ErrorMessage="Please provide the value for Name"
ControlToValidate="txtFirstName" SetFocusOnError="True" Text="Required" />
```

Demo Textbox value can be anything other than the text - "Demo"

```
<asp:TextBox ID="txtDemo" runat="server" Text="Demo"></asp:TextBox>
<asp:RequiredFieldValidator ID="rfvDemo " runat="server" ErrorMessage="Please provide the value other than
Demo"  ControlToValidate="txtDemo" InitialValue="Demo" SetFocusOnError="True" Text="Required" />
```

User will be forced to select a value other than (Select One)

```
<asp:DropDownList ID="ddl" runat="server">
  <asp:ListItem Value="-1">(Select One)</asp:ListItem>
  <asp:ListItem Value="1">One</asp:ListItem>
  <asp:ListItem Value="2">Two</asp:ListItem>
```

```
  <asp:ListItem Value="3">Three</asp:ListItem>
</asp:DropDownList>
<asp:RequiredFieldValidator ID="rfvDDl" runat="server" ControlToValidate="ddl"
ErrorMessage="Please select a value in ddl"  InitialValue="-1">*</asp:RequiredFieldValidator>
```

CompareValidator

1. Comparing Value of Control with the value of another control - ControlToCompare

2. To compare what a user has entered against a constant value - ValueToCompare

3. Value should be particular type only- DataTypeCheck

Properties: ValueToCompare, ControlToCompare, Type (String, Integer, Double, Date, Currency), Operator (EqualTo, LessThan….DataTypeCheck)

Example:

Confirm Password must be same as password.

```
Password: <asp:TextBox ID="txtPassword" runat="server"></asp:TextBox>
<asp:RequiredFieldValidator ID="rfvPassword" runat="server" ControlToValidate="txtPassword"
  Display="Dynamic" ErrorMessage="Please provide the value for Password"
Text="Required"></asp:RequiredFieldValidator>
Confirm Password <asp:TextBox ID="txtConfirmPassword" runat="server"></asp:TextBox>
<asp:CompareValidator ID="cvConfirmPassword" runat="server" ControlToCompare="txtPassword"
  ControlToValidate="txtConfirmPassword" Display="Dynamic" Operator="Equal" Type="String"
  ErrorMessage="The Password and Confirm Password must be same" Text="Must be same as Password" />
```

Note: With one Server Control we can have more than one validation control associated but the same validation control cannot be associated with multiple server controls.

Amount must be currency (a value with max of two digits after decimal point)

```
Amount: <asp:TextBox ID="txtAmount" runat="server"></asp:TextBox>
<asp:CompareValidator ID="cvAmount" runat="server" ControlToValidate="txtAmount"
  Text="Invalid Amount" Type="Currency" Operator="DataTypeCheck" ErrorMessage="Please provide a valid
Amount"></asp:CompareValidator><br />
```

Note: In the above example we have not set ControlToCompare and ValueToCompare

RangeValidator

Checks that a users entry is between lower and upper boundaries.

Properties: MinimumValue, MaximumValue, Type

Example:Age must be between 18 and 40

```
Age: <asp:TextBox ID="txtAge" runat="server"></asp:TextBox>
```

```
<asp:RangeValidator ID="rvAge" runat="server" ControlToValidate="txtAge" ErrorMessage="Age must be in range of 18 to 40" MaximumValue="40" MinimumValue="18" Type="Integer"> </asp:RangeValidator>
```

RegularExpressionValidator

It Checks the users entry matches a pattern defined by a regularexpression.

Properties: ValidationExpression

Example: Must enter a valid email id.

Example: email id validation

```
Email <asp:TextBox ID="txtEmail" runat="server"></asp:TextBox>
<asp:RegularExpressionValidator ID="revtxtEmail" runat="server" ControlToValidate="txtEmail"
   ErrorMessage="Please provide a valid e-mail id" Text="Invalid e-mail id" ValidationExpression="\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*" />
```

Note: RegEx is the class to work with Regular Expressions in MS.NET.

The RegEx expressions:

? = 0 or 1

* = 0 or More

+ = 1 or More

[ ] = any one of them

{x} = x times

\w = word

\s = space

CustomValidator

Checks the user's entry using validation logic that you write yourself.

Properties: ClientValidationFunction, ValidateEmptyText **Event**: ServerValidate

Code snippet for handling overflow checks

```
<head>
  <script language="javascript">
    function IsOdd(src, args) //src is reference to ValidationControl
    {
      if (isNaN(args.Value) || args.Value % 2 == 0) {
        args.IsValid = false; //The value provided is not valid
        src.errormessage = args.Value + " is not an odd number"
      }
    }
  </script>
```

```
</head>
Enter Odd Number:
<asp:TextBox ID="txtOddValue" runat="server"></asp:TextBox>
<asp:CustomValidator ID="cvOddValue" runat="server" ErrorMessage="Please provide an Odd number"
   ClientValidationFunction="IsOdd" OnServerValidate="cvOddNumber_ServerValidate"
   ControlToValidate="txtOddValue"></asp:CustomValidator>
```

Note: If Javascript is disabled on Client/Browser, for validation Custom Validator, the server event "*ServerValidate*" must be handled.

Handling cvOddNumber_ServerValidate

```
protected void cvOddNumber_ServerValidate(object source, ServerValidateEventArgs args)
{
   int n;
   if (!int.TryParse(args.Value, out n))
      args.IsValid = false;
   else if (n % 2 == 0)
      args.IsValid = false;
   else
      args.IsValid = true;
}
```

CausesValidation Property

To skip validation if submit button is clicked

```
<asp:Button ID="btnSave" runat="server" Text="Save" OnClick="btnSave_Click" />
<asp:Button ID="btnCancel" runat="server" Text="Cancel" CausesValidation="False" /><br />
```

Note: If Cancel button is clicked the form is submitted to server without doing any client side validation because its *CausesValidation= "False"*

Grouping of Controls (ValidationGroup Property)

In one webform (aspx page) we can have only one server side form tag.

ValidationGroup Property: Is used to group all the validation control into a single group. A form can have more than one group of controls and based on the group of the button clicked, only controls in given group are validated.

Use of validation group property

```
<asp:ValidationSummary ID="ValidationSummary2" runat="server" ValidationGroup="g1" />
Name:
```

```
<asp:TextBox ID="txtName" runat="server" ValidationGroup="g1" />
<asp:RequiredFieldValidator ID="rfvName" runat="server" ControlToValidate="txtName"
   ErrorMessage="Please provide the value for txtName" ValidationGroup="g1" />
<asp:Button ID="btnSave" runat="server" Text="Save" ValidationGroup="g1" />
```

To show Text of all the validation controls when the page is rendered:

Page_Load event

```
protected void Page_Load(object sender, EventArgs e)
{
   if (!IsPostBack)
   {
      Page.Validate("");//Validates all the validation controls on the page
      ValidationSummary1.Style["display"] = "none";   //It does not display the ErrorMessages for the first time.
   }
}
```

To handle situation where client side javascript is disabled:

To handle when Javascript is disabled at client side

```
protected void btnSave_Click(object sender, EventArgs e)
{
   if (!Page.IsValid) //validation is done on server. This is needed if the browser doesn't support java script.
      return;
   //Write here code for taking data from controls and using for whatever purpose it might be for.
   Response.Write("Data taken from controls and saved...");
}
```

Note: Page.IsValid checks for only those validation controls whose GroupName is same as GroupName of the Button clicked to submit the form

Notes:

- Page.Validators - It's a collection of all validation controls in that page.
- We can disable the validation control if we don't want them to work on both client and server side (Enabled = "False")
- The Server Controls which can be validated using Validation Controls are TextBox, FileUpload, DropDownList, ListBox etc…i.e. not all Server controls can be validated using Validation Controls.