.NET PROGRAMMING USING C#

Validation in ASP.NET

What is Validation?

- An important aspect of creating ASP.NET Web pages for user input is to be able to check that the information users enter is valid.
- ASP.NET provides a set of validation controls that provide an easy-to-use but powerful way to check for errors and, if necessary, display messages to the user.

- There are two ways we can perform validation:
  - Client side validation
  - Server side validation

Client Side Validation

- Client side validation is something that will happen on user’s browser. The validation will occur before the data gets posted back to server.
- It is a good idea to have client side validation as the user gets to know what needs to be changed immediately, i.e., **no round trips to servers are made**.
- From the user’s point of view, it gives them fast response and from the developer’s point of view, it saves valuable resources of server.
- **JavaScript** provides full control to the developer on how client side validation should happen but developers will have to write the validation code themselves.

Server Side Validation

- Server side validation occurs at server.
- The benefit of having server side validation is that if the **user somehow bypasses the client side validation** then we can catch the problem on the server side.
- So having server side validation provides more security and ensures that no invalid data gets processed by the application.

List of Server Side Validation

- ***RequiredFiledValidator***
- Ensures that the user does not skip a mandatory entry field.
- ***CompareValidator***
- Compares one controls value with another controls value, constants and data type using a comparison operator
- ***RangeValidator***
- Checks the user's input is in a given range

- ***RegularExpressionValidator***
- Checks that the user's entry matches a pattern defined by a regular expression.
- ***CustomValidator***
- Checks the user's entry using custom-coded validation logic.
- ***ValidationSummary***
- Displays a summary of all validation errors inline on a web page, in a message box, or both.

Required Filed Validator

- You can use it to make sure that the user has entered something in a control.

- Properties
- ***ControlToValidate:***

  Indicates the input control to validate.
- ***ErrorMessage:***

  Indicates error string
- ***Text:***

  Error text to be shown if validation fails.

- ***SetFocusOnError – true, false***

  Set SetFocusOnError to true on one or several of your validators, to give focus to the first invalid field when the form is validated.

- ***Display – none, static, dynamic***

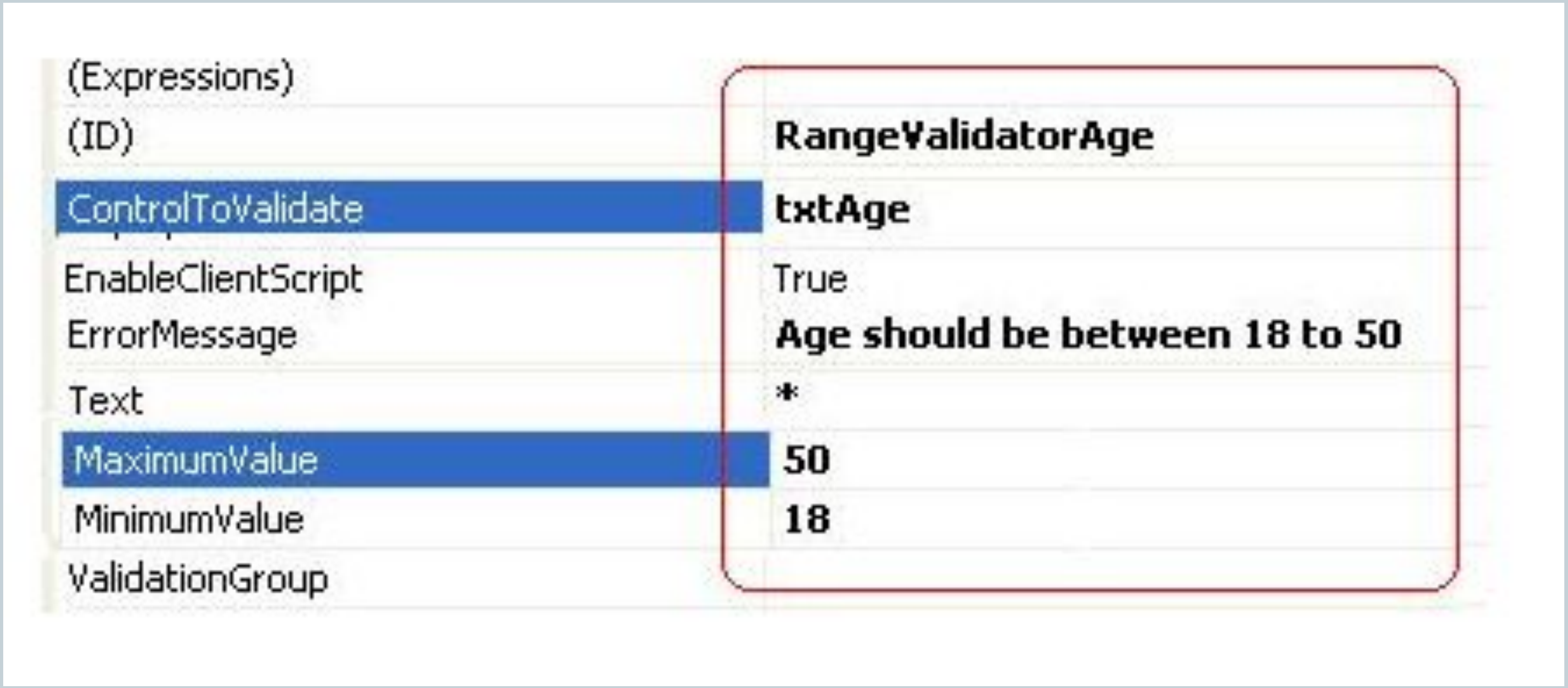
  This attribute decides how the validator is rendered to your page.

- ***ValidationGroup***

Range Validator

- The RangeValidator Server Control is another validator control, which checks to see if a control value is within a valid range.
- Properties
- ControlToValidate, ErrorMessage, Text, Display, ValidationGroup
- **MinimumValue**
- **MaximumValue**
- Type - Currency, Date, Double, Integer, String

Example of RangeValidator



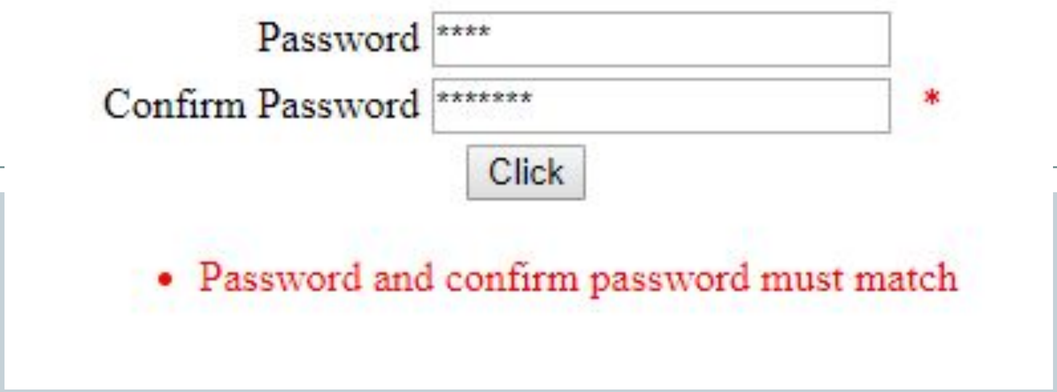
(Expressions)	
(ID)	RangeValidatorAge
ControlToValidate	txtAge
EnableClientScript	True
ErrorMessage	Age should be between 18 to 50
Text	*
MaximumValue	50
MinimumValue	18
ValidationGroup	

Compare Validator

- The CompareValidator control allows you to make comparison to compare data entered in an input control with a constant value or a value in a different control.
- This control will compare the value of its **ControlToValidate** with **ControlToCompare**.

Properties

- ErrorMessage, Text, Display, ValidationGroup
- ControlToValidate
- ControlToCompare
- ValueToCompare
- Operator – Equal, NotEqual, LessThan, GreaterThan, GreaterThanEqual, LessThanEqual, DataTypeCheck


Password ****

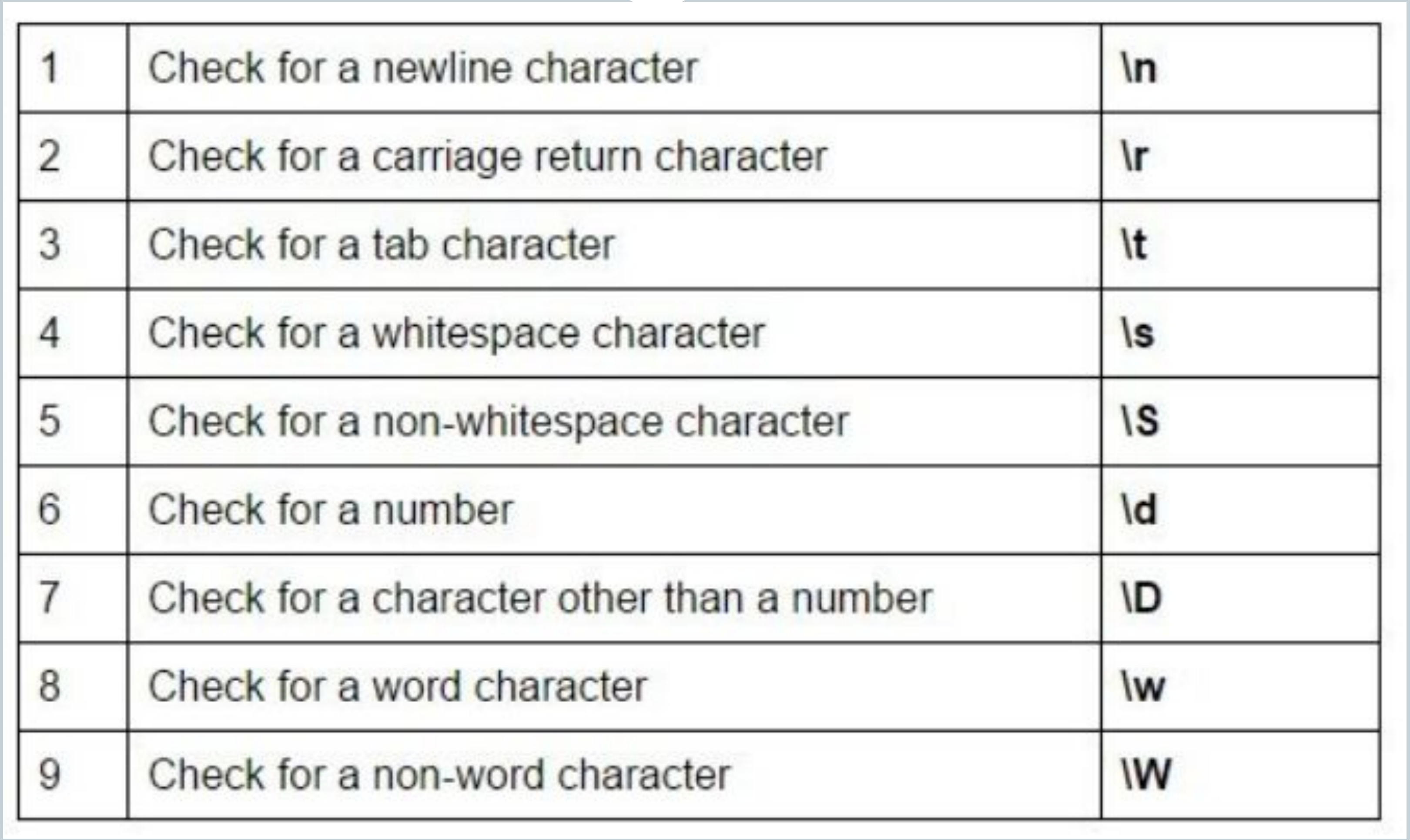
Confirm Password ******* *

- Password and confirm password must match

```
<asp:CompareValidator
    ID="CompareValidator1" runat="server"
    ControlToCompare="TextBox1"
    ControlToValidate="TextBox2"
    ErrorMessage="Password and confirm
password must match"
    Text = “*”>
</asp:CompareValidator>
```

Regular Expression Validator

- RegularExpressionValidator useful when we want input data to be in some specific format.
- <u>Properties</u>
- ControlToValidate, ErrorMessage, Text, Display
- ValidationExpression



1	Check for a newline character	**\n**
2	Check for a carriage return character	**\r**
3	Check for a tab character	**\t**
4	Check for a whitespace character	**\s**
5	Check for a non-whitespace character	**\S**
6	Check for a number	**\d**
7	Check for a character other than a number	**\D**
8	Check for a word character	**\w**
9	Check for a non-word character	**\W**

Custom Validator

- If none of the other validators can help you, the **CustomValidator** usually can. It doesn't come with a predefined way of working; you write the code for validating your self.
- The CustomValidator Control can be used on client side and server side. JavaScript is used to do client validation and you can use any .NET language to do server side validation.

- Properties
- ControlToValidate, ErrorMessage, Text, Display
- Event
- ServerValidate

Validation Summary

- ASP.NET has provided an additional control that complements the validator controls.

  The ValidationSummary control is reporting control, which is used by the other validation controls on a page.

ASP.NET State Management Engine

Why is it Required in ASP.NET

- Browsers are generally state less.
- Stateless means, whenever we visit a website, our browser communicates with the respective server depending on our requested functionality or the request. The browser communicates with the respective server using the HTTP or HTTPs protocol.
- But after that response HTTP/HTTPs doesn't remember what website or URL we visited or in other words we can say it doesn't hold the state of a previous website that we visited before closing our browser that is called stateless.

- In ASP.NET there are the following 2 State Management methodologies:
  - Client Side Statement
  - Server Side Statement

Client Side State Management

- View State
- Hidden fields
- Cookies
- Query Strings

Server Side State Management

- Application State
- Session State

Cookies

- A set of Cookies is a small text file that is stored in the user's hard drive using the client's browser.
- The cookie access depends upon the life cycle or expiration of that specific cookie file.

Some features of cookies are:

- Store information temporarily
- It's just a simple small sized text file
- Can be changed depending on requirements
- User Preferred
- Requires only a few bytes or KBs of space for creating cookies

Writing and Reading Cookies

- **By using Response directly**
- **Writing**
  - Response.Cookies["userName"].Value = “INDUS";
  - Response.Cookies[“userName”].Expires = DataTime.Now.AddDays(10);
- **Reading**
  - String value = Request.Cookies[“userName”].Value;

- **By using Response directly**
- **Writing**
  - Response.Cookies[“login"][“username”] = “INDUS";
  - Response.Cookies[“login"][“password”] = “INDUS";
  - Response.Cookies[“login”].Expires = DataTime.Now.AddDays(10);
- **Reading**
  - String value1 = Request.Cookies[“login”][“username];
  - String value2 = Request.Cookies[“login”][“password];

HttpCookie Class - Properties

1. Expires
   - Which is used to set expire date for the cookies.
2. Values
   - We can manipulate cookies with key/value pair.
3. Value
   - We can manipulate individual cookie.
4. Name
   - It contains the name of the cookie.

Deleting Cookies

- Response.Cookies[“user"].Expires =

  DateTime.Now.AddDays(-1);

Query String

- Query string is one of the technique to send data from one webform to another through URL.
- Query string consist of two parts (**field and value**), and each of pair separated by ampersand **(&)**.
- ?(Question Mark), indicates the beginning of a query string and it's value.

- *Query String – Response.Redirect(“webform2.aspx?name=‘Indus’lastName=‘University’”);*
- *url - http://www.localhost.com/Webform2.aspx?name=Indus&lastName=University*

1. Webform2.aspx this is the page your browser will go.
2. name=Indus you send a name variable which is set to Indus
3. lastName=University you send a lastName variable which is set to University

How to access data in another page

- lblName.Text = Request.QueryString[“name"];
- lblLastName.Text = Request.QueryString["*lastName* "];

Hidden Field

- A hidden field is used for storing small amounts of data on the client side. In most simple words it's just a container of some objects but their result is not rendered on our web browser. It is invisible in the browser.
- It stores the value between the roundtrip. Anyone can see HiddenField details by simply viewing the source of document.
- HiddenFields are not encrypted or protected and can be changed by any one. However, from a security point of view, this is not suggested.

Some features of hidden fields are:

- Contains a small amount of memory
- Direct functionality access

Store the value in HiddenField -

```
<asp:HiddenField ID="hDateTime" runat="server"
/>
```

Retrieve the value from HiddenField -

```
lblDateTime.Text =
Convert.ToString(hDateTime.Value);
```

View State

- In ASP.NET applications the user wants to maintain or store their data temporarily after a post-back.
- In this case VIEW STATE is the most used and preferred way of doing that.
- This property is enabled by default but we can make changes depending on our functionality with the help of EnableViewState value to either TRUE for enabling it or FALSE for the opposite operation.

Some of the features of view state are:

- It is page-level State Management
- Used for holding data temporarily
- Can store any type of data

How to Enable and Disable View State

- In Page Directive set EnableViewState="false“;

Syntax For View State

- ViewState[<variable_name>] = <value>; //set view state value

- String abc = ViewState[<variable_name>]; //get view state value

Example of ViewState

- Set value of ViewState
  - ViewState["NameOfUser"] = NameField.Text;
- Get value of ViewState
  - NameLabel.Text = ViewState["NameOfUser"].ToString();

Sessions In Asp.Net

- When a user connects to an ASP.NET website, a new session object is created.
- When session state is turned on, a new session state object is created for each new request.
- This session state object becomes part of the context and it is available through the page.

Session ID

- Sessions are identified and tracked with a 120-bit SessionID, which is passed from client to server and back as cookie or a modified URL.
- The SessionID is globally unique and random.

HttpSessionState (Session)

- **Properties**
  - Count
  - SessionID
  - TimeOut

- IsNewSession
- IsCookieless
- Keys

HttpSessionState (Session)

- Methos
  - Add(string key, string value)
  - Abandon()
  - Clear() or RemoveAll()
  - Remove(string key)
  - RemoveAt(int index)

How to Add Session

- Session[“user”] = “admin”
- or
- Session.Add(“user”,”admin”)

How to Remove Session

- Session.Abandon();
- or
- Session.Remove(“user”) – only one key remove

Application State

- Application state is a server side state management technique.
- The date stored in application state is common for all users of that particular ASP.NET application and can be accessed anywhere in the application.
- It is also called application level state management.
- Data stored in the application should be of small size.
- Application object is not permanent and is lost any time the application is restarted.

Application State in Global.asax file

ASP.NET provides three events in ***global.asax*** that enable you to initialize Application variables and respond to Application errors:

1. Application_Start
2. Application_End
3. Application_Error
4. Session_Start
5. Session_End

web.config

- Visual Studio generates a default web.config file for each project.
- An application can execute without a web.config file, however, you cannot debug an application without a web.config file.
- Configuration file is used to manage various settings that define a website. The settings are stored in XML files that are separate from your application code

What Web.config file contains?

- Database connections
- Caching settings
- Session States
- Error Handling
- Security

System.Web Element
<system.web> </system.web>

- Authentication
- Authorization
- Caching
- CustomErrors
- Trace

- Pages
- RoleManager
- SessionState
- HttpCookies

Authentication

```
<system.web>
        <authorization>
            <allow roles="admin"/>
            <deny users="*"/>
        </authorization>
 </system.web>
```

Authentication and Authorization

```
<authentication mode="Forms"/>
    <authorization> <deny users="?"/>
</authorization>
</authentication>
```