LECTURE NOTES

TITLE OF THE COURSE: PROGRAMMING IN ASP.NET

PAPER CODE: 18PCS10

COMPILED BY

DR. S.HARI GANESH

DEPARTMENT OF COMPUTER SCIENCE

HH THE RAJAH'S COLLEGE, PUDUKOTTAI

**** UNIT-I ****

FAQ: WHAT IS ASP.NET?

ASP.NET is Microsoft corporation's propitiatory frontend language. It means "Active Server Pages". In the name itself, it's saying I'm active. The ASP code will run in the client machine and used to represent the data which is returning from the server.

ASP.NET is a web development platform, which provides a programming model, a comprehensive software infrastructure and various services required to build up robust web applications for PC, as well as mobile devices.

ASP.NET works on top of the HTTP protocol, and uses the HTTP commands and policies to set a browser-to-server bilateral communication and cooperation.

ASP.NET is a part of Microsoft .Net platform. ASP.NET applications are compiled codes, written using the extensible and reusable components or objects present in .Net framework. These codes can use the entire hierarchy of classes in .Net framework.

The ASP.NET application codes can be written in any of the following languages:

- C#
- Visual Basic.Net
- Jscript
- J#

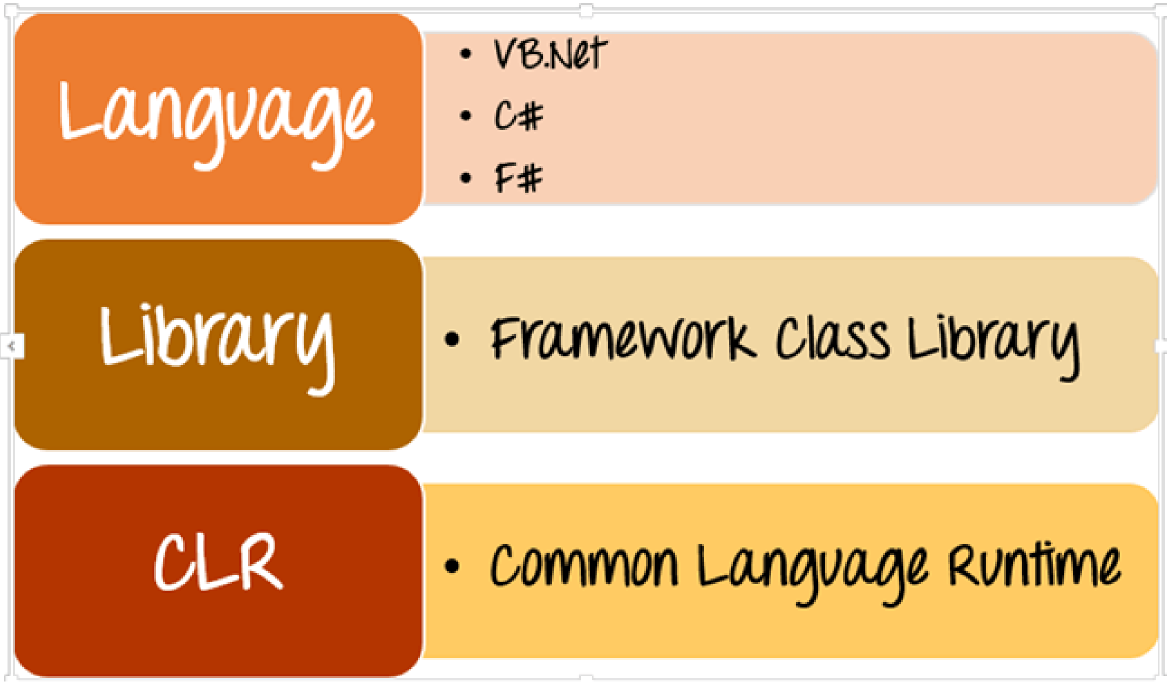
ASP.NET is used to produce interactive, data-driven web applications over the internet. It consists of a large number of controls such as text boxes, buttons, and labels for assembling, configuring, and manipulating code to create HTML pages.

FAQ: EXPLAIN THE FRAME WORK ARCHTECTTURE OF ASP.NET?

ASP.NET Architecture and its Components

ASP.Net is a framework which is used to develop a Web-based application. The basic architecture of the ASP.Net framework is as shown below.

ASP.NET Architecture Diagram

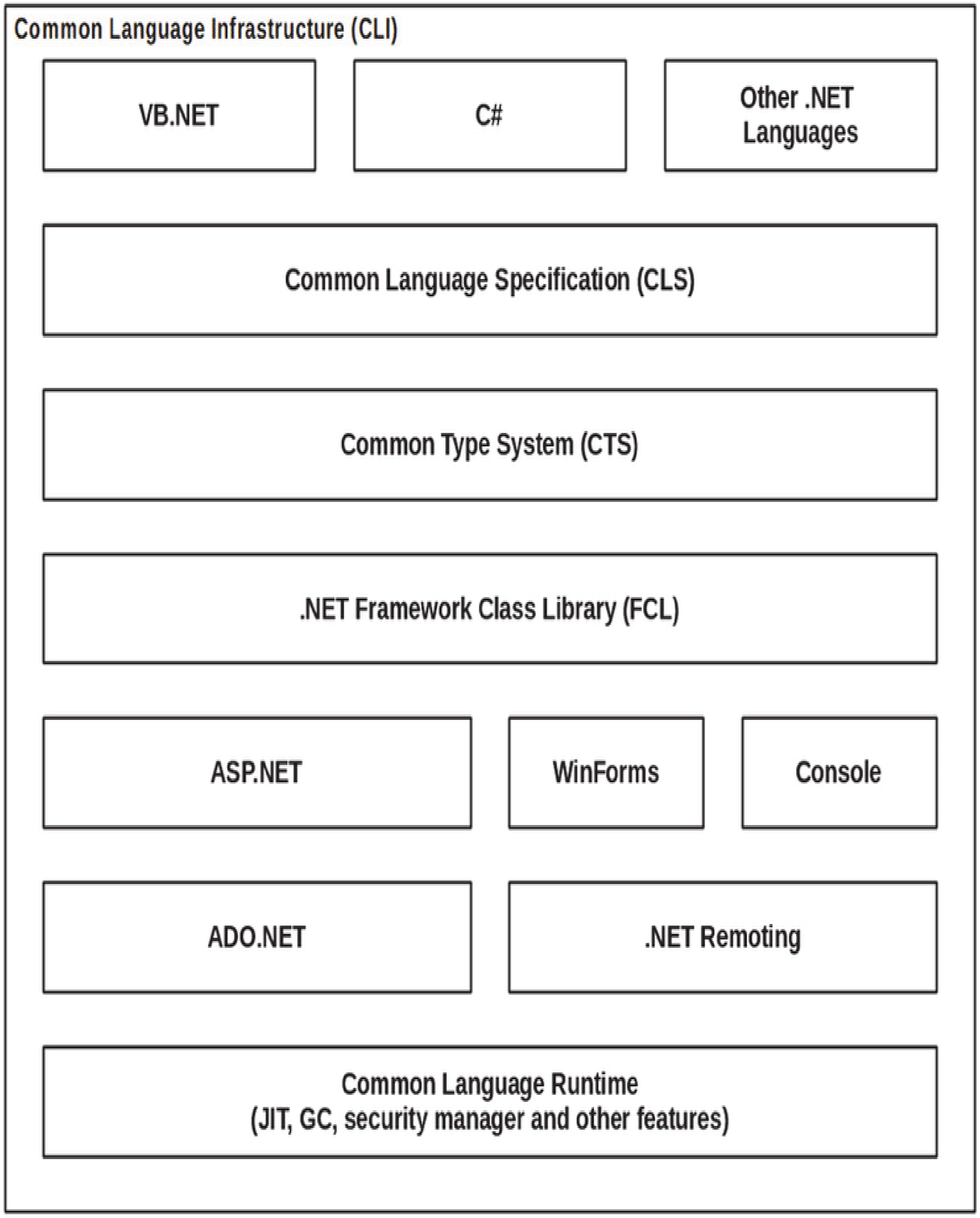


ASP.NET Architecture Diagram

The architecture of the.Net framework is based on the following key components

1. **Language** – A variety of languages exists for .net framework. They are VB.net and C#. These can be used to develop web applications.
2. **Library** - The .NET Framework includes a set of standard class libraries. The most common library used for web applications in .net is the Web library. The web library has all the necessary components used to develop.Net web-based applications.
3. **Common Language Runtime** - The Common Language Infrastructure or CLI is a platform. .Net programs are executed on this platform. The CLR is used for performing key activities. Activities include Exception handling and Garbage collection.

ASP.NET FRAME WORK


Common Language Infrastructure (CLI)
VB.NET
C#
Other .NET Languages
Common Language Specification (CLS)
Common Type System (CTS)
.NET Framework Class Library (FCL)
ASP.NET
WinForms
Console
ADO.NET
.NET Remoting
Common Language Runtime
(JIT, GC, security manager and other features)

Components of .Net Framework 3.5

Before going to the next session on Visual Studio.Net, let us go through at the various components of the .Net framework 3.5. The following table describes the components of the .Net framework 3.5 and the job they perform:

Components and their Description
(1) Common Language Runtime or CLR It performs memory management, exception handling, debugging, security checking, thread execution, code execution, code safety, verification, and compilation. The code that is directly managed by the CLR is called the managed code. When the managed code is compiled, the compiler converts the source code into a CPU independent intermediate language (IL) code. A Just In Time(JIT) compiler compiles the IL code into native code, which is CPU specific.
(2) .Net Framework Class Library It contains a huge library of reusable types. classes, interfaces, structures, and enumerated values, which are collectively called types.
(3) Common Language Specification It contains the specifications for the .Net supported languages and implementation of language integration.
(4) Common Type System It provides guidelines for declaring, using, and managing types at runtime, and cross-language communication.
(5) Metadata and Assemblies Metadata is the binary information describing the program, which is either stored in a portable executable file (PE) or in the memory. Assembly is a logical unit consisting of the assembly manifest, type metadata, IL code, and a set of resources like image files.

(6) Windows Forms

Windows Forms contain the graphical representation of any window displayed in the application.

(7) ASP.NET and ASP.NET AJAX

ASP.NET is the web development model and AJAX is an extension of ASP.NET for developing and implementing AJAX functionality. ASP.NET AJAX contains the components that allow the developer to update data on a website without a complete reload of the page.

(8) ADO.NET

It is the technology used for working with data and databases. It provides access to data sources like SQL server, OLE DB, XML etc. The ADO.NET allows connection to data sources for retrieving, manipulating, and updating data.

(9) Windows Workflow Foundation (WF)

It helps in building workflow-based applications in Windows. It contains activities, workflow runtime, workflow designer, and a rules engine.

(10) Windows Presentation Foundation

It provides a separation between the user interface and the business logic. It helps in developing visually stunning interfaces using documents, media, two and three dimensional graphics, animations, and more.

(11) Windows Communication Foundation (WCF)

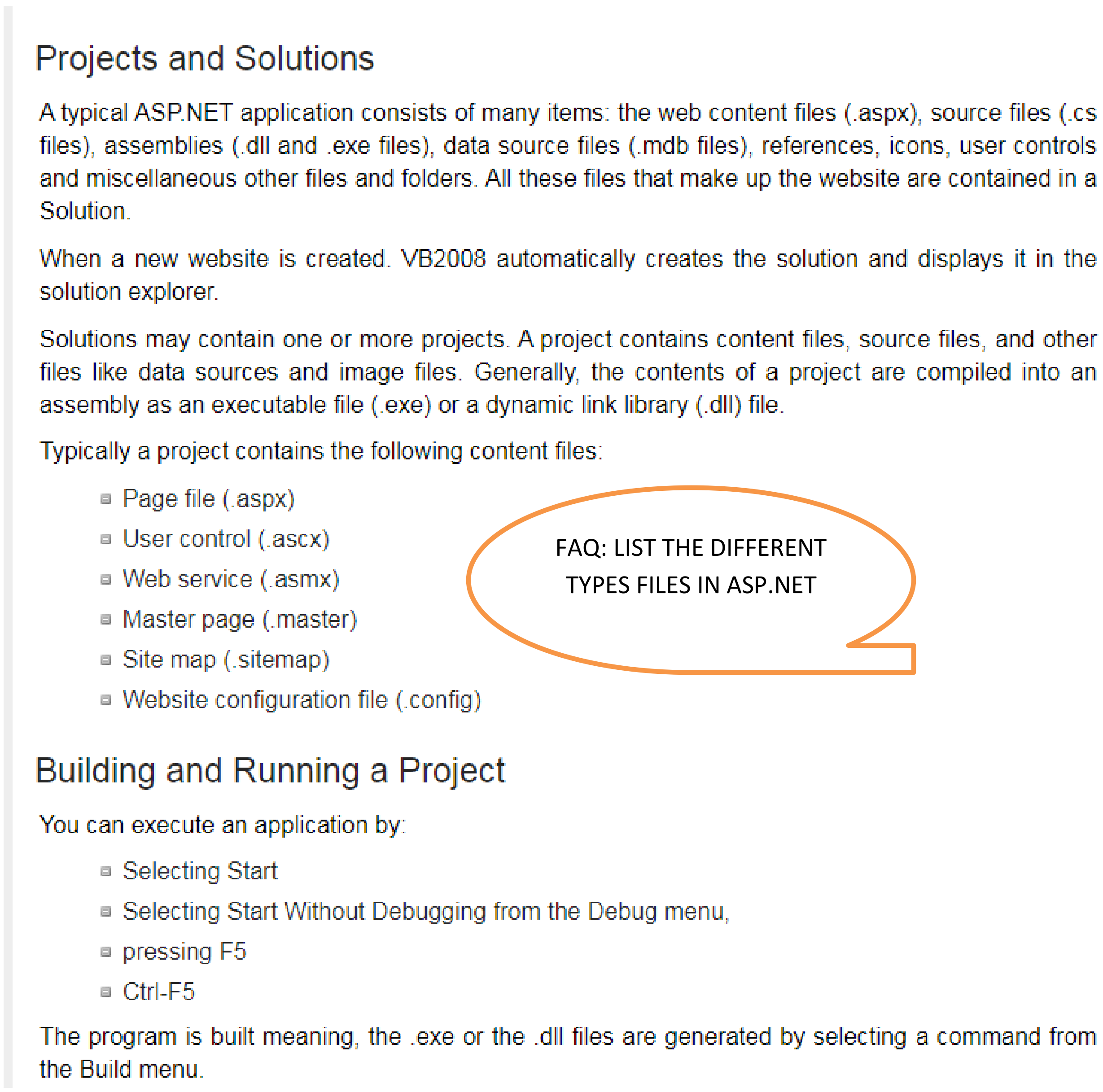
It is the technology used for building and executing connected systems.

(12) Windows CardSpace

It provides safety for accessing resources and sharing personal information on the internet.

(13) LINQ

It imparts data querying capabilities to .Net languages using a syntax which is similar to the tradition query language SQL.



Projects and Solutions

A typical ASP.NET application consists of many items: the web content files (.aspx), source files (.cs files), assemblies (.dll and .exe files), data source files (.mdb files), references, icons, user controls and miscellaneous other files and folders. All these files that make up the website are contained in a Solution.

When a new website is created. VB2008 automatically creates the solution and displays it in the solution explorer.

Solutions may contain one or more projects. A project contains content files, source files, and other files like data sources and image files. Generally, the contents of a project are compiled into an assembly as an executable file (.exe) or a dynamic link library (.dll) file.

Typically a project contains the following content files:

- Page file (.aspx)
- User control (.ascx)
- Web service (.asmx)
- Master page (.master)
- Site map (.sitemap)
- Website configuration file (.config)

FAQ: LIST THE DIFFERENT TYPES FILES IN ASP.NET

Building and Running a Project

You can execute an application by:

- Selecting Start
- Selecting Start Without Debugging from the Debug menu,
- pressing F5
- Ctrl-F5

The program is built meaning, the .exe or the .dll files are generated by selecting a command from the Build menu.

ASP.NET life cycle specifies, how:

- ASP.NET processes pages to produce dynamic output
- The application and its pages are instantiated and processed
- ASP.NET compiles the pages dynamically

The ASP.NET life cycle could be divided into two groups:

- Application Life Cycle
- Page Life Cycle

ASP.NET Application Life Cycle

The application life cycle has the following stages:

- User makes a request for accessing application resource, a page. Browser sends this request to the web server.
- A unified pipeline receives the first request and the following events take place:
    - An object of the class ApplicationManager is created.
    - An object of the class HostingEnvironment is created to provide information regarding the resources.
    - Top level items in the application are compiled.
- Response objects are created. The application objects such as HttpContext, HttpRequest and HttpResponse are created and initialized.
- An instance of the HttpApplication object is created and assigned to the request.
- The request is processed by the HttpApplication class. Different events are raised by this class for processing the request.

FAQ: DESCIRBE THE PAGE LIFE CYCLE OF ASP.NET

ASP.NET Page Life Cycle

When a page is requested, it is loaded into the server memory, processed, and sent to the browser. Then it is unloaded from the memory. At each of these steps, methods and events are available, which could be overridden according to the need of the application. In other words, you can write your own code to override the default code.

The Page class creates a hierarchical tree of all the controls on the page. All the components on the page, except the directives, are part of this control tree. You can see the control tree by adding trace= "true" to the page directive. We will cover page directives and tracing under 'directives' and 'event handling'.

The page life cycle phases are:

- Initialization
- Instantiation of the controls on the page
- Restoration and maintenance of the state
- Execution of the event handler codes
- Page rendering

Understanding the page cycle helps in writing codes for making some specific thing happen at any stage of the page life cycle. It also helps in writing custom controls and initializing them at right time, populate their properties with view-state data and run control behavior code.

Following are the different stages of an ASP.NET page:

- **Page request** - When ASP.NET gets a page request, it decides whether to parse and compile the page, or there would be a cached version of the page; accordingly the response is sent.
- **Starting of page life cycle** - At this stage, the Request and Response objects are set. If the request is an old request or post back, the IsPostBack property of the page is set to true. The UICulture property of the page is also set.
- **Page initialization** - At this stage, the controls on the page are assigned unique ID by setting the UniqueID property and the themes are applied. For a new request, postback data is loaded and the control properties are restored to the view-state values.
- **Page load** - At this stage, control properties are set using the view state and control state values.
- **Validation** - Validate method of the validation control is called and on its successful execution, the IsValid property of the page is set to true.
- **Postback event handling** - If the request is a postback (old request), the related event handler is invoked.
- **Page rendering** - At this stage, view state for the page and all controls are saved. The page calls the Render method for each control and the output of rendering is written to the OutputStream class of the Response property of page.
- **Unload** - The rendered page is sent to the client and page properties, such as Response and Request, are unloaded and all cleanup done.

ASP.NET Page Life Cycle Events

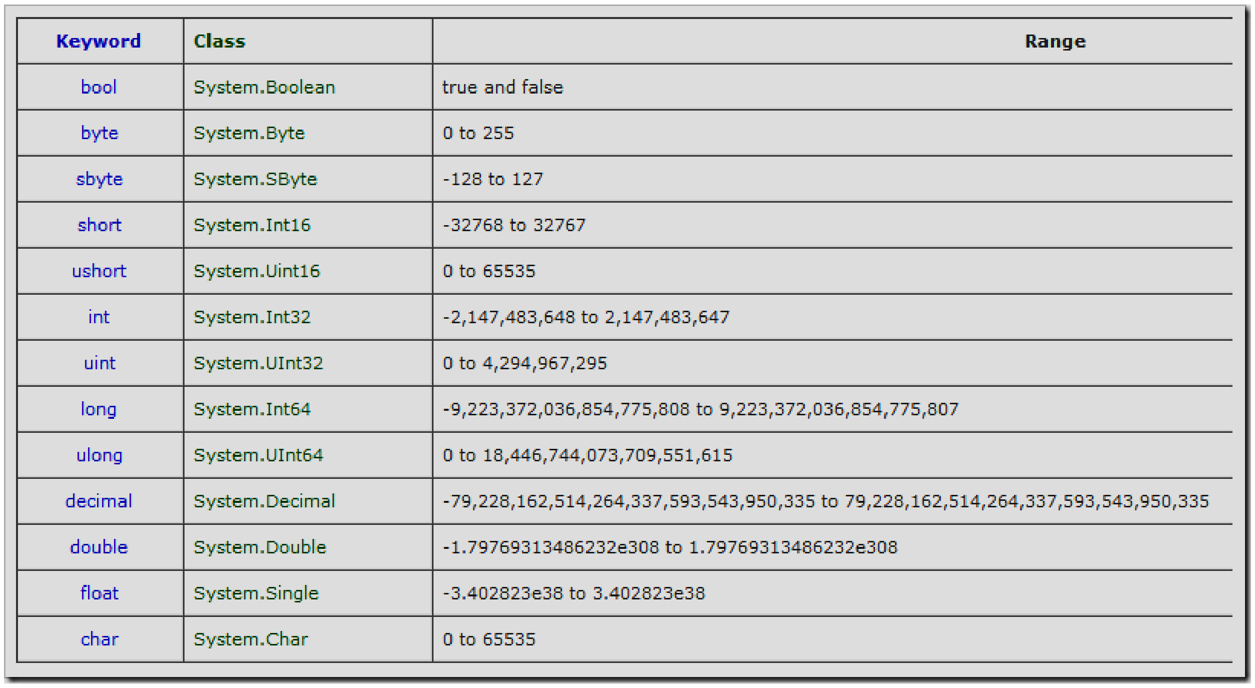
At each stage of the page life cycle, the page raises some events, which could be coded. An event handler is basically a function or subroutine, bound to the event, using declarative attributes such as Onclick or handle.

Following are the page life cycle events:

- **PreInit** - PreInit is the first event in page life cycle. It checks the IsPostBack property and determines whether the page is a postback. It sets the themes and master pages, creates dynamic controls, and gets and sets profile property values. This event can be handled by overloading the OnPreInit method or creating a Page_PreInit handler.
- **Init** - Init event initializes the control property and the control tree is built. This event can be handled by overloading the OnInit method or creating a Page_Init handler.
- **InitComplete** - InitComplete event allows tracking of view state. All the controls turn on view-state tracking.

- **LoadViewState** - LoadViewState event allows loading view state information into the controls.
- **LoadPostData** - During this phase, the contents of all the input fields are defined with the <form> tag are processed.
- **PreLoad -** PreLoad occurs before the post back data is loaded in the controls. This event can be handled by overloading the OnPreLoad method or creating a Page_PreLoad handler.
- **Load** - The Load event is raised for the page first and then recursively for all child controls. The controls in the control tree are created. This event can be handled by overloading the OnLoad method or creating a Page_Load handler.
- **LoadComplete** - The loading process is completed, control event handlers are run, and page validation takes place. This event can be handled by overloading the OnLoadComplete method or creating a Page_LoadComplete handler
- **PreRender** - The PreRender event occurs just before the output is rendered. By handling this event, pages and controls can perform any updates before the output is rendered.
- **PreRenderComplete** - As the PreRender event is recursively fired for all child controls, this event ensures the completion of the pre-rendering phase.
- **SaveStateComplete** - State of control on the page is saved. Personalization, control state and view state information is saved. The HTML markup is generated. This stage can be handled by overriding the Render method or creating a Page_Render handler.
- **UnLoad -** The UnLoad phase is the last phase of the page life cycle. It raises the UnLoad event for all controls recursively and lastly for the page itself. Final cleanup is done and all resources and references, such as database connections, are freed. This event can be handled by modifying the OnUnLoad method or creating a Page_UnLoad handler.

FAQ: DISCUSS THE VARIOUS DATATYPES OF ASP.NET

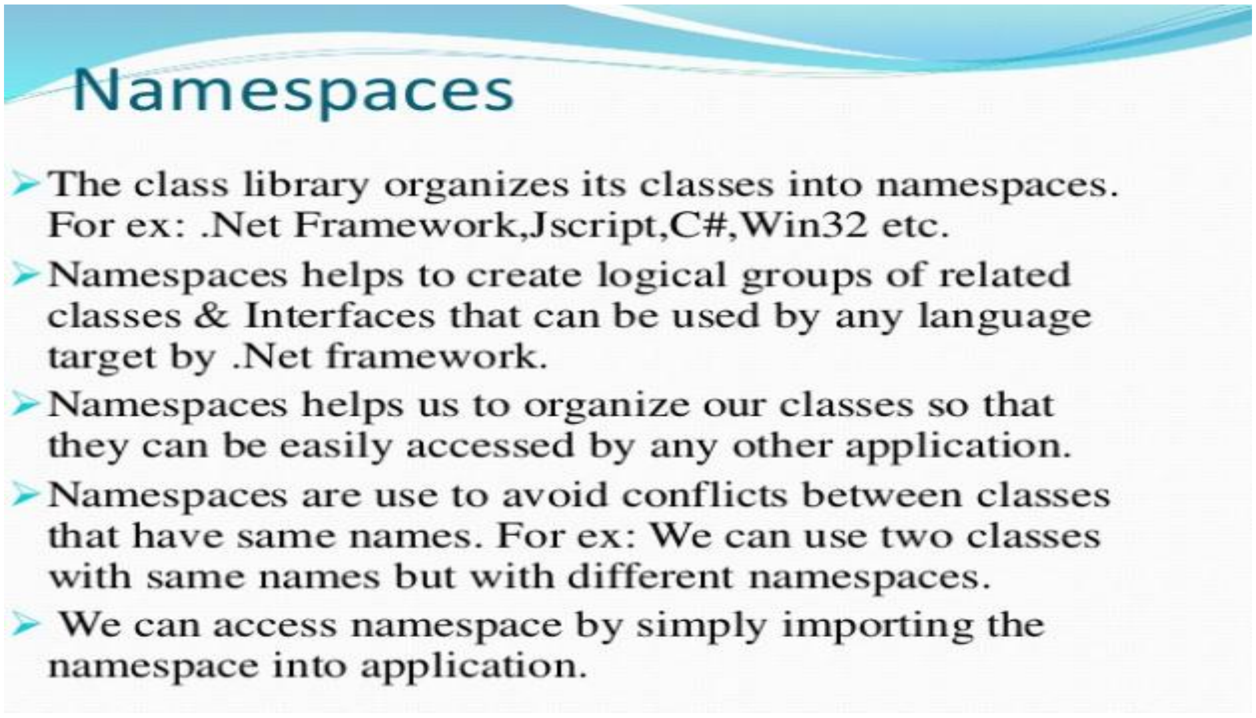


Keyword	Class	Range
bool	System.Boolean	true and false
byte	System.Byte	0 to 255
sbyte	System.SByte	-128 to 127
short	System.Int16	-32768 to 32767
ushort	System.Uint16	0 to 65535
int	System.Int32	-2,147,483,648 to 2,147,483,647
uint	System.UInt32	0 to 4,294,967,295
long	System.Int64	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
ulong	System.UInt64	0 to 18,446,744,073,709,551,615
decimal	System.Decimal	-79,228,162,514,264,337,593,543,950,335 to 79,228,162,514,264,337,593,543,950,335
double	System.Double	-1.79769313486232e308 to 1.79769313486232e308
float	System.Single	-3.402823e38 to 3.402823e38
char	System.Char	0 to 65535

DATA TYPES IN ASP.NET

FAQ: WRITE ABOUT THE NAMESPACES IN ASP.NET

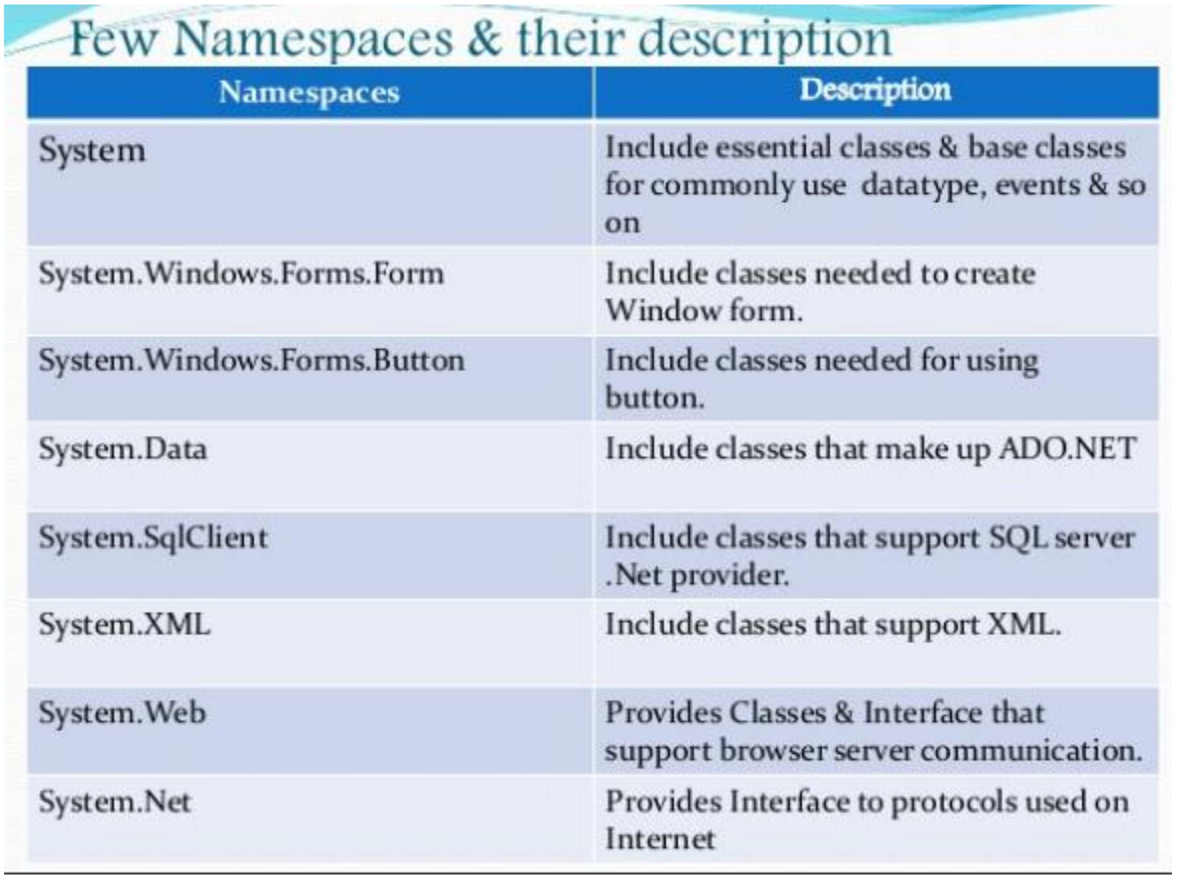
OBJECTS AND NAME SPACES



Namespaces

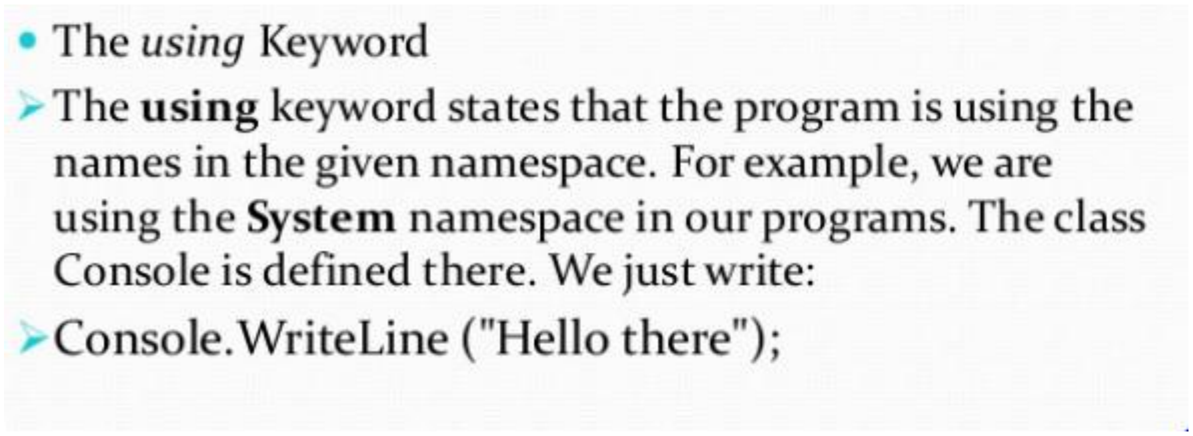
- The class library organizes its classes into namespaces. For ex: .Net Framework,Jscript,C#,Win32 etc.
- Namespaces helps to create logical groups of related classes & Interfaces that can be used by any language target by .Net framework.
- Namespaces helps us to organize our classes so that they can be easily accessed by any other application.
- Namespaces are use to avoid conflicts between classes that have same names. For ex: We can use two classes with same names but with different namespaces.
- We can access namespace by simply importing the namespace into application.

The **GridView** control displays the values of a data source in a table. Each column represents a field, while each row represents a record. The **GridView** control supports the following features: Binding to data source controls, such as SqlDataSource



Few Namespaces & their description

Namespaces	Description
System	Include essential classes & base classes for commonly use datatype, events & so on
System.Windows.Forms.Form	Include classes needed to create Window form.
System.Windows.Forms.Button	Include classes needed for using button.
System.Data	Include classes that make up ADO.NET
System.SqlClient	Include classes that support SQL server .Net provider.
System.XML	Include classes that support XML.
System.Web	Provides Classes & Interface that support browser server communication.
System.Net	Provides Interface to protocols used on Internet



- The *using* Keyword
  - The **using** keyword states that the program is using the names in the given namespace. For example, we are using the **System** namespace in our programs. The class Console is defined there. We just write:
  - Console.WriteLine ("Hello there");

CODE BEHIND FILE

One major point of **Code-Behind** is that the **code** for all the Web pages is compiled into a DLL file that allows the web pages to be hosted free from any **Inline** Server **Code**. **Inline Code**. **Inline Code** refers to the **code** that is written inside an **ASP**.**NET** Web Page that has an extension of . **aspx**
In today's article, you will learn the differences between Code-Behind and Inline Code in ASP.NET.

Many people remain confused about the differences between Code-Behind and Inline Code. Here I am explaining both of them with examples that will help you to understand the differences between the two.

Code Behind

Code Behind refers to the code for an ASP.NET Web page that is written in a separate class file that can have the extension of .aspx.cs or .aspx.vb depending on the language used. Here the code is compiled into a separate class from which the .aspx file derives. You can write the code in a separate .cs or .vb code file for each .aspx page. One major point of Code-Behind is that the code for all the Web pages is compiled into a DLL file that allows the web pages to be hosted free from any Inline Server Code.

Inline Code

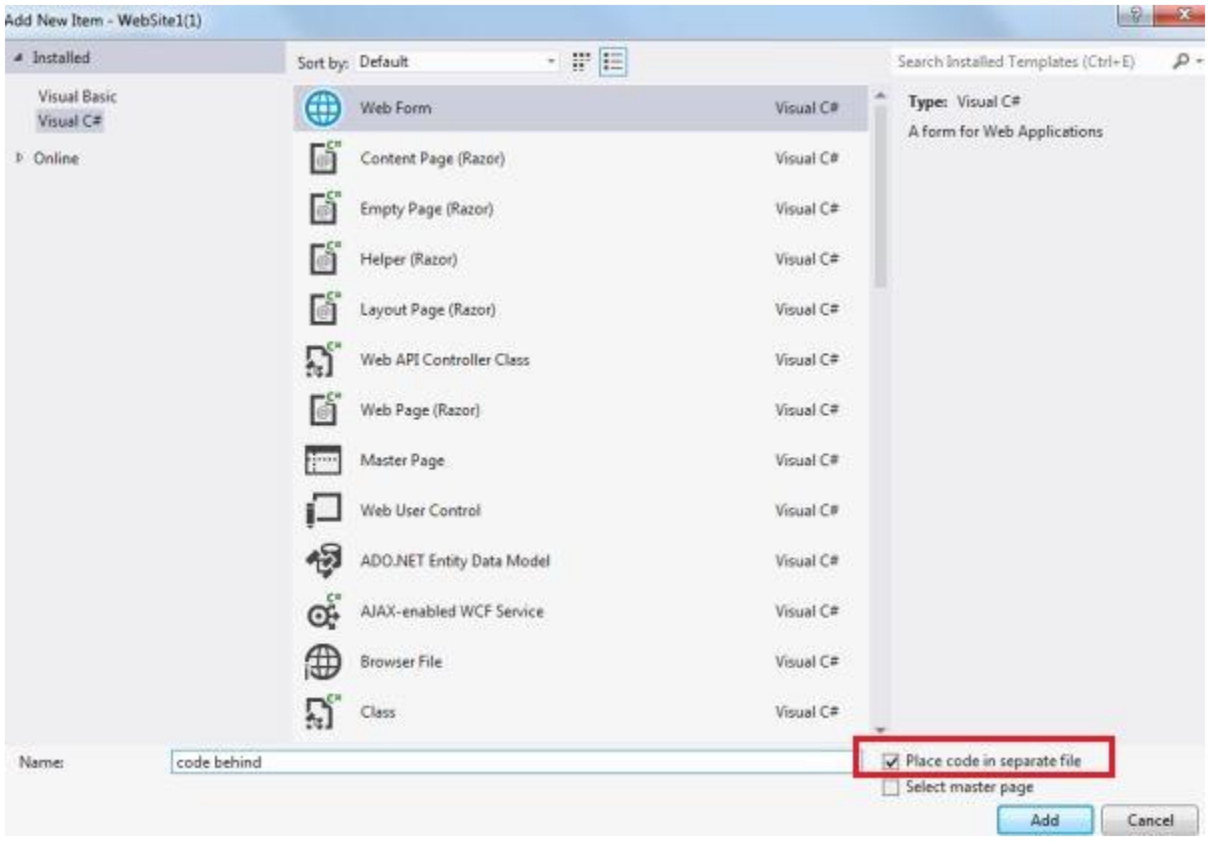
Inline Code refers to the code that is written inside an ASP.NET Web Page that has an extension of .aspx. It allows the code to be written along with the HTML source code using a <Script> tag. Its major point is that since it's physically in the .aspx file it's deployed with the Web Form page whenever the Web Page is deployed.

Now I will show you these differences by using an example.

Step 1

First of all, create a new Blank Website in Visual Studio, then add a Web page to it. Here **we are first creating a Web page for the Code Behind** so remember one thing **"the check box should**

be checked while adding this page". In other words, check the "Place the code in a separate file" and then click on the "Add" button.

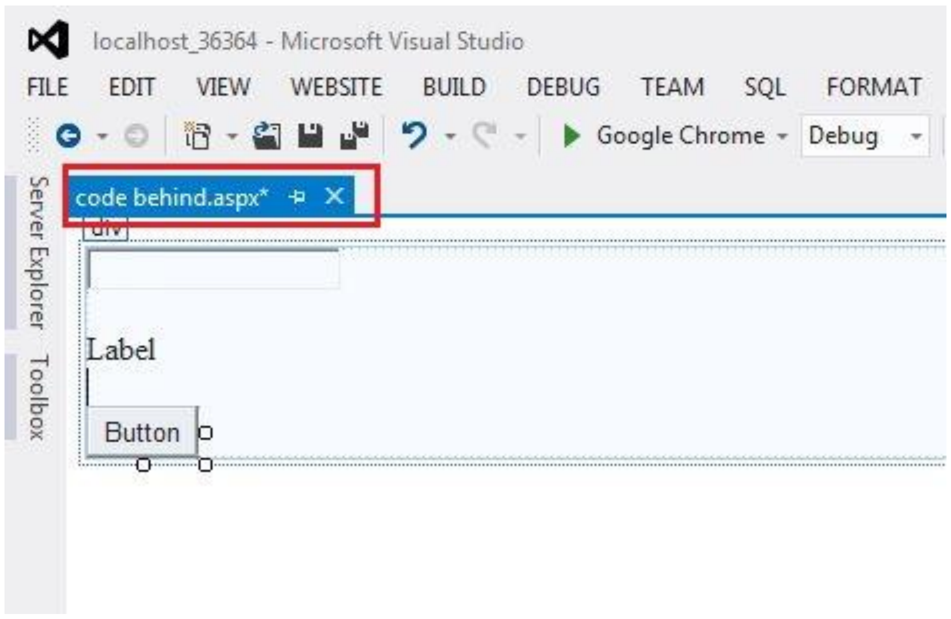

Add New Item - WebSite1(1)
Installed
Visual Basic
Visual C#
Online
Sort by: Default
Search Installed Templates (Ctrl+E)
Web Form Visual C#
Content Page (Razor) Visual C#
Empty Page (Razor) Visual C#
Helper (Razor) Visual C#
Layout Page (Razor) Visual C#
Web API Controller Class Visual C#
Web Page (Razor) Visual C#
Master Page Visual C#
Web User Control Visual C#
ADO.NET Entity Data Model Visual C#
AJAX-enabled WCF Service Visual C#
Browser File Visual C#
Class Visual C#
Type: Visual C#
A form for Web Applications
Add
Cancel

Name: code behind

[✓] Place code in separate file

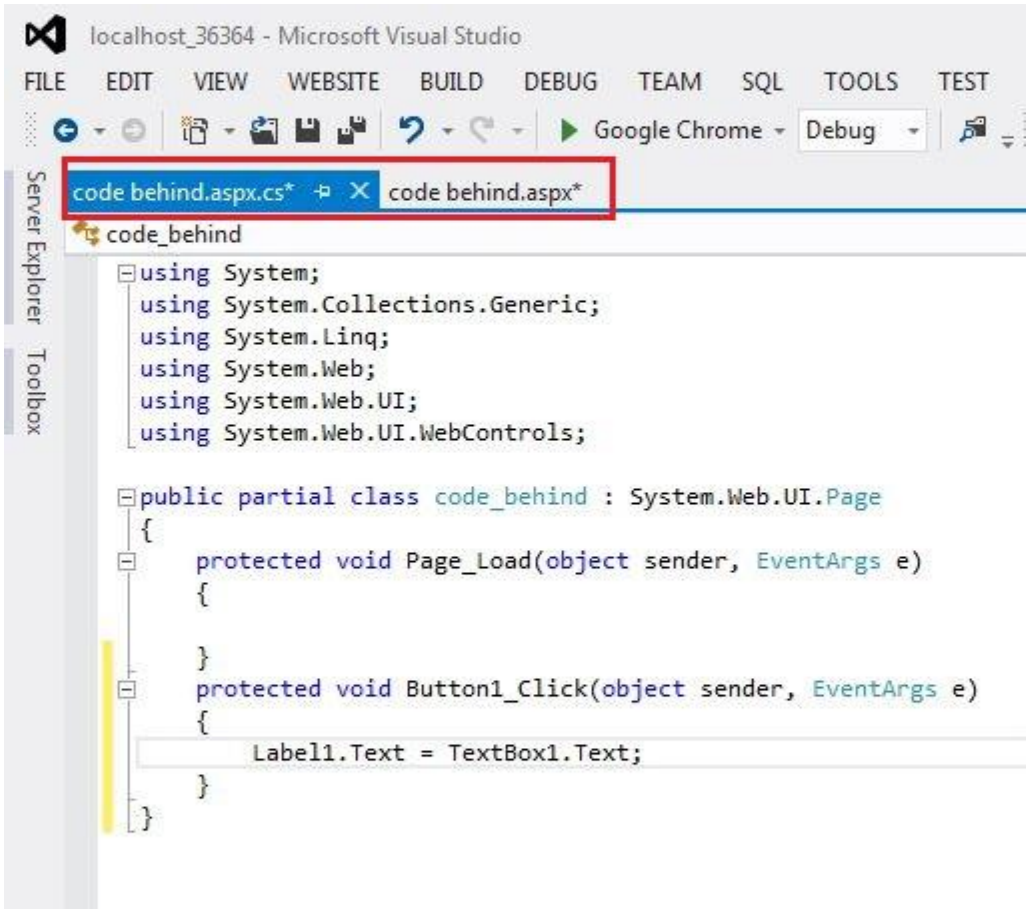
[ ] Select master page

Now on the .aspx page use a Button, a Link, and a Text Box.


localhost_36364 - Microsoft Visual Studio
FILE EDIT VIEW WEBSITE BUILD DEBUG TEAM SQL FORMAT
Google Chrome
Debug
code behind.aspx*
Server Explorer
Toolbox
div
Label
Button

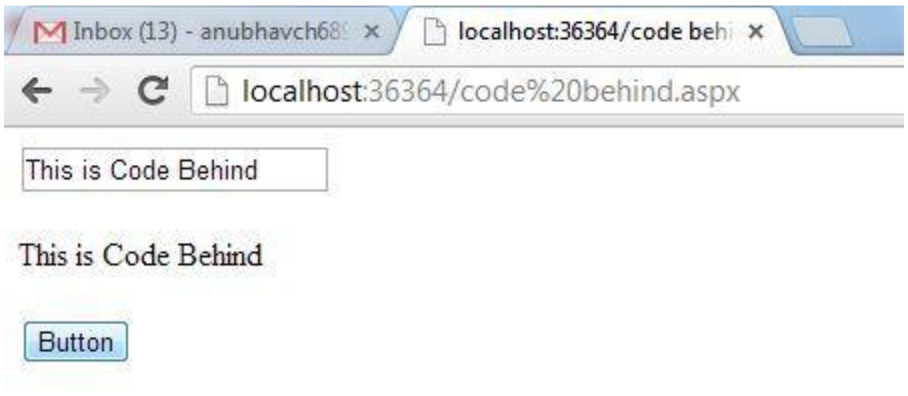
Step 2

Now double-click on the Button, this will use the Code window. Now you will see that this **coding section is opened in a separate window whose extension is .aspx.cs**. Write the code in this window. I wrote the code such that whatever I wrote in the TextBox will also appear in the Label.


localhost_36364 - Microsoft Visual Studio
FILE EDIT VIEW WEBSITE BUILD DEBUG TEAM SQL TOOLS TEST
Google Chrome
Debug
Server Explorer
Toolbox
code behind.aspx.cs*
code behind.aspx*
code_behind

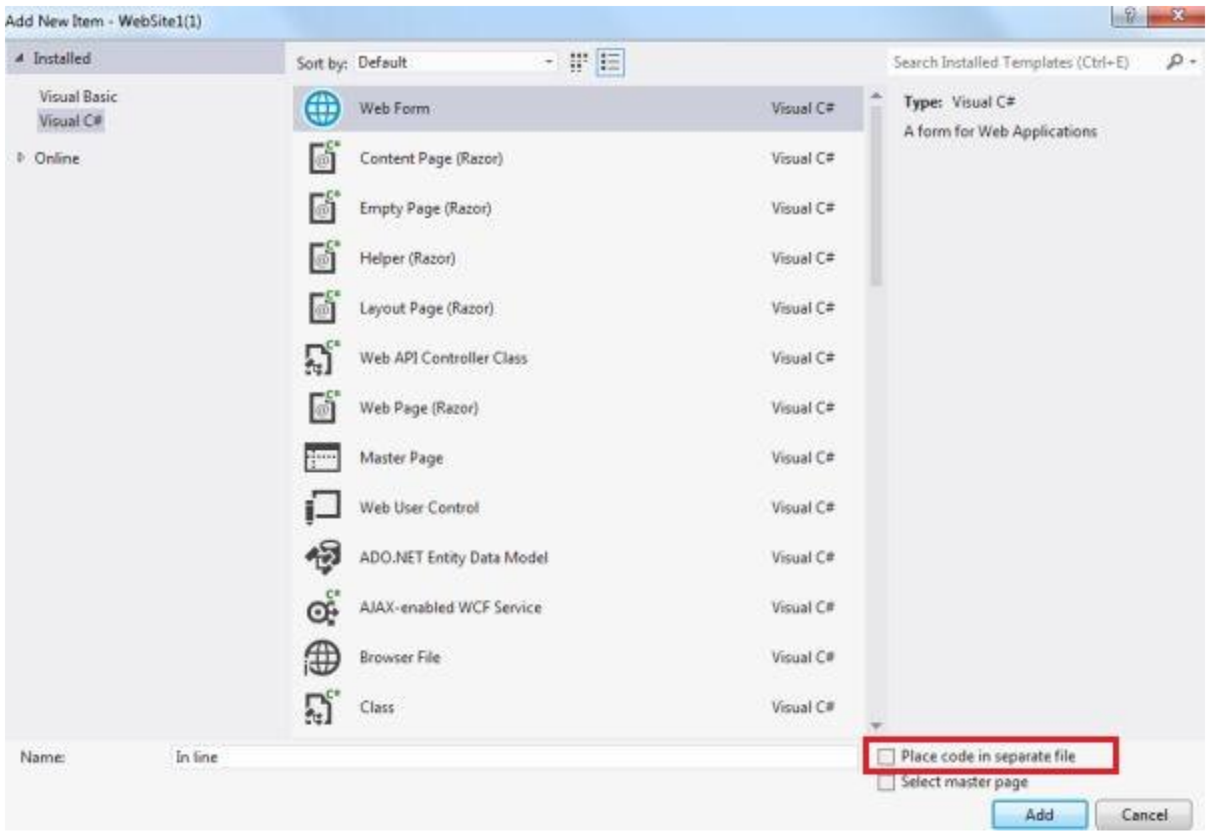
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

public partial class code_behind : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {

    }
    protected void Button1_Click(object sender, EventArgs e)
    {
        Label1.Text = TextBox1.Text;
    }
}
```

Now debug this page and verify that your program is running.

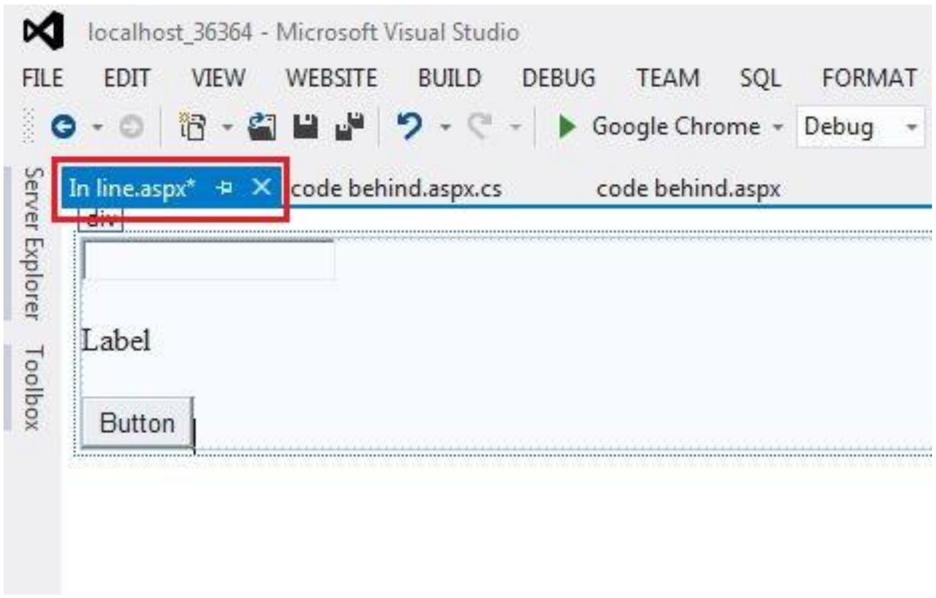


Step 3

Until now we were working on the Code Behind but **now we will work on the Inline Code**, for that add another web page to your Web Site. But this time things are different, this time **don't check the check box and if it's checked then Uncheck it and then click on "Add".**

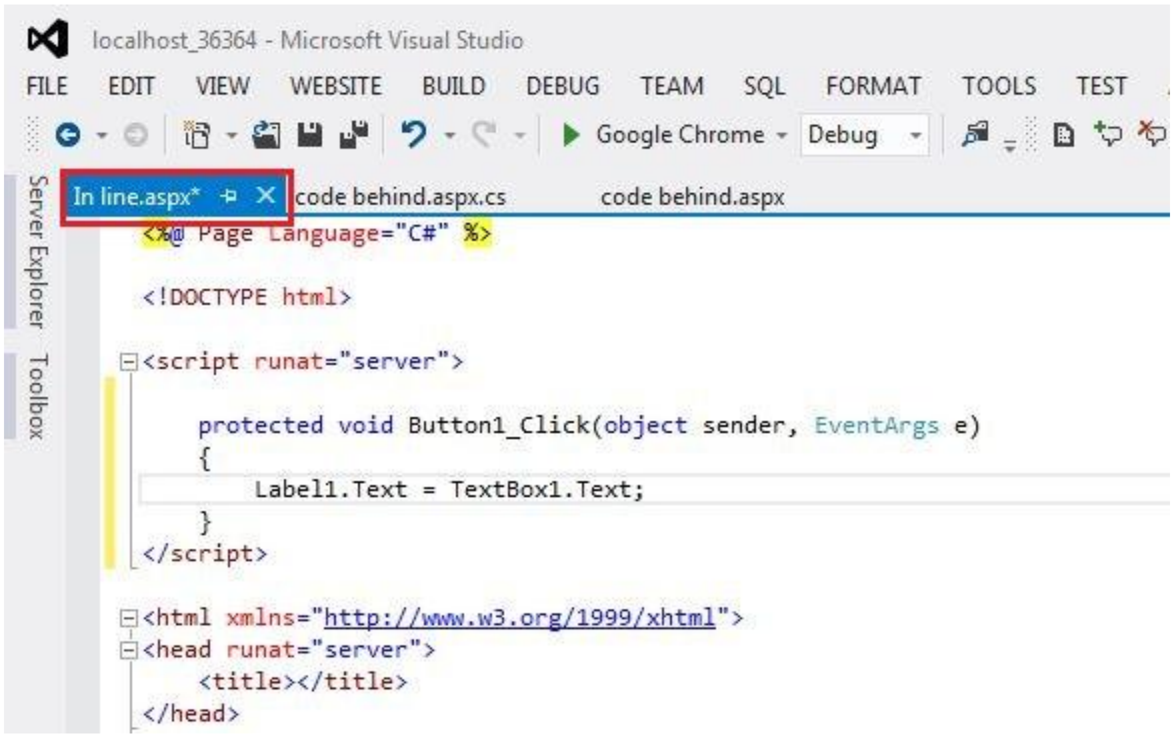


Now on this new .aspx page again use a Button, a Link, and a Text Box.



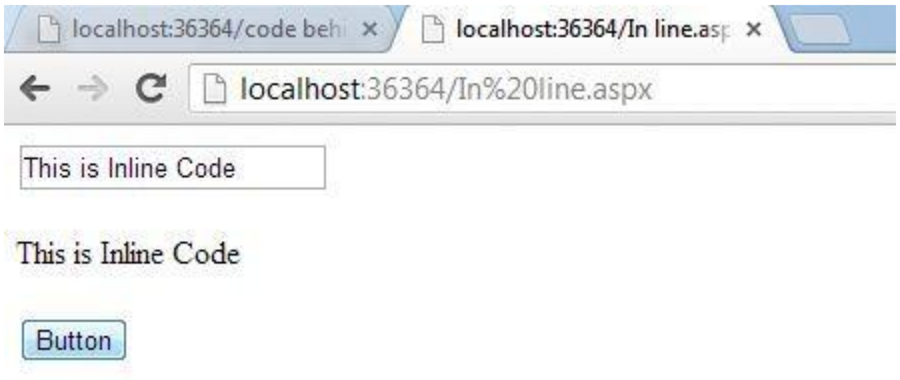
Step 4

Now double-click on the Button so that you can write the code on click of this button. But Now **No New Window will be opened; now the coding will appear on the same .aspx page**. **Here no .aspx.cs page is available.**


localhost_36364 - Microsoft Visual Studio
FILE EDIT VIEW WEBSITE BUILD DEBUG TEAM SQL FORMAT TOOLS TEST
Google Chrome
Debug
In line.aspx*
code behind.aspx.cs
code behind.aspx
Server Explorer
Toolbox

```
<%@ Page Language="C#" %>

<!DOCTYPE html>

<script runat="server">

    protected void Button1_Click(object sender, EventArgs e)
    {
        Label1.Text = TextBox1.Text;
    }
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
```

Write the same code here also and then debug it.


localhost:36364/code behi
localhost:36364/In line.asp
localhost:36364/In%20line.aspx
Button

This is Inline Code

This is Inline Code

As you can see it's giving the same output as the Code Behind gave but even after that is different from the Code Behind.

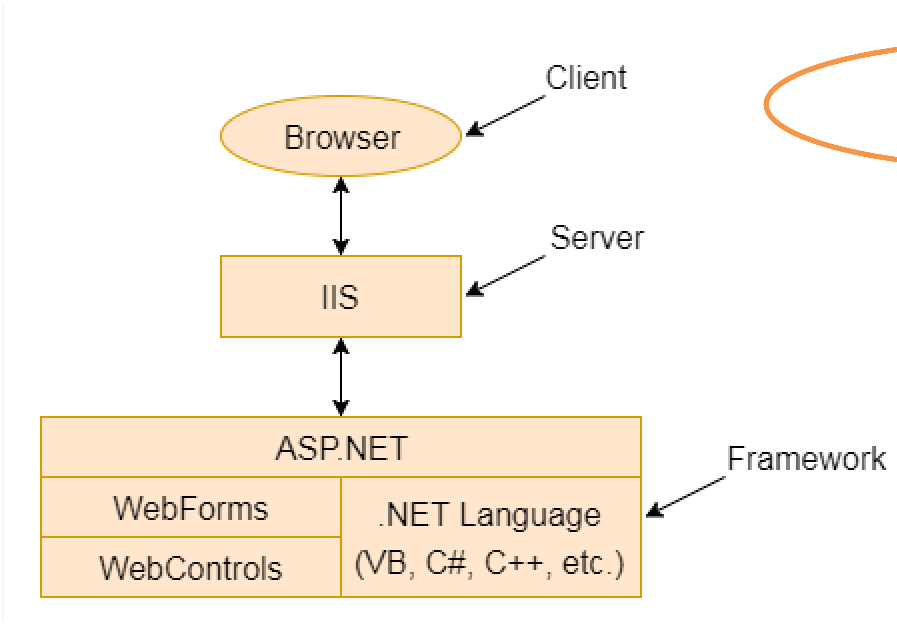
****** END OF UNIT- I ******

ASP.NET Web Forms

Web Forms are web pages built on the ASP.NET Technology. It executes on the server and generates output to the browser. It is compatible to any browser to any language supported by .NET common language runtime. It is flexible and allows us to create and add custom controls.

We can use Visual Studio to create ASP.NET Web Forms. It is an IDE (Integrated Development Environment) that allows us to drag and drop server controls to the web forms. It also allows us to set properties, events and methods for the controls. To write business logic, we can choose any .NET language like: Visual Basic or Visual C#.

Web Forms are made up of two components: the visual portion (the ASPX file), and the code behind the form, which resides in a separate class file.



FAQ: EXPLAIN THE WEB FORMS IN DETAIL

Fig: This diagram shows the components of the ASP.NET

The main purpose of Web Forms is to overcome the limitations of ASP and separate view from the application logic.

ASP.NET provides various controls like: server controls and HTML controls for the Web Forms. We have tables all these controls below.

FAQ: DESCRIBE SERVER- SIDE CONTROLS

Server Controls

The following table contains the server-side controls for the Web Forms.

Control Name	Applicable Events	Description
Label	None	It is used to display text on the HTML page.
TextBox	TextChanged	It is used to create a text input in the form.
Button	Click, Command	It is used to create a button.
LinkButton	Click, Command	It is used to create a button that looks similar to the hyperlink.
ImageButton	Click	It is used to create an imagesButton. Here, an image works as a Button.
Hyperlink	None	It is used to create a hyperlink control that responds to a click event.
DropDownList	SelectedIndexChanged	It is used to create a dropdown list control.

ListBox	SelectedIndexCnhaged	It is used to create a ListBox control like the HTML control.
DataGrid	CancelCommand, EditCommand, DeleteCommand, ItemCommand, SelectedIndexChanged, PageIndexChanged, SortCommand, UpdateCommand, ItemCreated, ItemDataBound	It used to create a frid that is used to show data. We can also perform paging, sorting, and formatting very easily with this control.
DataList	CancelCommand, EditCommand, DeleteCommand, ItemCommand, SelectedIndexChanged, UpdateCommand, ItemCreated, ItemDataBound	It is used to create datalist that is non-tabular and used to show data.
Repeater	ItemCommand, ItemCreated, ItemDataBound	It allows us to create a non-tabular type of format for data. You can bind the data to template items, which are like bits of HTML put together in a specific repeating format.
CheckBox	CheckChanged	It is used to create checkbox.

CheckBoxList	SelectedIndexChanged	It is used to create a group of check boxes that all work together.
RadioButton	CheckChanged	It is used to create radio button.
RadioButtonList	SelectedIndexChanged	It is used to create a group of radio button controls that all work together.
Image	None	It is used to show image within the page.
Panel	None	It is used to create a panel that works as a container.
PlaceHolder	None	It is used to set placeholder for the control.
Calendar	SelectionChanged, VisibleMonthChanged, DayRender	It is used to create a calendar. We can set the default date, move forward and backward etc.
AdRotator	AdCreated	It allows us to specify a list of ads to display. Each time the user re-displays the page.

Table	None	It is used to create table.
XML	None	It is used to display XML documents within the HTML.
Literal	None	It is like a label in that it displays a literal, but allows us to create new literals at runtime and place them into this control.

FAQ: DISCUSS THE HTML CONTROLS IN DETAIL

HTML Controls

These controls render by the browser. We can also make HTML controls as server control. we will discuss about this in further our tutorial.

Controls Name	**Description**
Button	It is used to create HTML button.
Reset Button	Resets all other HTML form elements on a form to a default value
Submit Button	Automatically POSTs the form data to the specified page listed in the Action attribute in the FORM tag

Text Field	Gives the user an input area on an HTML form
Text Area	Used for multi-line input on an HTML form
File Field	Places a text field and a Browse button on a form and allows the user to select a file name from their local machine when the Browse button is clicked
Password Field	An input area on an HTML form, although any characters typed into this field are displayed as asterisks
CheckBox	Gives the user a check box that they can select or clear
Radio Button	Used two or more to a form, and allows the user to choose one of the controls
Table	Allows you to present information in a tabular format
Image	Displays an image on an HTML form
ListBox	Displays a list of items to the user. You can set the size from two or more to specify how many items you wish show. If there are more items than will fit within this limit, a scroll bar is automatically added to this control.
Dropdown	Displays a list of items to the user, but only one item at a time will appear. The user can click a down arrow from the side of this control and a list of items will be displayed.
Horizontal Rule	Displays a horizontal line across the HTML page

WEB CONTROLS

ASP.NET uses five types of web controls, which are:

- HTML controls
- HTML Server controls
- ASP.NET Server controls
- ASP.NET Ajax Server controls
- User controls and custom controls
-

ASP.NET server controls are the primary controls used in ASP.NET. These controls can be grouped into the following categories:

- **Validation controls** - These are used to validate user input and they work by running client-side script.
- **Data source controls** - These controls provides data binding to different data sources.
- **Data view controls** - These are various lists and tables, which can bind to data from data sources for displaying.
- **Personalization controls** - These are used for personalization of a page according to the user preferences, based on user information.
- **Login and security controls** - These controls provide user authentication.
- **Master pages** - These controls provide consistent layout and interface throughout the application.
- **Navigation controls** - These controls help in navigation. For example, menus, tree view etc.
- **Rich controls** - These controls implement special features. For example, AdRotator, FileUpload, and Calendar control.

The syntax for using server controls is:

```
<asp:controlType  ID ="ControlID" runat="server" Property1=value1  [Property2=value2] />
```

In addition, visual studio has the following features, to help produce in error-free coding:

- Dragging and dropping of controls in design view
- IntelliSense feature that displays and auto-completes the properties
- The properties window to set the property values directly

Properties of the Server Controls

ASP.NET server controls with a visual aspect are derived from the WebControl class and inherit all the properties, events, and methods of this class.

The WebControl class itself and some other server controls that are not visually rendered are derived from the System.Web.UI.Control class. For example, PlaceHolder control or XML control.

ASP.Net server controls inherit all properties, events, and methods of the WebControl and System.Web.UI.Control class.

The following table shows the inherited properties, common to all server controls:

Property	Description
AccessKey	Pressing this key with the Alt key moves focus to the control.
Attributes	It is the collection of arbitrary attributes (for rendering only) that do not correspond to properties on the control.
BackColor	Background color.
BindingContainer	The control that contains this control's data binding.
BorderColor	Border color.
BorderStyle	Border style.
BorderWidth	Border width.
CausesValidation	Indicates if it causes validation.
ChildControlCreated	It indicates whether the server control's child controls have been created.
ClientID	Control ID for HTML markup.

Context	The HttpContext object associated with the server control.
Controls	Collection of all controls contained within the control.
ControlStyle	The style of the Web server control.
CssClass	CSS class
DataItemContainer	Gets a reference to the naming container if the naming container implements IDataItemContainer.
DataKeysContainer	Gets a reference to the naming container if the naming container implements IDataKeysControl.
DesignMode	It indicates whether the control is being used on a design surface.
DisabledCssClass	Gets or sets the CSS class to apply to the rendered HTML element when the control is disabled.
Enabled	Indicates whether the control is grayed out.
EnableTheming	Indicates whether theming applies to the control.
EnableViewState	Indicates whether the view state of the control is maintained.
Events	Gets a list of event handler delegates for the control.
Font	Font.

Forecolor	Foreground color.
HasAttributes	Indicates whether the control has attributes set.
HasChildViewState	Indicates whether the current server control's child controls have any saved view-state settings.
Height	Height in pixels or %.
ID	Identifier for the control.
IsChildControlStateCleared	Indicates whether controls contained within this control have control state.
IsEnabled	Gets a value indicating whether the control is enabled.
IsTrackingViewState	It indicates whether the server control is saving changes to its view state.
IsViewStateEnabled	It indicates whether view state is enabled for this control.
LoadViewStateById	It indicates whether the control participates in loading its view state by ID instead of index.
Page	Page containing the control.
Parent	Parent control.
RenderingCompatibility	It specifies the ASP.NET version that the rendered HTML will be compatible with.

Site	The container that hosts the current control when rendered on a design surface.
SkinID	Gets or sets the skin to apply to the control.
Style	Gets a collection of text attributes that will be rendered as a style attribute on the outer tag of the Web server control.
TabIndex	Gets or sets the tab index of the Web server control.
TagKey	Gets the HtmlTextWriterTag value that corresponds to this Web server control.
TagName	Gets the name of the control tag.
TemplateControl	The template that contains this control.
TemplateSourceDirectory	Gets the virtual directory of the page or control containing this control.
ToolTip	Gets or sets the text displayed when the mouse pointer hovers over the web server control.
UniqueID	Unique identifier.
ViewState	Gets a dictionary of state information that saves and restores the view state of a server control across multiple requests for the same page.
ViewStateIgnoreCase	It indicates whether the StateBag object is case-insensitive.
ViewStateMode	Gets or sets the view-state mode of this control.

Visible	It indicates whether a server control is visible.
Width	Gets or sets the width of the Web server control.

Methods of the Server Controls

The following table provides the methods of the server controls:

Method	Description
AddAttributesToRender	Adds HTML attributes and styles that need to be rendered to the specified HtmlTextWriterTag.
AddedControl	Called after a child control is added to the Controls collection of the control object.
AddParsedSubObject	Notifies the server control that an element, either XML or HTML, was parsed, and adds the element to the server control's control collection.
ApplyStyleSheetSkin	Applies the style properties defined in the page style sheet to the control.
ClearCachedClientID	Infrastructure. Sets the cached ClientID value to null.
ClearChildControlState	Deletes the control-state information for the server control's child controls.
ClearChildState	Deletes the view-state and control-state information for all the server control's child controls.
ClearChildViewState	Deletes the view-state information for all the server control's child controls.

CreateChildControls	Used in creating child controls.
CreateControlCollection	Creates a new ControlCollection object to hold the child controls.
CreateControlStyle	Creates the style object that is used to implement all style related properties.
DataBind	Binds a data source to the server control and all its child controls.
DataBind(Boolean)	Binds a data source to the server control and all its child controls with an option to raise the DataBinding event.
DataBindChildren	Binds a data source to the server control's child controls.
Dispose	Enables a server control to perform final clean up before it is released from memory.
EnsureChildControls	Determines whether the server control contains child controls. If it does not, it creates child controls.
EnsureID	Creates an identifier for controls that do not have an identifier.
Equals(Object)	Determines whether the specified object is equal to the current object.
Finalize	Allows an object to attempt to free resources and perform other cleanup operations before the object is reclaimed by garbage collection.
FindControl(String)	Searches the current naming container for a server control with the specified id parameter.

FindControl(String, Int32)	Searches the current naming container for a server control with the specified id and an integer.
Focus	Sets input focus to a control.
GetDesignModeState	Gets design-time data for a control.
GetType	Gets the type of the current instance.
GetUniqueIDRelativeTo	Returns the prefixed portion of the UniqueID property of the specified control.
HasControls	Determines if the server control contains any child controls.
HasEvents	Indicates whether events are registered for the control or any child controls.
IsLiteralContent	Determines if the server control holds only literal content.
LoadControlState	Restores control-state information.
LoadViewState	Restores view-state information.
MapPathSecure	Retrieves the physical path that a virtual path, either absolute or relative, maps to.
MemberwiseClone	Creates a shallow copy of the current object.
MergeStyle	Copies any nonblank elements of the specified style to the web control, but does not overwrite any existing style elements of the control.

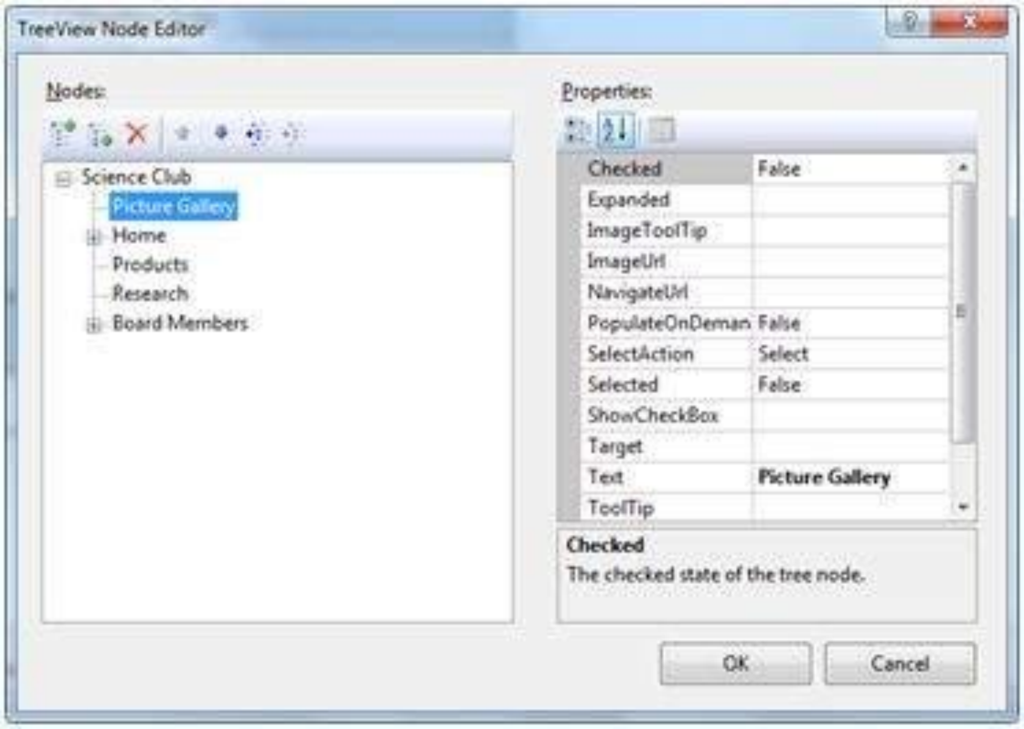
OnBubbleEvent	Determines whether the event for the server control is passed up the page's UI server control hierarchy.
OnDataBinding	Raises the data binding event.
OnInit	Raises the Init event.
OnLoad	Raises the Load event.
OnPreRender	Raises the PreRender event.
OnUnload	Raises the Unload event.
OpenFile	Gets a Stream used to read a file.
RemovedControl	Called after a child control is removed from the controls collection of the control object.
Render	Renders the control to the specified HTML writer.
RenderBeginTag	Renders the HTML opening tag of the control to the specified writer.
RenderChildren	Outputs the contents of a server control's children to a provided HtmlTextWriter object, which writes the contents to be rendered on the client.
RenderContents	Renders the contents of the control to the specified writer.
RenderControl(HtmlTextWriter)	Outputs server control content to a provided HtmlTextWriter object and stores tracing information about the control if tracing is enabled.

RenderEndTag	Renders the HTML closing tag of the control into the specified writer.
ResolveAdapter	Gets the control adapter responsible for rendering the specified control.
SaveControlState	Saves any server control state changes that have occurred since the time the page was posted back to the server.
SaveViewState	Saves any state that was modified after the TrackViewState method was invoked.
SetDesignModeState	Sets design-time data for a control.
ToString	Returns a string that represents the current object.
TrackViewState	Causes the control to track changes to its view state so that they can be stored in the object's view state property.

Example

Let us look at a particular server control - a tree view control. A Tree view control comes under navigation controls. Other Navigation controls are: Menu control and SiteMapPath control.

Add a tree view control on the page. Select Edit Nodes... from the tasks. Edit each of the nodes using the Tree view node editor as shown:

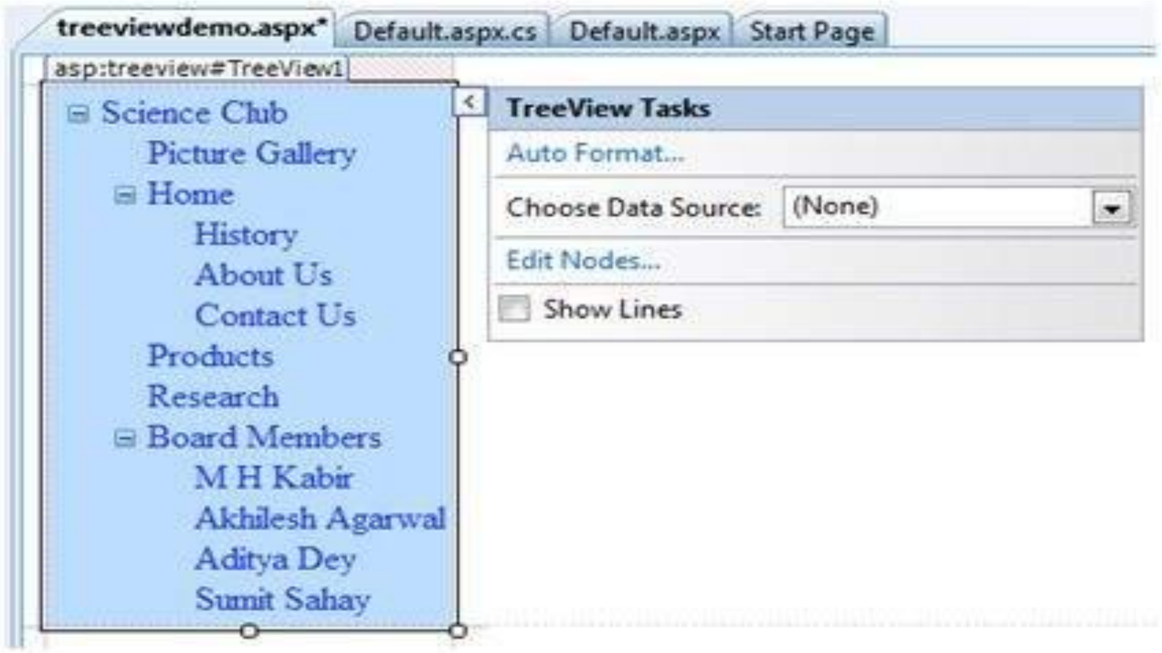


Science Club
- Picture Gallery
- Home
- Products
- Research
- Board Members

Checked	False
Expanded	
ImageToolTip	
ImageUrl	
NavigateUrl	
PopulateOnDeman	False
SelectAction	Select
Selected	False
ShowCheckBox	
Target	
Text	Picture Gallery
ToolTip	

Checked
The checked state of the tree node.

Once you have created the nodes, it looks like the following in design view:



Choose Data Source: (None)

Science Club
- Picture Gallery
- Home
  - History
  - About Us
  - Contact Us
- Products
- Research
- Board Members
  - M H Kabir
  - Akhilesh Agarwal
  - Aditya Dey
  - Sumit Sahay

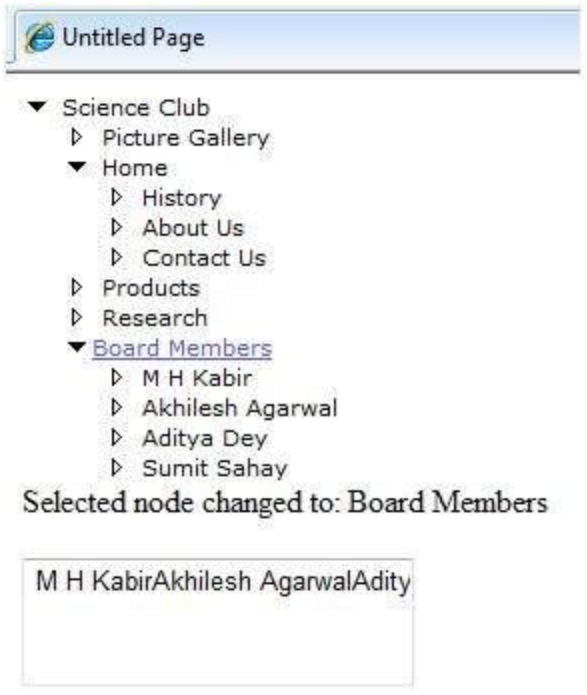
The AutoFormat... task allows you to format the tree view as shown:

Add a label control and a text box control on the page and name them lblmessage and txtmessage respectively.

Write a few lines of code to ensure that when a particular node is selected, the label control displays the node text and the text box displays all child nodes under it, if any. The code behind the file should look like this:

```
using System;
using System.Collections;
using System.Configuration;
using System.Data;
using System.Linq;

using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;

using System.Xml.Linq;

namespace eventdemo {
  public partial class treeviewdemo : System.Web.UI.Page {

    protected void Page_Load(object sender, EventArgs e) {
      txtmessage.Text = " ";
    }

    protected void TreeView1_SelectedNodeChanged(object sender, EventArgs e) {

      txtmessage.Text = " ";
      lblmessage.Text = "Selected node changed to: " + TreeView1.SelectedNode.Text;
      TreeNodeCollection childnodes = TreeView1.SelectedNode.ChildNodes;

      if(childnodes != null) {
        txtmessage.Text = " ";

        foreach (TreeNode t in childnodes) {
          txtmessage.Text += t.Value;
        }
      }
    }
  }
}
```

Execute the page to see the effects. You will be able to expand and collapse the nodes.



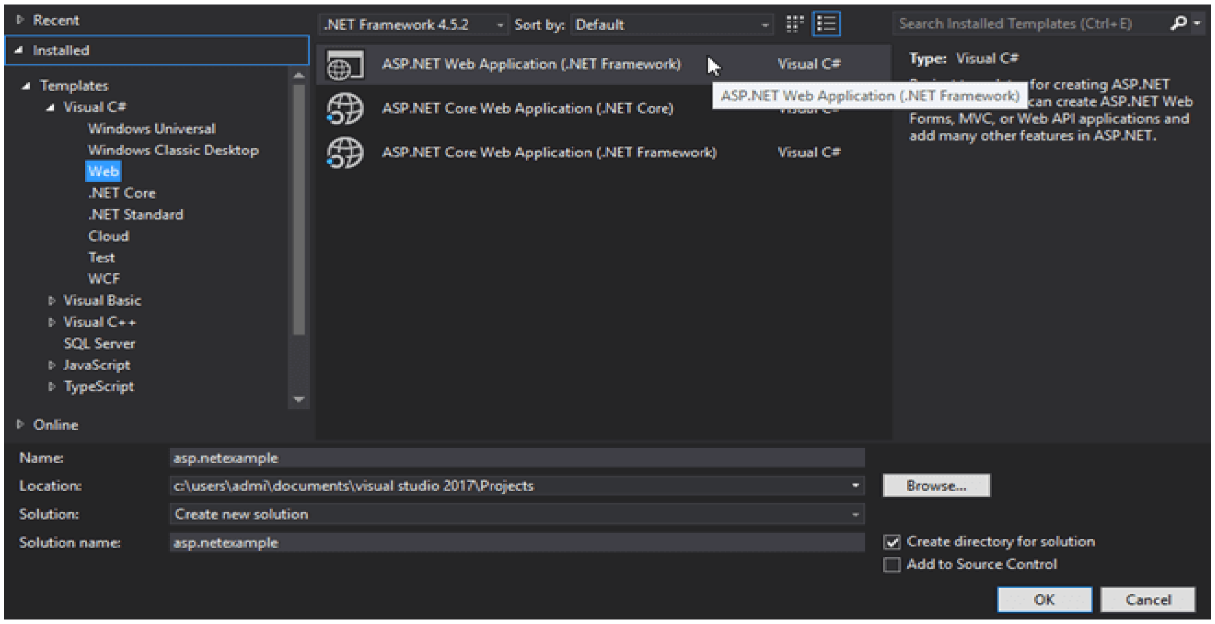
```
▼ Science Club
    ▷ Picture Gallery
    ▼ Home
        ▷ History
        ▷ About Us
        ▷ Contact Us
    ▷ Products
    ▷ Research
    ▼ Board Members
        ▷ M H Kabir
        ▷ Akhilesh Agarwal
        ▷ Aditya Dey
        ▷ Sumit Sahay
Selected node changed to: Board Members

M H KabirAkhilesh AgarwalAdity
```

ASP.NET Web Forms Project

We are using Visual studio 2017 to create web project. It includes the following steps:

1. Creating a new project

   Click on the file menu from the menu bar and select **new -> project.**



Type: Visual C#

for creating ASP.NET can create ASP.NET Web Forms, MVC, or Web API applications and add many other features in ASP.NET.

Name: asp.netexample

Location: c:\users\admi\documents\visual studio 2017\Projects

Solution: Create new solution

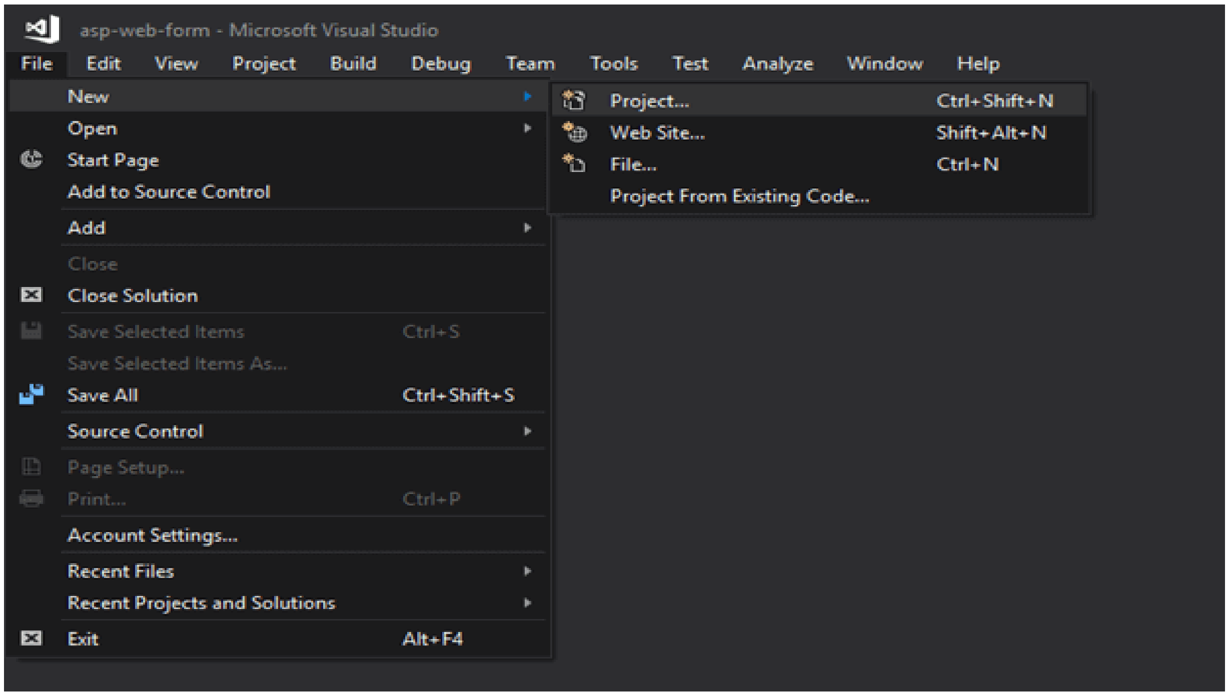
Solution name: asp.netexample

[✓] Create directory for solution

[ ] Add to Source Control

Select Project type

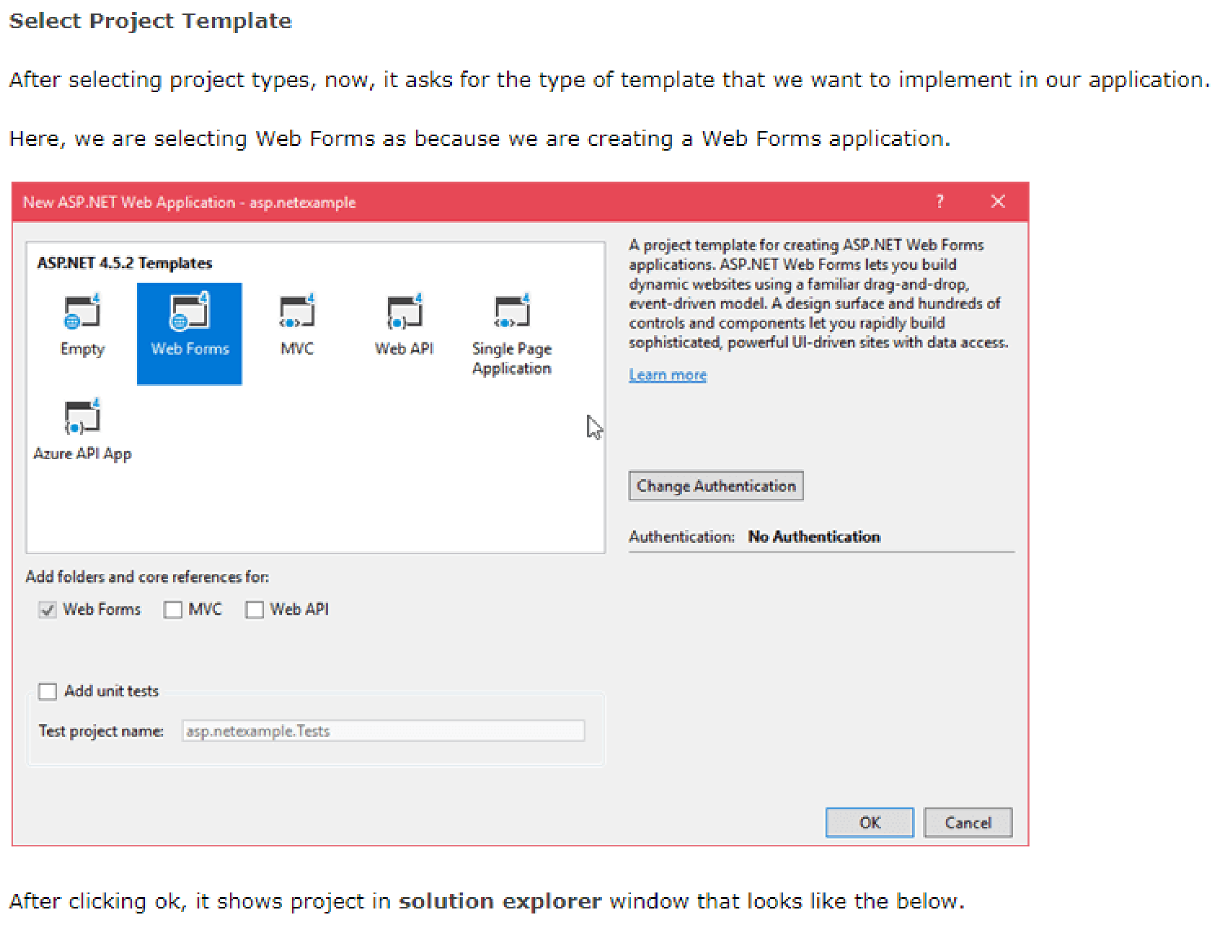
It provides couple of choices but we selecting ASP.NET Web Application.



Select Project Template

After selecting project types, now, it asks for the type of template that we want to implement in our application.

Here, we are selecting Web Forms as because we are creating a Web Forms application.



Select Project Template

After selecting project types, now, it asks for the type of template that we want to implement in our application.

Here, we are selecting Web Forms as because we are creating a Web Forms application.

New ASP.NET Web Application - asp.netexample

ASP.NET 4.5.2 Templates

A project template for creating ASP.NET Web Forms applications. ASP.NET Web Forms lets you build dynamic websites using a familiar drag-and-drop, event-driven model. A design surface and hundreds of controls and components let you rapidly build sophisticated, powerful UI-driven sites with data access.

Learn more

Change Authentication

Authentication: **No Authentication**

Add folders and core references for:

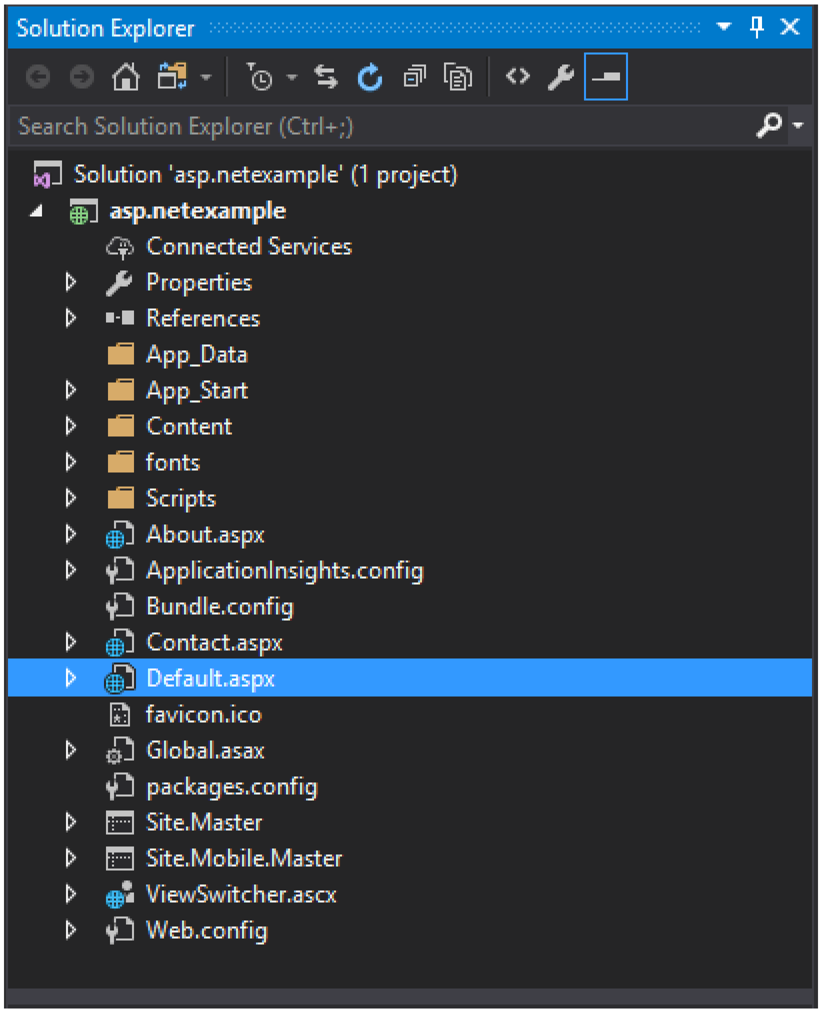
[✓] Web Forms [ ] MVC [ ] Web API

[ ] Add unit tests

Test project name: asp.netexample.Tests

After clicking ok, it shows project in **solution explorer** window that looks like the below.

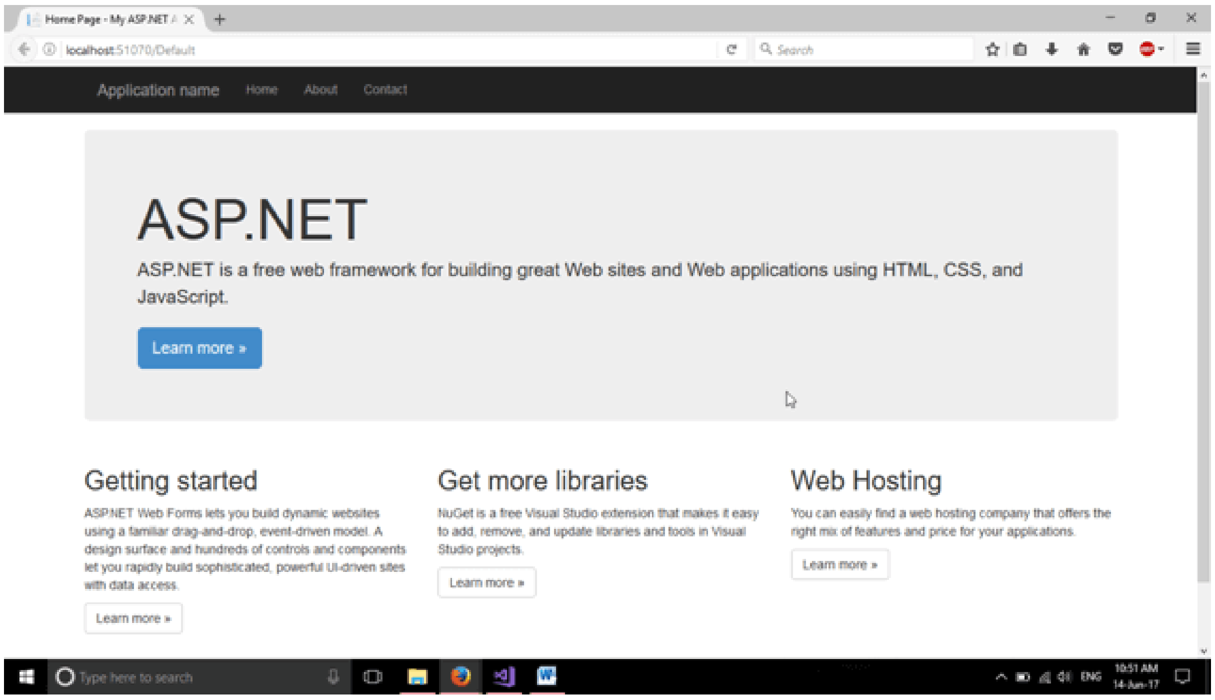
After clicking ok, it shows project in **solution explorer** window that looks like the below.



This project contains a **default.aspx** file which is a startup file. When we run the project this file executes first and display a home page of the site.

We can see its output on the browser by selecting **view in browser** option as we did below.

Finally, it shows output in the browser like this:



ASP.NET

ASP.NET is a free web framework for building great Web sites and Web applications using HTML, CSS, and JavaScript.

Getting started

ASP.NET Web Forms lets you build dynamic websites using a familiar drag-and-drop, event-driven model. A design surface and hundreds of controls and components let you rapidly build sophisticated, powerful UI-driven sites with data access.

Get more libraries

NuGet is a free Visual Studio extension that makes it easy to add, remove, and update libraries and tools in Visual Studio projects.

Web Hosting

You can easily find a web hosting company that offers the right mix of features and price for your applications.

Well, we have created a project successfully and running on the browser.

In next chapter, we will create a new web form and link that within project.

An ASP.NET page is made up of a number of server controls along with HTML controls, text, and images. Sensitive data from the page and the states of different controls on the page are stored in hidden fields that form the context of that page request.

ASP.NET runtime controls the association between a page instance and its state. An ASP.NET page is an object of the Page or inherited from it.

All the controls on the pages are also objects of the related control class inherited from a parent Control class. When a page is run, an instance of the object page is created along with all its content controls.

An ASP.NET page is also a server side file saved with the .aspx extension. It is modular in nature and can be divided into the following core sections:

- Page Directives
- Code Section
- Page Layout

Page Directives

The page directives set up the environment for the page to run. The @Page directive defines page-specific attributes used by ASP.NET page parser and compiler. Page directives specify how the page should be processed, and which assumptions need to be taken about the page.

It allows importing namespaces, loading assemblies, and registering new controls with custom tag names and namespace prefixes.

Code Section

The code section provides the handlers for the page and control events along with other functions required. We mentioned that, ASP.NET follows an object model. Now, these objects raise events when some events take place on the user interface, like a user clicks a button or moves the cursor. The kind of response these events need to reciprocate is coded in the event handler functions. The event handlers are nothing but functions bound to the controls.

The code section or the code behind file provides all these event handler routines, and other functions used by the developer. The page code could be precompiled and deployed in the form of a binary assembly.

Page Layout

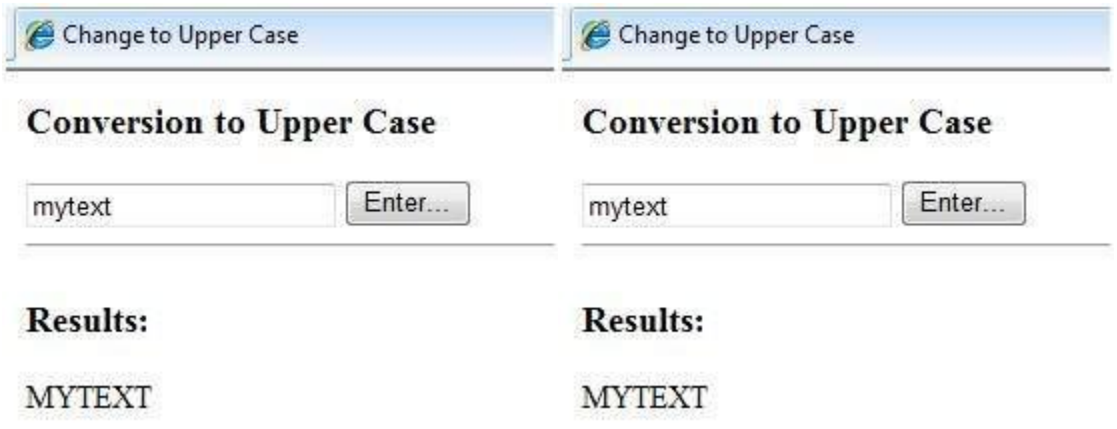
The page layout provides the interface of the page. It contains the server controls, text, inline JavaScript, and HTML tags.

The following code snippet provides a sample ASP.NET page explaining Page directives, code section and page layout written in C#:

```
<!-- directives -->
<% @Page Language="C#" %>

<!-- code section -->
<script runat="server">

  private void convertoupper(object sender, EventArgs e)
  {
     string str = mytext.Value;
     changed_text.InnerHtml = str.ToUpper();
  }
</script>

<!-- Layout -->
<html>
  <head>
```

```
    <title> Change to Upper Case </title>
  </head>

  <body>
    <h3> Conversion to Upper Case </h3>

    <form runat="server">
      <input runat="server" id="mytext" type="text" />
      <input runat="server" id="button1" type="submit" value="Enter..."
OnServerClick="convertoupper"/>

      <hr />
      <h3> Results: </h3>
      <span runat="server" id="changed_text" />
    </form>

  </body>

</html>
```

Copy this file to the web server root directory. Generally it is c:\iNETput\wwwroot. Open the file from the browser to execute it and it generates following result:



Conversion to Upper Case

mytext

Results:

MYTEXT

Conversion to Upper Case

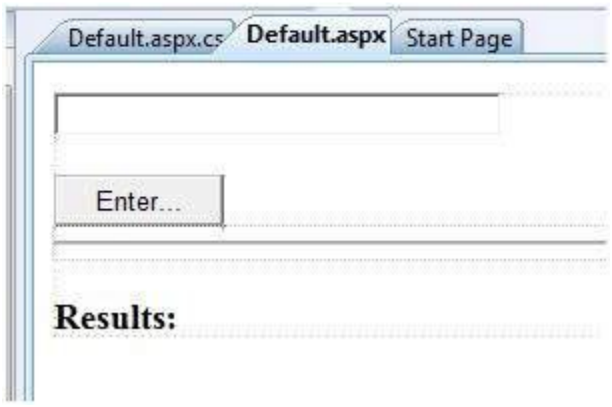
mytext

Results:

MYTEXT

Using Visual Studio IDE

Let us develop the same example using Visual Studio IDE. Instead of typing the code, you can just drag the controls into the design view:



The content file is automatically developed. All you need to add is the Button1_Click routine, which is as follows:

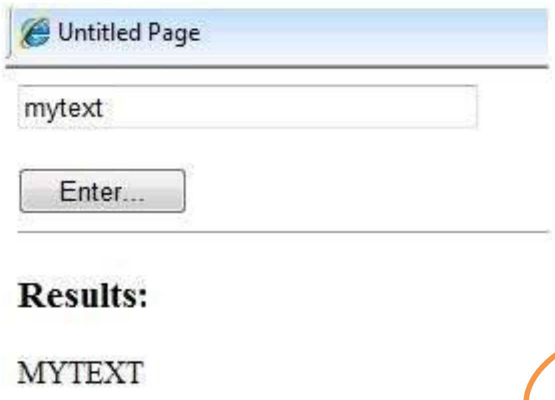
```
protected void Button1_Click(object sender, EventArgs e)
{
  string buf = TextBox1.Text;
  changed_text.InnerHtml = buf.ToUpper();
}
```

The content file code is as given:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
  Inherits="firstexample._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>
      Untitled Page
    </title>
  </head>

  <body>

    <form id="form1" runat="server">
      <div>

        <asp:TextBox ID="TextBox1" runat="server" style="width:224px">
```

```
         </asp:TextBox>

         <br />
         <br />

         <asp:Button ID="Button1" runat="server" Text="Enter..." style="width:85px"
onclick="Button1_Click" />
         <hr />

         <h3> Results: </h3>
         <span runat="server" id="changed_text" />

       </div>
     </form>

   </body>

</html>
```

Execute the example by right clicking on the design view and choosing 'View in Browser' from the popup menu. This generates the following result:



Results:

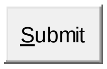
MYTEXT

FAQ: WHAT IS AUTOPOSTBACK IN ASP.NET?

AutoPostback in ASP.NET

- AutoPostback or Postback is nothing but submitting page to server.
- AutoPostback is webpage going to server, Server processes the values and sends back to same page or redirects to different page.

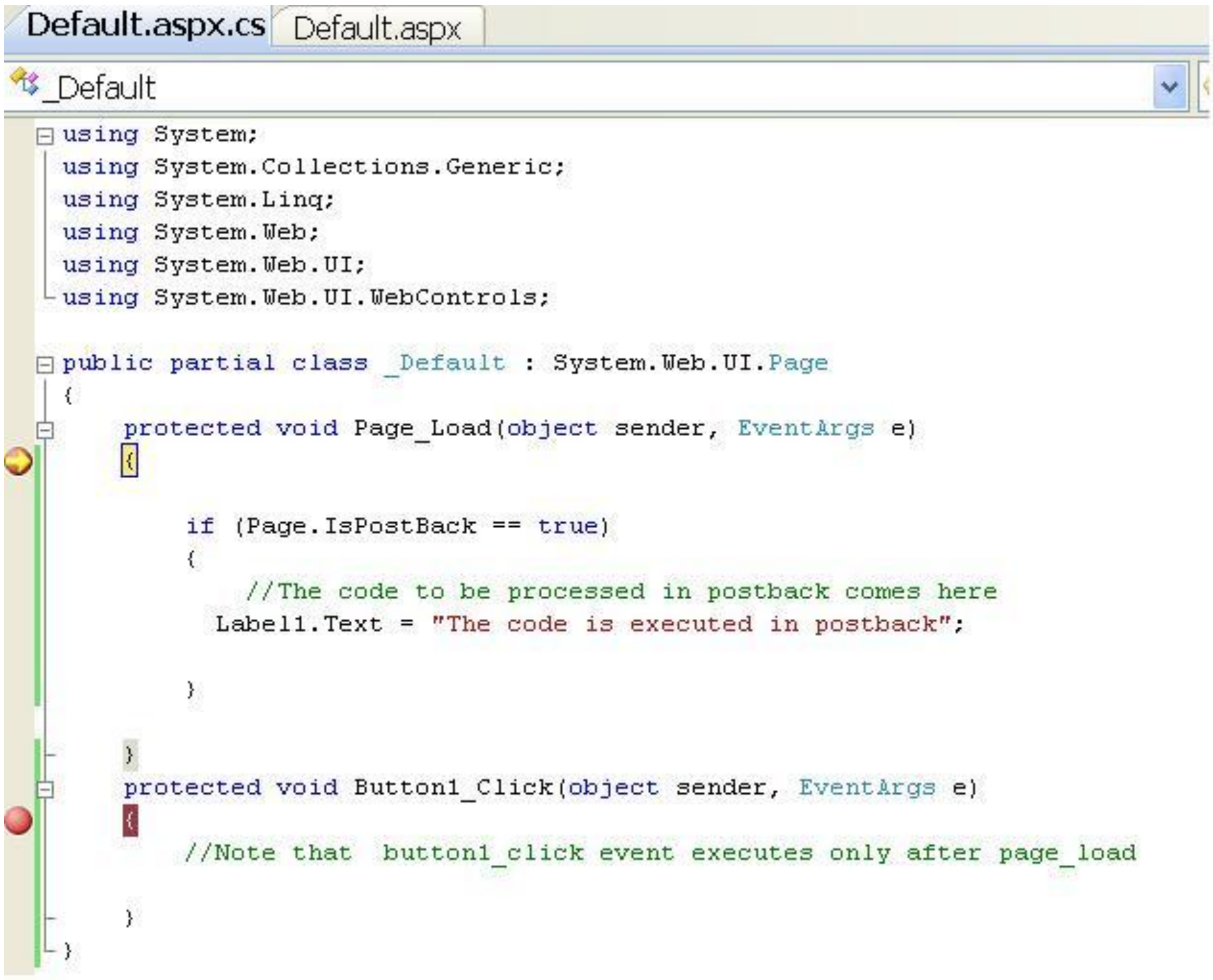
Example -1:



Click on the above button and observe the progressbar of browser.

Just Postback happened

When you click on above button, the request goes to server below. The server processes the request(The code executes) and returns the output(response) to webpage.



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {

        if (Page.IsPostBack == true)
        {
            //The code to be processed in postback comes here
          Label1.Text = "The code is executed in postback";

        }

    }
    protected void Button1_Click(object sender, EventArgs e)
    {
        //Note that  button1_click event executes only after page_load

    }
}
```

In the above picture , you can observe what happens on server side. Page.IsPostback value will be true if it is postback.

EVENT HANDLING IN ASP.NET

FAQ: EXPLAIN EVENT HANDLING IN ASP.NET WITH EXAMPLES

An event is an action or occurrence such as a mouse click, a key press, mouse movements, or any system-generated notification. A process communicates through events. For example, interrupts are system-generated events. When events occur, the application should be able to respond to it and manage it.

Events in ASP.NET raised at the client machine, and handled at the server machine. For example, a user clicks a button displayed in the browser. A Click event is raised. The browser handles this client-side event by posting it to the server.

The server has a subroutine describing what to do when the event is raised; it is called the event-handler. Therefore, when the event message is transmitted to the server, it checks whether the Click event has an associated event handler. If it has, the event handler is executed.

Event Arguments

ASP.NET event handlers generally take two parameters and return void. The first parameter represents the object raising the event and the second parameter is event argument.

The general syntax of an event is:

```
private void EventName (object sender, EventArgs e);
```

Application and Session Events

The most important application events are:

- **Application_Start** - It is raised when the application/website is started.
- **Application_End** - It is raised when the application/website is stopped.

Similarly, the most used Session events are:

- **Session_Start** - It is raised when a user first requests a page from the application.
- **Session_End** - It is raised when the session ends.

Page and Control Events

Common page and control events are:

- **DataBinding** - It is raised when a control binds to a data source.
- **Disposed** - It is raised when the page or the control is released.
- **Error** - It is a page event, occurs when an unhandled exception is thrown.
- **Init** - It is raised when the page or the control is initialized.
- **Load** - It is raised when the page or a control is loaded.
- **PreRender** - It is raised when the page or the control is to be rendered.
- **Unload** - It is raised when the page or control is unloaded from memory.

Event Handling Using Controls

All ASP.NET controls are implemented as classes, and they have events which are fired when a user performs a certain action on them. For example, when a user clicks a button the 'Click' event is generated. For handling events, there are in-built attributes and event handlers. Event handler is coded to respond to an event, and take appropriate action on it.

By default, Visual Studio creates an event handler by including a Handles clause on the Sub procedure. This clause names the control and event that the procedure handles.

The ASP tag for a button control:

```
<asp:Button ID="btnCancel" runat="server" Text="Cancel" />
```

The event handler for the Click event:

```
Protected Sub btnCancel_Click(ByVal sender As Object, ByVal e As System.EventArgs)

  Handles btnCancel.Click

End Sub
```

An event can also be coded without Handles clause. Then, the handler must be named according to the appropriate event attribute of the control.

The ASP tag for a button control:

```
<asp:Button ID="btnCancel" runat="server" Text="Cancel" Onclick="btnCancel_Click" />
```

The event handler for the Click event:

```
Protected Sub btnCancel_Click(ByVal sender As Object, ByVal e As System.EventArgs)

End Sub
```

The common control events are:

Event	Attribute	Controls
Click	OnClick	Button, image button, link button, image map
Command	OnCommand	Button, image button, link button
TextChanged	OnTextChanged	Text box
SelectedIndexChanged	OnSelectedIndexChanged	Drop-down list, list box, radio button list, check box list.
CheckedChanged	OnCheckedChanged	Check box, radio button

Some events cause the form to be posted back to the server immediately, these are called the postback events. For example, the click event such as, Button.Click.

Some events are not posted back to the server immediately, these are called non-postback events.

For example, the change events or selection events such as TextBox.TextChanged or CheckBox.CheckedChanged. The nonpostback events could be made to post back immediately by setting their AutoPostBack property to true.

Default Events

The default event for the Page object is Load event. Similarly, every control has a default event. For example, default event for the button control is the Click event.

The default event handler could be created in Visual Studio, just by double clicking the control in design view. The following table shows some of the default events for common controls:

Control	Default Event
AdRotator	AdCreated

BulletedList	Click
Button	Click
Calender	SelectionChanged
CheckBox	CheckedChanged
CheckBoxList	SelectedIndexChanged
DataGrid	SelectedIndexChanged
DataList	SelectedIndexChanged
DropDownList	SelectedIndexChanged
HyperLink	Click
ImageButton	Click
ImageMap	Click
LinkButton	Click
ListBox	SelectedIndexChanged
Menu	MenuItemClick

RadioButton	CheckedChanged
RadioButtonList	SelectedIndexChanged

Example

This example includes a simple page with a label control and a button control on it. As the page events such as Page_Load, Page_Init, Page_PreRender etc. take place, it sends a message, which is displayed by the label control. When the button is clicked, the Button_Click event is raised and that also sends a message to be displayed on the label.

Create a new website and drag a label control and a button control on it from the control tool box. Using the properties window, set the IDs of the controls as .lblmessage. and .btnclick. respectively. Set the Text property of the Button control as 'Click'.

The markup file (.aspx):

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
  Inherits="eventdemo._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>Untitled Page</title>
  </head>

  <body>
    <form id="form1" runat="server">
      <div>
        <asp:Label ID="lblmessage" runat="server" >

        </asp:Label>

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

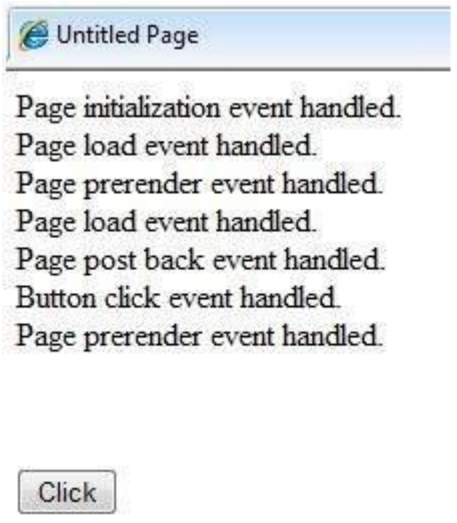
```
         <asp:Button ID="btnclick" runat="server" Text="Click" onclick="btnclick_Click" />
       </div>
     </form>
   </body>

</html>
```

Double click on the design view to move to the code behind file. The Page_Load event is automatically created without any code in it. Write down the following self-explanatory code lines:

```
using System;
using System.Collections;
using System.Configuration;
using System.Data;
using System.Linq;

using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;

using System.Xml.Linq;

namespace eventdemo {

   public partial class _Default : System.Web.UI.Page {

      protected void Page_Load(object sender, EventArgs e) {
         lblmessage.Text += "Page load event handled. <br />";

         if (Page.IsPostBack) {
            lblmessage.Text += "Page post back event handled.<br/>";
         }
      }

      protected void Page_Init(object sender, EventArgs e) {
         lblmessage.Text += "Page initialization event handled.<br/>";
      }
```

```

    protected void Page_PreRender(object sender, EventArgs e) {
      lblmessage.Text += "Page prerender event handled. <br/>";
    }

    protected void btnclick_Click(object sender, EventArgs e) {
      lblmessage.Text += "Button click event handled. <br/>";
    }
  }
}
```

Execute the page. The label shows page load, page initialization and, the page pre-render events. Click the button to see effect:



```
Page initialization event handled.
Page load event handled.
Page prerender event handled.
Page load event handled.
Page post back event handled.
Button click event handled.
Page prerender event handled.
```

FAQ: DISCUSS ANY FIVE ASP.NET BASIC CONTROLS.

BASIC CONTROLS OF ASP.NET

Button Controls

ASP.NET provides three types of button control:

- **Button** : It displays text within a rectangular area.
- **Link Button** : It displays text that looks like a hyperlink.
- **Image Button** : It displays an image.

When a user clicks a button, two events are raised: Click and Command.

Basic syntax of button control:

```
<asp:Button ID="Button1" runat="server" onclick="Button1_Click" Text="Click" / >
```

Common properties of the button control:

Property	Description
Text	The text displayed on the button. This is for button and link button controls only.
ImageUrl	For image button control only. The image to be displayed for the button.
AlternateText	For image button control only. The text to be displayed if the browser cannot display the image.
CausesValidation	Determines whether page validation occurs when a user clicks the button. The default is true.
CommandName	A string value that is passed to the command event when a user clicks the button.
CommandArgument	A string value that is passed to the command event when a user clicks the button.
PostBackUrl	The URL of the page that is requested when the user clicks the button.

Text Boxes and Labels

Text box controls are typically used to accept input from the user. A text box control can accept one or more lines of text depending upon the settings of the TextMode attribute.

Label controls provide an easy way to display text which can be changed from one execution of a page to the next. If you want to display text that does not change, you use the literal text.

Basic syntax of text control:

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

Common Properties of the Text Box and Labels:

Property	Description
TextMode	Specifies the type of text box. SingleLine creates a standard text box, MultiLIne creates a text box that accepts more than one line of text and the Password causes the characters that are entered to be masked. The default is SingleLine.
Text	The text content of the text box.
MaxLength	The maximum number of characters that can be entered into the text box.
Wrap	It determines whether or not text wraps automatically for multi-line text box; default is true.
ReadOnly	Determines whether the user can change the text in the box; default is false, i.e., the user can not change the text.
Columns	The width of the text box in characters. The actual width is determined based on the font that is used for the text entry.
Rows	The height of a multi-line text box in lines. The default value is 0, means a single line text box.

The mostly used attribute for a label control is 'Text', which implies the text displayed on the label.

Check Boxes and Radio Buttons

A check box displays a single option that the user can either check or uncheck and radio buttons present a group of options from which the user can select just one option.

To create a group of radio buttons, you specify the same name for the GroupName attribute of each radio button in the group. If more than one group is required in a single form, then specify a different group name for each group.

If you want check box or radio button to be selected when the form is initially displayed, set its Checked attribute to true. If the Checked attribute is set to true for multiple radio buttons in a group, then only the last one is considered as true.

Basic syntax of check box:

```
<asp:CheckBox ID= "chkoption" runat= "Server">
</asp:CheckBox>
```

Basic syntax of radio button:

```
<asp:RadioButton ID= "rdboption" runat= "Server">
</asp: RadioButton>
```

Common properties of check boxes and radio buttons:

Property	**Description**
Text	The text displayed next to the check box or radio button.
Checked	Specifies whether it is selected or not, default is false.
GroupName	Name of the group the control belongs to.

List Controls

ASP.NET provides the following controls

- Drop-down list,
- List box,
- Radio button list,
- Check box list,
- Bulleted list.

These control let a user choose from one or more items from the list. List boxes and drop-down lists contain one or more list items. These lists can be loaded either by code or by the ListItemCollection editor.

Basic syntax of list box control:

```
<asp:ListBox ID="ListBox1" runat="server" AutoPostBack="True"
OnSelectedIndexChanged="ListBox1_SelectedIndexChanged">
</asp:ListBox>
```

Basic syntax of drop-down list control:

```
<asp:DropDownList ID="DropDownList1" runat="server" AutoPostBack="True"
OnSelectedIndexChanged="DropDownList1_SelectedIndexChanged">
</asp:DropDownList>
```

Common properties of list box and drop-down Lists:

Property	**Description**
Items	The collection of ListItem objects that represents the items in the control. This property returns an object of type ListItemCollection.
Rows	Specifies the number of items displayed in the box. If actual list contains more rows than displayed then a scroll bar is added.
SelectedIndex	The index of the currently selected item. If more than one item is selected, then the index of the first selected item. If no item is selected, the value of this property is -1.
SelectedValue	The value of the currently selected item. If more than one item is selected, then the value of the first selected item. If no item is selected, the value of this property is an empty string ("").
SelectionMode	Indicates whether a list box allows single selections or multiple selections.

Common properties of each list item objects:

Property	Description
Text	The text displayed for the item.
Selected	Indicates whether the item is selected.
Value	A string value associated with the item.

It is important to notes that:

- To work with the items in a drop-down list or list box, you use the Items property of the control. This property returns a ListItemCollection object which contains all the items of the list.
- The SelectedIndexChanged event is raised when the user selects a different item from a drop-down list or list box.

The ListItemCollection

The ListItemCollection object is a collection of ListItem objects. Each ListItem object represents one item in the list. Items in a ListItemCollection are numbered from 0.

When the items into a list box are loaded using strings like: lstcolor.Items.Add("Blue"), then both the Text and Value properties of the list item are set to the string value you specify. To set it differently you must create a list item object and then add that item to the collection.

The ListItemCollection Editor is used to add item to a drop-down list or list box. This is used to create a static list of items. To display the collection editor, select edit item from the smart tag menu, or select the control and then click the ellipsis button from the Item property in the properties window.

Common properties of ListItemCollection:

Property	Description
Item(integer)	A ListItem object that represents the item at the specified index.

Count	The number of items in the collection.

Common methods of ListItemCollection:

Methods	**Description**
Add(string)	Adds a new item at the end of the collection and assigns the string parameter to the Text property of the item.
Add(ListItem)	Adds a new item at the end of the collection.
Insert(integer, string)	Inserts an item at the specified index location in the collection, and assigns string parameter to the text property of the item.
Insert(integer, ListItem)	Inserts the item at the specified index location in the collection.
Remove(string)	Removes the item with the text value same as the string.
Remove(ListItem)	Removes the specified item.
RemoveAt(integer)	Removes the item at the specified index as the integer.
Clear	Removes all the items of the collection.
FindByValue(string)	Returns the item whose value is same as the string.
FindByValue(Text)	Returns the item whose text is same as the string.

Radio Button list and Check Box list

A radio button list presents a list of mutually exclusive options. A check box list presents a list of independent options. These controls contain a collection of ListItem objects that could be referred to through the Items property of the control.

Basic syntax of radio button list:

```
<asp:RadioButtonList ID="RadioButtonList1" runat="server" AutoPostBack="True"
  OnSelectedIndexChanged="RadioButtonList1_SelectedIndexChanged">
</asp:RadioButtonList>
```

Basic syntax of check box list:

```
<asp:CheckBoxList ID="CheckBoxList1" runat="server" AutoPostBack="True"
  OnSelectedIndexChanged="CheckBoxList1_SelectedIndexChanged">
</asp:CheckBoxList>
```

Common properties of check box and radio button lists:

Property	**Description**
RepeatLayout	This attribute specifies whether the table tags or the normal html flow to use while formatting the list when it is rendered. The default is Table.
RepeatDirection	It specifies the direction in which the controls to be repeated. The values available are Horizontal and Vertical. Default is Vertical.
RepeatColumns	It specifies the number of columns to use when repeating the controls; default is 0.

Bulleted lists and Numbered lists

The bulleted list control creates bulleted lists or numbered lists. These controls contain a collection of ListItem objects that could be referred to through the Items property of the control.

Basic syntax of a bulleted list:

```
<asp:BulletedList ID="BulletedList1" runat="server">
</asp:BulletedList>
```

Common properties of the bulleted list:

Property	Description
BulletStyle	This property specifies the style and looks of the bullets, or numbers.
RepeatDirection	It specifies the direction in which the controls to be repeated. The values available are Horizontal and Vertical. Default is Vertical.
RepeatColumns	It specifies the number of columns to use when repeating the controls; default is 0.

HyperLink Control

The HyperLink control is like the HTML <a> element.

Basic syntax for a hyperlink control:

```
<asp:HyperLink ID="HyperLink1" runat="server">
  HyperLink
</asp:HyperLink>
```

It has the following important properties:

Property	Description
ImageUrl	Path of the image to be displayed by the control.
NavigateUrl	Target link URL.
Text	The text to be displayed as the link.
Target	The window or frame which loads the linked page.

Image Control

The image control is used for displaying images on the web page, or some alternative text, if the image is not available.

Basic syntax for an image control:

```
<asp:Image ID="Image1" runat="server">
```

It has the following important properties:

Property	Description
AlternateText	Alternate text to be displayed in absence of the image.
ImageAlign	Alignment options for the control.
ImageUrl	Path of the image to be displayed by the control.

ASP.NET client side coding has two aspects:

- **Client side scripts** : It runs on the browser and in turn speeds up the execution of page. For example, client side data validation which can catch invalid data and warn the user accordingly without making a round trip to the server.
- **Client side source code** : ASP.NET pages generate this. For example, the HTML source code of an ASP.NET page contains a number of hidden fields and automatically injected blocks of JavaScript code, which keeps information like view state or does other jobs to make the page work.

Client Side Scripts

All ASP.NET server controls allow calling client side code written using JavaScript or VBScript. Some ASP.NET server controls use client side scripting to provide response to the users without posting back to the server. For example, the validation controls.

Apart from these scripts, the Button control has a property OnClientClick, which allows executing client-side script, when the button is clicked.

The traditional and server HTML controls have the following events that can execute a script when they are raised:

Event	Description
onblur	When the control loses focus
onfocus	When the control receives focus
onclick	When the control is clicked
onchange	When the value of the control changes
onkeydown	When the user presses a key
onkeypress	When the user presses an alphanumeric key
onkeyup	When the user releases a key
onmouseover	When the user moves the mouse pointer over the control
onserverclick	It raises the ServerClick event of the control, when the control is clicked

Client Side Source Code

We have already discussed that, ASP.NET pages are generally written in two files:

- The content file or the markup file (.aspx)
- The code-behind file

The content file contains the HTML or ASP.NET control tags and literals to form the structure of the page. The code behind file contains the class definition. At run-time, the content file is parsed and transformed into a page class.

This class, along with the class definition in the code file, and system generated code, together make the executable code (assembly) that processes all posted data, generates response, and sends it back to the client.

Consider the simple page:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
  Inherits="clientside._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>
      Untitled Page
    </title>
  </head>

  <body>
    <form id="form1" runat="server">

      <div>
        <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
        <asp:Button ID="Button1" runat="server" OnClick="Button1_Click" Text="Click" />
      </div>

      <hr />

      <h3> <asp:Label ID="Msg" runat="server" Text=""> </asp:Label> </h3>
    </form>
  </body>

</html>
```

When this page is run on the browser, the View Source option shows the HTML page sent to the browser by the ASP.Net runtime:

```
<!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>
    <title>
      Untitled Page
    </title>
  </head>

  <body>
    <form name="form1" method="post" action="Default.aspx" id="form1">

      <div>
        <input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"

value="/wEPDwUKMTU5MTA2ODYwOWRk31NudGDgvhhA7joJum9Qn5RxU2M=" />
      </div>

      <div>
        <input type="hidden" name="__EVENTVALIDATION"
id="__EVENTVALIDATION"

value="/wEWAwKpjZj0DALs0bLrBgKM54rGBhHsyM61rraxE+KnBTCS8cd1QDJ/"/>
      </div>

      <div>
        <input name="TextBox1" type="text" id="TextBox1" />
        <input type="submit" name="Button1" value="Click" id="Button1" />
      </div>

      <hr />
      <h3><span id="Msg"></span></h3>

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

If you go through the code properly, you can see that first two <div> tags contain the hidden fields which store the view state and validation information.

We have studied the page life cycle and how a page contains various controls. The page itself is instantiated as a control object. All web forms are basically instances of the ASP.NET Page class. The page class has the following extremely useful properties that correspond to intrinsic objects:

Server side code

- **Session: Background. ASP.NET session state enables you to store and retrieve values for a user as the user navigates ASP.NET pages in a Web application. HTTP is a stateless protocol. This means that a Web server treats each HTTP request for a page as an independent request**
- **Application**
- **Cache**
- **Request**
- **Response**
- **Server**
- **User**
- **Trace**

FAQ: WHAT IS SESSION IN ASP.NET?

We will discuss each of these objects in due time. In this tutorial we will explore the Server object, the Request object, and the Response object.

Server Object

The Server object in Asp.NET is an instance of the System.Web.HttpServerUtility class. The HttpServerUtility class provides numerous properties and methods to perform various jobs.

Properties and Methods of the Server object

The methods and properties of the HttpServerUtility class are exposed through the intrinsic Server object provided by ASP.NET.

The following table provides a list of the properties:

Property	Description
MachineName	Name of server computer
ScriptTimeOut	Gets and sets the request time-out value in seconds.

The following table provides a list of some important methods:

Method	Description
CreateObject(String)	Creates an instance of the COM object identified by its ProgID (Programmatic ID).
CreateObject(Type)	Creates an instance of the COM object identified by its Type.
Equals(Object)	Determines whether the specified Object is equal to the current Object.
Execute(String)	Executes the handler for the specified virtual path in the context of the current request.
Execute(String, Boolean)	Executes the handler for the specified virtual path in the context of the current request and specifies whether to clear the QueryString and Form collections.
GetLastError	Returns the previous exception.
GetType	Gets the Type of the current instance.
HtmlEncode	Changes an ordinary string into a string with legal HTML characters.
HtmlDecode	Converts an Html string into an ordinary string.
ToString	Returns a String that represents the current Object.
Transfer(String)	For the current request, terminates execution of the current page and starts execution of a new page by using the specified URL path of the page.

UrlDecode	Converts an URL string into an ordinary string.
UrlEncodeToken	Works same as UrlEncode, but on a byte array that contains Base64-encoded data.
UrlDecodeToken	Works same as UrlDecode, but on a byte array that contains Base64-encoded data.
MapPath	Return the physical path that corresponds to a specified virtual file path on the server.
Transfer	Transfers execution to another web page in the current application.

Request Object

The request object is an instance of the System.Web.HttpRequest class. It represents the values and properties of the HTTP request that makes the page loading into the browser.

The information presented by this object is wrapped by the higher level abstractions (the web control model). However, this object helps in checking some information such as the client browser and cookies.

Properties and Methods of the Request Object

The following table provides some noteworthy properties of the Request object:

Property	Description
AcceptTypes	Gets a string array of client-supported MIME accept types.
ApplicationPath	Gets the ASP.NET application's virtual application root path on the server.
Browser	Gets or sets information about the requesting client's browser

	capabilities.
ContentEncoding	Gets or sets the character set of the entity-body.
ContentLength	Specifies the length, in bytes, of content sent by the client.
ContentType	Gets or sets the MIME content type of the incoming request.
Cookies	Gets a collection of cookies sent by the client.
FilePath	Gets the virtual path of the current request.
Files	Gets the collection of files uploaded by the client, in multipart MIME format.
Form	Gets a collection of form variables.
Headers	Gets a collection of HTTP headers.
HttpMethod	Gets the HTTP data transfer method (such as GET, POST, or HEAD) used by the client.
InputStream	Gets the contents of the incoming HTTP entity body.
IsSecureConnection	Gets a value indicating whether the HTTP connection uses secure sockets (that is, HTTPS).
QueryString	Gets the collection of HTTP query string variables.

RawUrl	Gets the raw URL of the current request.
RequestType	Gets or sets the HTTP data transfer method (GET or POST) used by the client.
ServerVariables	Gets a collection of Web server variables.
TotalBytes	Gets the number of bytes in the current input stream.
Url	Gets information about the URL of the current request.
UrlReferrer	Gets information about the URL of the client's previous request that is linked to the current URL.
UserAgent	Gets the raw user agent string of the client browser.
UserHostAddress	Gets the IP host address of the remote client.
UserHostName	Gets the DNS name of the remote client.
UserLanguages	Gets a sorted string array of client language preferences.

The following table provides a list of some important methods:

Method	**Description**
BinaryRead	Performs a binary read of a specified number of bytes from the current input stream.
Equals(Object)	Determines whether the specified object is equal to the current object.

	(Inherited from object.)
GetType	Gets the Type of the current instance.
MapImageCoordinates	Maps an incoming image-field form parameter to appropriate x-coordinate and y-coordinate values.
MapPath(String)	Maps the specified virtual path to a physical path.
SaveAs	Saves an HTTP request to disk.
ToString	Returns a String that represents the current object.
ValidateInput	Causes validation to occur for the collections accessed through the Cookies, Form, and QueryString properties.

Response Object

The Response object represents the server's response to the client request. It is an instance of the System.Web.HttpResponse class.

In ASP.NET, the response object does not play any vital role in sending HTML text to the client, because the server-side controls have nested, object oriented methods for rendering themselves.

However, the HttpResponse object still provides some important functionalities, like the cookie feature and the Redirect() method. The Response.Redirect() method allows transferring the user to another page, inside as well as outside the application. It requires a round trip.

Properties and Methods of the Response Object

The following table provides some noteworthy properties of the Response object:

Property	Description
Buffer	Gets or sets a value indicating whether to buffer the output and send it after the complete response is finished processing.
BufferOutput	Gets or sets a value indicating whether to buffer the output and send it after the complete page is finished processing.
Charset	Gets or sets the HTTP character set of the output stream.
ContentEncoding	Gets or sets the HTTP character set of the output stream.
ContentType	Gets or sets the HTTP MIME type of the output stream.
Cookies	Gets the response cookie collection.
Expires	Gets or sets the number of minutes before a page cached on a browser expires.
ExpiresAbsolute	Gets or sets the absolute date and time at which to remove cached information from the cache.
HeaderEncoding	Gets or sets an encoding object that represents the encoding for the current header output stream.
Headers	Gets the collection of response headers.
IsClientConnected	Gets a value indicating whether the client is still connected to the server.

Output	Enables output of text to the outgoing HTTP response stream.
OutputStream	Enables binary output to the outgoing HTTP content body.
RedirectLocation	Gets or sets the value of the Http Location header.
Status	Sets the status line that is returned to the client.
StatusCode	Gets or sets the HTTP status code of the output returned to the client.
StatusDescription	Gets or sets the HTTP status string of the output returned to the client.
SubStatusCode	Gets or sets a value qualifying the status code of the response.
SuppressContent	Gets or sets a value indicating whether to send HTTP content to the client.

The following table provides a list of some important methods:

Method	Description
AddHeader	Adds an HTTP header to the output stream. AddHeader is provided for compatibility with earlier versions of ASP.
AppendCookie	Infrastructure adds an HTTP cookie to the intrinsic cookie collection.
AppendHeader	Adds an HTTP header to the output stream.
AppendToLog	Adds custom log information to the InterNET Information Services (IIS) log file.

BinaryWrite	Writes a string of binary characters to the HTTP output stream.
ClearContent	Clears all content output from the buffer stream.
Close	Closes the socket connection to a client.
End	Sends all currently buffered output to the client, stops execution of the page, and raises the EndRequest event.
Equals(Object)	Determines whether the specified object is equal to the current object.
Flush	Sends all currently buffered output to the client.
GetType	Gets the Type of the current instance.
Pics	Appends a HTTP PICS-Label header to the output stream.
Redirect(String)	Redirects a request to a new URL and specifies the new URL.
Redirect(String, Boolean)	Redirects a client to a new URL. Specifies the new URL and whether execution of the current page should terminate.
SetCookie	Updates an existing cookie in the cookie collection.
ToString	Returns a String that represents the current Object.
TransmitFile(String)	Writes the specified file directly to an HTTP response output stream, without buffering it in memory.

Write(Char)	Writes a character to an HTTP response output stream.
Write(Object)	Writes an object to an HTTP response stream.
Write(String)	Writes a string to an HTTP response output stream.
WriteFile(String)	Writes the contents of the specified file directly to an HTTP response output stream as a file block.
WriteFile(String, Boolean)	Writes the contents of the specified file directly to an HTTP response output stream as a memory block.

Example

The following simple example has a text box control where the user can enter name, a button to send the information to the server, and a label control to display the URL of the client computer.

The content file:

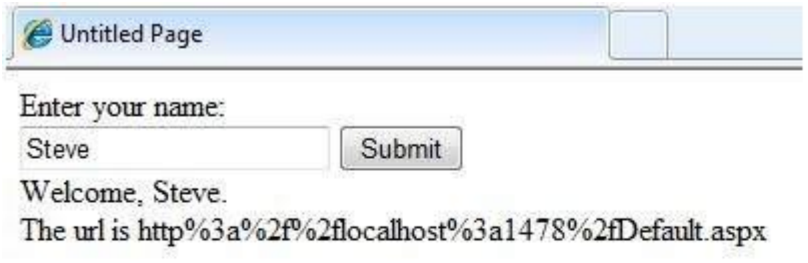
```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
  Inherits="server_side._Default" %>

<!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>Untitled Page</title>
  </head>

  <body>
    <form id="form1" runat="server">
      <div>

        Enter your name:
        <br />
```

```
          <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
          <asp:Button ID="Button1" runat="server" OnClick="Button1_Click" Text="Submit" />
          <br />
          <asp:Label ID="Label1" runat="server"/>

        </div>
      </form>
   </body>

</html>
```

The code behind Button1_Click:

```
protected void Button1_Click(object sender, EventArgs e) {

   if (!String.IsNullOrEmpty(TextBox1.Text)) {

      // Access the HttpServerUtility methods through
      // the intrinsic Server object.
      Label1.Text = "Welcome, " + Server.HtmlEncode(TextBox1.Text) + ". <br/> The url is " +
Server.UrlEncode(Request.Url.ToString())
   }
}
```

Run the page to see the following result:



```
Welcome, Steve.
The url is http%3a%2f%2flocalhost%3a1478%2fDefault.aspx
```

Hyper Text Transfer Protocol (HTTP) is a stateless protocol. When the client disconnects from the server, the ASP.NET engine discards the page objects. This way, each web application can scale up to serve numerous requests simultaneously without running out of server memory.

However, there needs to be some technique to store the information between requests and to retrieve it when required. This information i.e., the current value of all the controls and variables for the current user in the current session is called the State.

STATE MANAGEMENYT IN ASP.NET

ASP.NET manages four types of states:

- View State
- Control State
- Session State
- Application State

FAQ: EXPLAIN THE VARIOUS STATE MANAGEMENT IN ASP.NET

View State

The view state is the state of the page and all its controls. It is automatically maintained across posts by the ASP.NET framework.

When a page is sent back to the client, the changes in the properties of the page and its controls are determined, and stored in the value of a hidden input field named _VIEWSTATE. When the page is again posted back, the _VIEWSTATE field is sent to the server with the HTTP request.

The view state could be enabled or disabled for:

- **The entire application** by setting the EnableViewState property in the <pages> section of web.config file.
- **A page** by setting the EnableViewState attribute of the Page directive, as <%@ Page Language="C#" EnableViewState="false" %>
- **A control** by setting the Control.EnableViewState property.

It is implemented using a view state object defined by the StateBag class which defines a collection of view state items. The state bag is a data structure containing attribute value pairs, stored as strings associated with objects.

The StateBag class has the following properties:

Properties	Description
Item(name)	The value of the view state item with the specified name. This is the default property of the StateBag class.
Count	The number of items in the view state collection.
Keys	Collection of keys for all the items in the collection.
Values	Collection of values for all the items in the collection.

The StateBag class has the following methods:

Methods	Description
Add(name, value)	Adds an item to the view state collection and existing item is updated.
Clear	Removes all the items from the collection.
Equals(Object)	Determines whether the specified object is equal to the current object.
Finalize	Allows it to free resources and perform other cleanup operations.
GetEnumerator	Returns an enumerator that iterates over all the key/value pairs of the StateItem objects stored in the StateBag object.
GetType	Gets the type of the current instance.

IsItemDirty	Checks a StateItem object stored in the StateBag object to evaluate whether it has been modified.
Remove(name)	Removes the specified item.
SetDirty	Sets the state of the StateBag object as well as the Dirty property of each of the StateItem objects contained by it.
SetItemDirty	Sets the Dirty property for the specified StateItem object in the StateBag object.
ToString	Returns a string representing the state bag object.

Example

The following example demonstrates the concept of storing view state. Let us keep a counter, which is incremented each time the page is posted back by clicking a button on the page. A label control shows the value in the counter.

The markup file code is as follows:

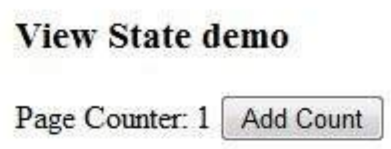
```
<%@ Page Language="C#" AutoEventWireup="true"
CodeBehind="Default.aspx.cs" Inherits="statedemo._Default" %>

<!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>
         Untitled Page
      </title>
   </head>

   <body>
      <form id="form1" runat="server">

         <div>
            <h3>View State demo</h3>

            Page Counter:
```

```
            <asp:Label ID="lblCounter" runat="server" />
            <asp:Button ID="btnIncrement" runat="server" Text="Add
Count" onclick="btnIncrement_Click" />
         </div>

      </form>
   </body>

</html>
```

The code behind file for the example is shown here:

```
public partial class _Default : System.Web.UI.Page
{
   public int counter
   {
      get
      {
         if (ViewState["pcounter"] != null)
         {
            return ((int)ViewState["pcounter"]);
         }
         else
         {
            return 0;
         }
      }

      set
      {
         ViewState["pcounter"] = value;
      }
   }

   protected void Page_Load(object sender, EventArgs e)
   {
      lblCounter.Text = counter.ToString();
      counter++;
   }
}
```

It would produce the following result:


View State demo

Page Counter: 1

Control State

Control state cannot be modified, accessed directly, or disabled.

Session State

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 state is generally used for storing application data such as inventory, supplier list, customer record, or shopping cart. It can also keep information about the user and his preferences, and keep the track of pending operations.

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.

The session state object is created from the HttpSessionState class, which defines a collection of session state items.

The HttpSessionState class has the following properties:

Properties	Description
SessionID	The unique session identifier.
Item(name)	The value of the session state item with the specified name. This is the default property of the HttpSessionState class.
Count	The number of items in the session state collection.
TimeOut	Gets and sets the amount of time, in minutes, allowed between requests before the session-state provider terminates the session.

The HttpSessionState class has the following methods:

Methods	Description

Add(name, value)	Adds an item to the session state collection.
Clear	Removes all the items from session state collection.
Remove(name)	Removes the specified item from the session state collection.
RemoveAll	Removes all keys and values from the session-state collection.
RemoveAt	Deletes an item at a specified index from the session-state collection.

The session state object is a name-value pair to store and retrieve some information from the session state object. You could use the following code for the same:

```
void StoreSessionInfo()
{
   String fromuser = TextBox1.Text;
   Session["fromuser"] = fromuser;
}

void RetrieveSessionInfo()
{
   String fromuser = Session["fromuser"];
   Label1.Text = fromuser;
}
```

The above code stores only strings in the Session dictionary object, however, it can store all the primitive data types and arrays composed of primitive data types, as well as the DataSet, DataTable, HashTable, and Image objects, as well as any user-defined class that inherits from the ISerializable object.

Example

The following example demonstrates the concept of storing session state. There are two buttons on the page, a text box to enter string and a label to display the text stored from last session.

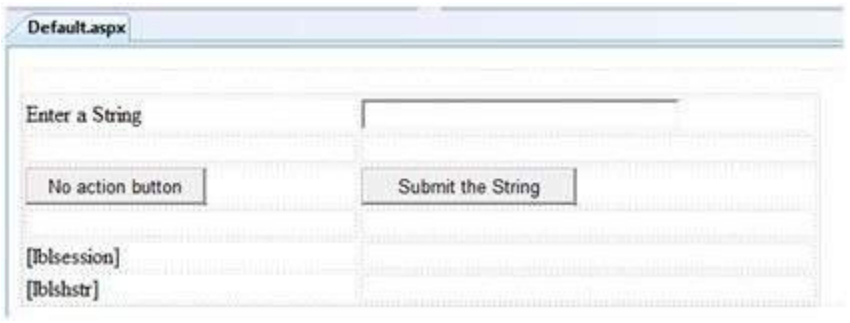
The mark up file code is as follows:

```
<%@ 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>
         Untitled Page
      </title>
   </head>

   <body>
      <form id="form1" runat="server">
         <div>
            &nbsp; &nbsp; &nbsp;

            <table style="width: 568px; height: 103px">

               <tr>
                  <td style="width: 209px">
                     <asp:Label ID="lblstr" runat="server"
Text="Enter a String"  style="width:94px">
                     </asp:Label>
                  </td>

                  <td style="width: 317px">
                     <asp:TextBox ID="txtstr" runat="server"
style="width:227px">
                     </asp:TextBox>
                  </td>
               </tr>

               <tr>
                  <td style="width: 209px"> </td>
                  <td style="width: 317px"> </td>
               </tr>

               <tr>
                  <td style="width: 209px">
                     <asp:Button ID="btnnrm" runat="server"
                        Text="No action button" style="width:128px"
/>
                  </td>

                  <td style="width: 317px">
                     <asp:Button ID="btnstr" runat="server"
                        OnClick="btnstr_Click" Text="Submit the
String" />
                  </td>
               </tr>

               <tr>
```

```
                  <td style="width: 209px">  </td>

                  <td style="width: 317px">  </td>
               </tr>

               <tr>
                  <td style="width: 209px">
                     <asp:Label ID="lblsession" runat="server"
style="width:231px"  >
                     </asp:Label>
                  </td>

                  <td style="width: 317px">  </td>
               </tr>

               <tr>
                  <td style="width: 209px">
                     <asp:Label ID="lblshstr" runat="server">
                     </asp:Label>
                  </td>

                  <td style="width: 317px">  </td>
               </tr>

            </table>

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

It should look like the following in design view:

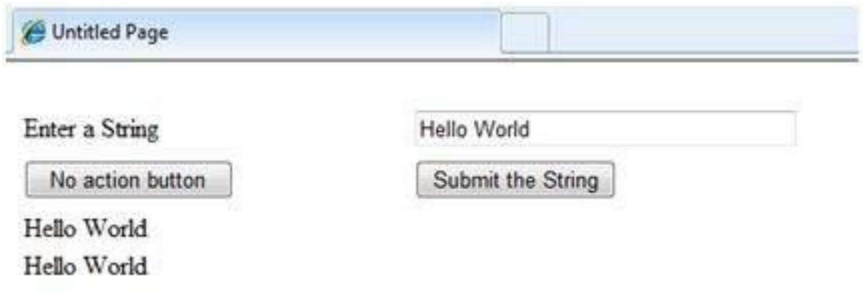


The code behind file is given here:

```
public partial class _Default : System.Web.UI.Page
{
   String mystr;

   protected void Page_Load(object sender, EventArgs e)
```

```
    {
       this.lblshstr.Text = this.mystr;
       this.lblsession.Text = (String)this.Session["str"];
    }

    protected void btnstr_Click(object sender, EventArgs e)
    {
       this.mystr = this.txtstr.Text;
       this.Session["str"] = this.txtstr.Text;
       this.lblshstr.Text = this.mystr;
       this.lblsession.Text = (String)this.Session["str"];
    }
}
```

Execute the file and observe how it works:



Hello World

Hello World

Application State

The ASP.NET application is the collection of all web pages, code and other files within a single virtual directory on a web server. When information is stored in application state, it is available to all the users.

To provide for the use of application state, ASP.NET creates an application state object for each application from the HTTPApplicationState class and stores this object in server memory. This object is represented by class file global.asax.

Application State is mostly used to store hit counters and other statistical data, global application data like tax rate, discount rate etc. and to keep the track of users visiting the site.

The HttpApplicationState class has the following properties:

Properties	Description
Item(name)	The value of the application state item with the specified name. This is the default property of the HttpApplicationState class.

Count	The number of items in the application state collection.

The HttpApplicationState class has the following methods:

Methods	Description
Add(name, value)	Adds an item to the application state collection.
Clear	Removes all the items from the application state collection.
Remove(name)	Removes the specified item from the application state collection.
RemoveAll	Removes all objects from an HttpApplicationState collection.
RemoveAt	Removes an HttpApplicationState object from a collection by index.
Lock()	Locks the application state collection so only the current user can access it.
Unlock()	Unlocks the application state collection so all the users can access it.

Application state data is generally maintained by writing handlers for the events:

- Application_Start
- Application_End
- Application_Error
- Session_Start
- Session_End

The following code snippet shows the basic syntax for storing application state information:

```
Void Application_Start(object sender, EventArgs e)
{
   Application["startMessage"] = "The application has started.";
}

Void Application_End(object sender, EventArgs e)
```

```
{
   Application["endtMessage"] = "The application has ended.";
}
```

******END OF UNIT-II******

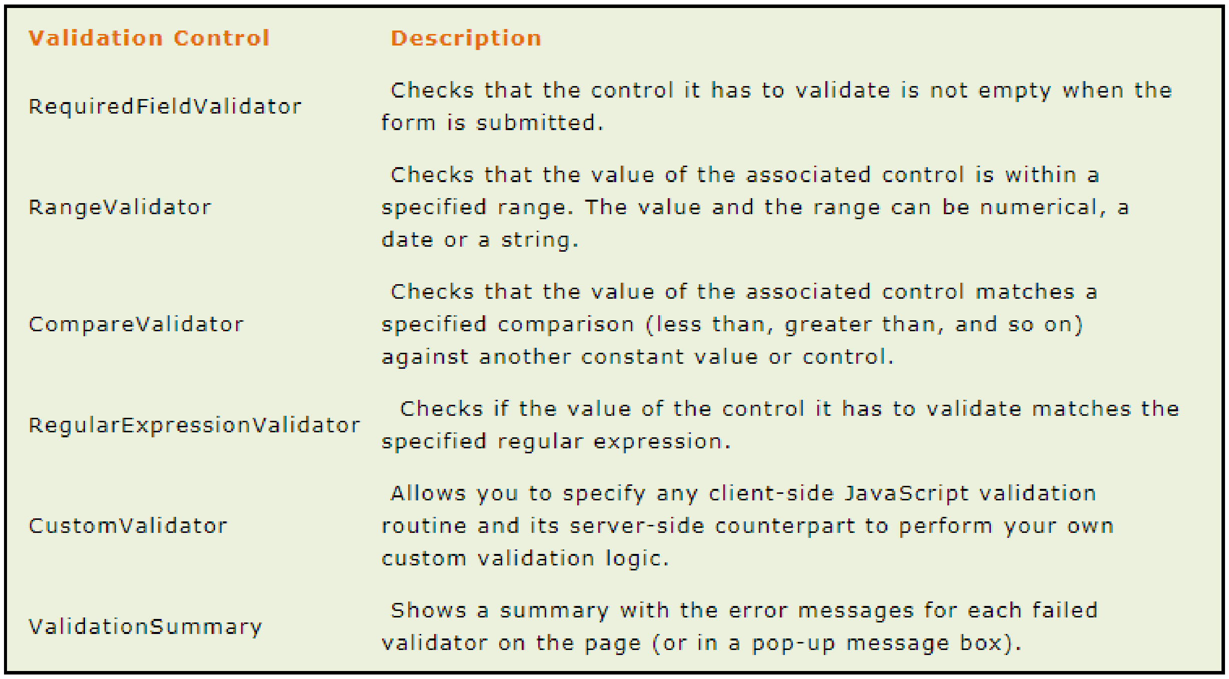
FAQ: DEFINE VALIDATION

****** UNIT – III ******

VALIDATORS

ASP.NET **validation** controls validate the user input data to ensure that useless, unauthenticated, or contradictory data don't get stored.

The available validation server controls are



Validation Control	Description
RequiredFieldValidator	Checks that the control it has to validate is not empty when the form is submitted.
RangeValidator	Checks that the value of the associated control is within a specified range. The value and the range can be numerical, a date or a string.
CompareValidator	Checks that the value of the associated control matches a specified comparison (less than, greater than, and so on) against another constant value or control.
RegularExpressionValidator	Checks if the value of the control it has to validate matches the specified regular expression.
CustomValidator	Allows you to specify any client-side JavaScript validation routine and its server-side counterpart to perform your own custom validation logic.
ValidationSummary	Shows a summary with the error messages for each failed validator on the page (or in a pop-up message box).

ASP.NET provides the following validation controls:

FAQ: DISCUSS THE VARIOUS VALIDATORS WITH EXAMPLES

- RequiredFieldValidator
- RangeValidator
- CompareValidator
- RegularExpressionValidator
- CustomValidator
- ValidationSummary

BaseValidator Class

The validation control classes are inherited from the BaseValidator class hence they inherit its properties and methods. Therefore, it would help to take a look at the properties and the methods of this base class, which are common for all the validation controls:

Members	Description
ControlToValidate	Indicates the input control to validate.
Display	Indicates how the error message is shown.
EnableClientScript	Indicates whether client side validation will take.
Enabled	Enables or disables the validator.
ErrorMessage	Indicates error string.
Text	Error text to be shown if validation fails.
IsValid	Indicates whether the value of the control is valid.
SetFocusOnError	It indicates whether in case of an invalid control, the focus should switch to the related input control.
ValidationGroup	The logical group of multiple validators, where this control belongs.
Validate()	This method revalidates the control and updates the IsValid property.

RequiredFieldValidator Control

The RequiredFieldValidator control ensures that the required field is not empty. It is generally tied to a text box to force input into the text box.

The syntax of the control is as given:

```
<asp:RequiredFieldValidator ID="rfvcandidate"
  runat="server" ControlToValidate ="ddlcandidate"
  ErrorMessage="Please choose a candidate"
  InitialValue="Please choose a candidate">

</asp:RequiredFieldValidator>
```

RangeValidator Control

The **RangeValidator** control verifies that the input value falls within a predetermined range.

It has three specific properties:

Properties	Description
Type	It defines the type of the data. The available values are: Currency, Date, Double, Integer, and String.
MinimumValue	It specifies the minimum value of the range.
MaximumValue	It specifies the maximum value of the range.

The syntax of the control is as given:

```
<asp:RangeValidator ID="rvclass" runat="server" ControlToValidate="txtclass"
  ErrorMessage="Enter your class (6 - 12)" MaximumValue="12"
  MinimumValue="6" Type="Integer">

</asp:RangeValidator>
```

CompareValidator Control

The **CompareValidator** control compares a value in one control with a fixed value or a value in another control.

It has the following specific properties:

Properties	Description
Type	It specifies the data type.
ControlToCompare	It specifies the value of the input control to compare with.
ValueToCompare	It specifies the constant value to compare with.
Operator	It specifies the comparison operator, the available values are: Equal, NotEqual, GreaterThan, GreaterThanEqual, LessThan, LessThanEqual, and DataTypeCheck.

The basic syntax of the control is as follows:

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

</asp:CompareValidator>
```

RegularExpressionValidator

The **RegularExpressionValidator** allows validating the input text by matching against a pattern of a regular expression. The regular expression is set in the ValidationExpression property.

The following table summarizes the commonly used syntax constructs for regular expressions:

Character Escapes	Description
\b	Matches a backspace.
\t	Matches a tab.

\r	Matches a carriage return.
\v	Matches a vertical tab.
\f	Matches a form feed.
\n	Matches a new line.
\	Escape character.

Apart from single character match, a class of characters could be specified that can be matched, called the metacharacters.

Metacharacters	**Description**
.	Matches any character except \n.
[abcd]	Matches any character in the set.
[^abcd]	Excludes any character in the set.
[2-7a-mA-M]	Matches any character specified in the range.
\w	Matches any alphanumeric character and underscore.
\W	Matches any non-word character.
\s	Matches whitespace characters like, space, tab, new line etc.

\S	Matches any non-whitespace character.
\d	Matches any decimal character.
\D	Matches any non-decimal character.

Quantifiers could be added to specify number of times a character could appear.

Quantifier	Description
*	Zero or more matches.
+	One or more matches.
?	Zero or one matches.
{N}	N matches.
{N,}	N or more matches.
{N,M}	Between N and M matches.

The syntax of the control is as given:

```
<asp:RegularExpressionValidator ID="string" runat="server" ErrorMessage="string"
   ValidationExpression="string" ValidationGroup="string">

</asp:RegularExpressionValidator>
```

CustomValidator

The **CustomValidator** control allows writing application specific custom validation routines for both the client side and the server side validation.

The client side validation is accomplished through the ClientValidationFunction property. The client side validation routine should be written in a scripting language, such as JavaScript or VBScript, which the browser can understand.

The server side validation routine must be called from the control's ServerValidate event handler. The server side validation routine should be written in any .Net language, like C# or VB.Net.

The basic syntax for the control is as given:

```
<asp:CustomValidator ID="CustomValidator1" runat="server"
  ClientValidationFunction=.cvf_func. ErrorMessage="CustomValidator">

</asp:CustomValidator>
```

ValidationSummary

The **ValidationSummary** control does not perform any validation but shows a summary of all errors in the page. The summary displays the values of the ErrorMessage property of all validation controls that failed validation.

The following two mutually inclusive properties list out the error message:

- **ShowSummary** : shows the error messages in specified format.
- **ShowMessageBox** : shows the error messages in a separate window.

The syntax for the control is as given:

```
<asp:ValidationSummary ID="ValidationSummary1" runat="server"
  DisplayMode = "BulletList" ShowSummary = "true" HeaderText="Errors:" />
```

Validation Groups

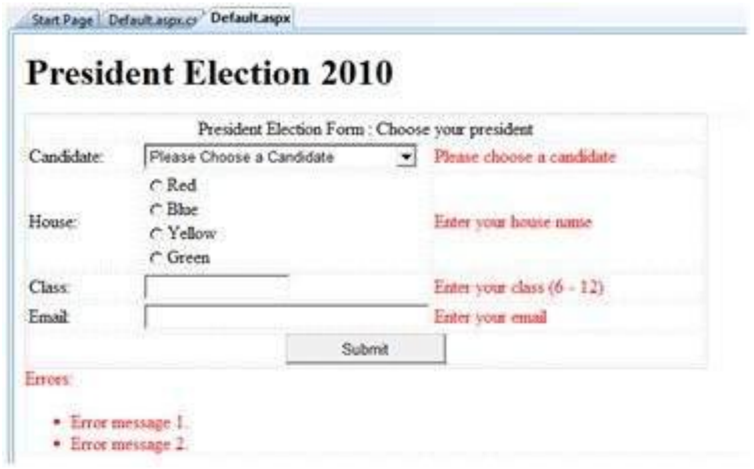
Complex pages have different groups of information provided in different panels. In such situation, a need might arise for performing validation separately for separate group. This kind of situation is handled using validation groups.

To create a validation group, you should put the input controls and the validation controls into the same logical group by setting their ***ValidationGroup*** property.

Example

The following example describes a form to be filled up by all the students of a school, divided into four houses, for electing the school president. Here, we use the validation controls to validate the user input.

This is the form in design view:



President Election 2010

President Election Form : Choose your president

Candidate: Please Choose a Candidate — Please choose a candidate

House: Red / Blue / Yellow / Green — Enter your house name

Class: — Enter your class (6 - 12)

Email: — Enter your email

Errors:

- Error message 1.
- Error message 2.

The content file code is as given:

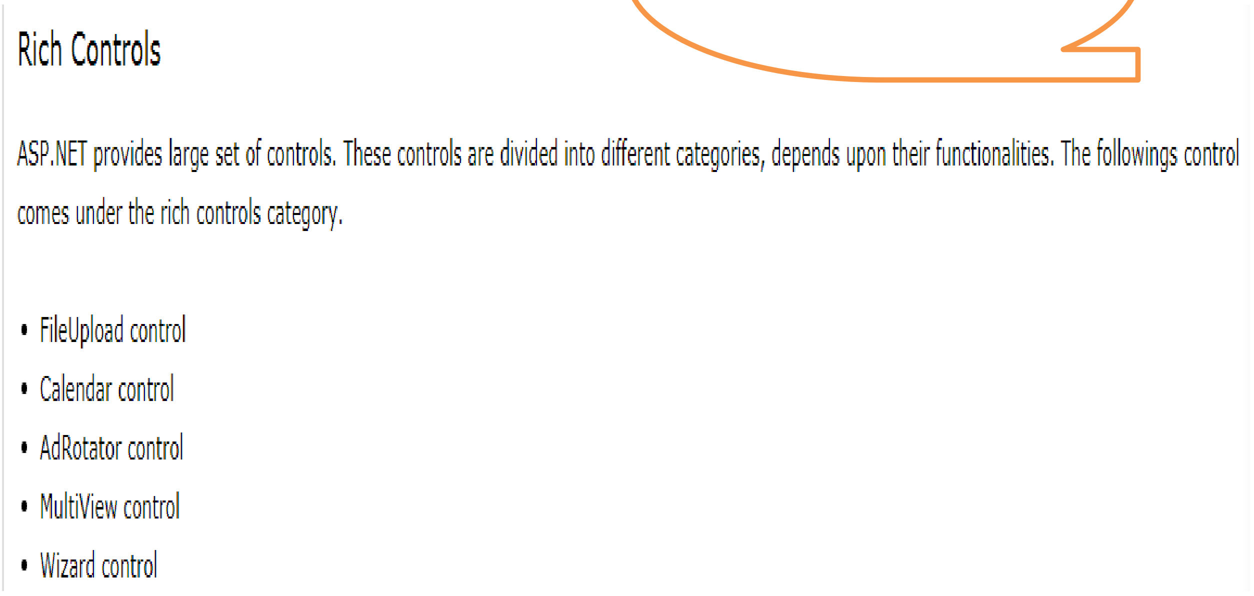
```
<form id="form1" runat="server">

  <table style="width: 66%;">

    <tr>
      <td class="style1" colspan="3" align="center">
      <asp:Label ID="lblmsg"
        Text="President Election Form : Choose your president"
        runat="server" />
      </td>
    </tr>

    <tr>
      <td class="style3">
        Candidate:
      </td>

      <td class="style2">
        <asp:DropDownList ID="ddlcandidate" runat="server"  style="width:239px">
          <asp:ListItem>Please Choose a Candidate</asp:ListItem>
          <asp:ListItem>M H Kabir</asp:ListItem>
          <asp:ListItem>Steve Taylor</asp:ListItem>
          <asp:ListItem>John Abraham</asp:ListItem>
          <asp:ListItem>Venus Williams</asp:ListItem>
        </asp:DropDownList>
      </td>

      <td>
        <asp:RequiredFieldValidator ID="rfvcandidate"
          runat="server" ControlToValidate ="ddlcandidate"
          ErrorMessage="Please choose a candidate"
          InitialValue="Please choose a candidate">
        </asp:RequiredFieldValidator>
      </td>
    </tr>

    <tr>
      <td class="style3">
        House:
      </td>

      <td class="style2">
        <asp:RadioButtonList ID="rblhouse" runat="server" RepeatLayout="Flow">
          <asp:ListItem>Red</asp:ListItem>
          <asp:ListItem>Blue</asp:ListItem>
          <asp:ListItem>Yellow</asp:ListItem>
          <asp:ListItem>Green</asp:ListItem>
        </asp:RadioButtonList>
      </td>

      <td>
        <asp:RequiredFieldValidator ID="rfvhouse" runat="server"
          ControlToValidate="rblhouse" ErrorMessage="Enter your house name" >
        </asp:RequiredFieldValidator>
        <br />
      </td>
    </tr>

    <tr>
      <td class="style3">
        Class:
      </td>

      <td class="style2">
        <asp:TextBox ID="txtclass" runat="server"></asp:TextBox>
      </td>

      <td>
        <asp:RangeValidator ID="rvclass"
          runat="server" ControlToValidate="txtclass"
          ErrorMessage="Enter your class (6 - 12)" MaximumValue="12"
          MinimumValue="6" Type="Integer">
        </asp:RangeValidator>
      </td>
    </tr>

    <tr>
      <td class="style3">
        Email:
      </td>

      <td class="style2">
        <asp:TextBox ID="txtemail" runat="server" style="width:250px">
        </asp:TextBox>
      </td>

      <td>
        <asp:RegularExpressionValidator ID="remail" runat="server"
          ControlToValidate="txtemail" ErrorMessage="Enter your email"
          ValidationExpression="\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*">
        </asp:RegularExpressionValidator>
      </td>
    </tr>

    <tr>
      <td class="style3" align="center" colspan="3">
        <asp:Button ID="btnsubmit" runat="server" onclick="btnsubmit_Click"
          style="text-align: center" Text="Submit" style="width:140px" />
      </td>
    </tr>
  </table>
  <asp:ValidationSummary ID="ValidationSummary1" runat="server"
    DisplayMode ="BulletList" ShowSummary ="true" HeaderText="Errors:" />
</form>
```

The code behind the submit button:

```
protected void btnsubmit_Click(object sender, EventArgs e)
{
  if (Page.IsValid)
  {
    lblmsg.Text = "Thank You";
  }
  else
  {
    lblmsg.Text = "Fill up all the fields";
  }
```

RICH CONTROLS

FAQ: WHAT ARE RICH CONTROLS?



Rich Controls

ASP.NET provides large set of controls. These controls are divided into different categories, depends upon their functionalities. The followings control comes under the rich controls category.

- FileUpload control
- Calendar control
- AdRotator control
- MultiView control
- Wizard control

ASP.NET has two controls that allow users to upload files to the web server. Once the server receives the posted file data, the application can save it, check it, or ignore it. The following controls allow the file uploading:

- **HtmlInputFile** - an HTML server control

- **FileUpload** - and ASP.NET web control

Both controls allow file uploading, but the FileUpload control automatically sets the encoding of the form, whereas the HtmlInputFile does not do so.

In this tutorial, we use the FileUpload control. The FileUpload control allows the user to browse for and select the file to be uploaded, providing a browse button and a text box for entering the filename.

Once, the user has entered the filename in the text box by typing the name or browsing, the SaveAs method of the FileUpload control can be called to save the file to the disk.

The basic syntax of FileUpload is:

```
<asp:FileUpload ID= "Uploader" runat = "server" />
```

The FileUpload class is derived from the WebControl class, and inherits all its members. Apart from those, the FileUpload class has the following read-only properties:

Properties	Description
FileBytes	Returns an array of the bytes in a file to be uploaded.
FileContent	Returns the stream object pointing to the file to be uploaded.
FileName	Returns the name of the file to be uploaded.
HasFile	Specifies whether the control has a file to upload.
PostedFile	Returns a reference to the uploaded file.

The posted file is encapsulated in an object of type HttpPostedFile, which could be accessed through the PostedFile property of the FileUpload class.

The HttpPostedFile class has the following frequently used properties:

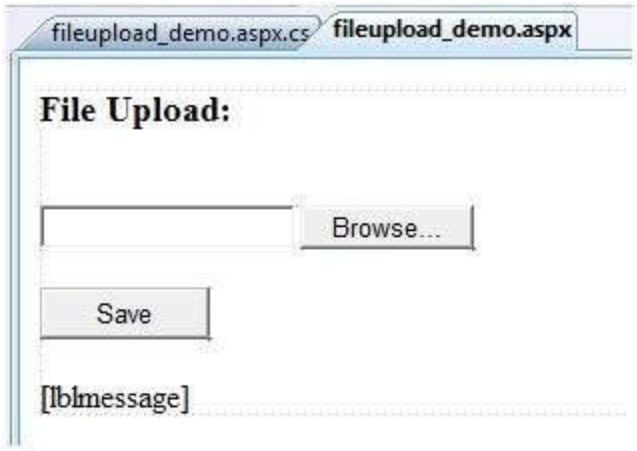
Properties	Description

ContentLength	Returns the size of the uploaded file in bytes.
ContentType	Returns the MIME type of the uploaded file.
FileName	Returns the full filename.
InputStream	Returns a stream object pointing to the uploaded file.

Example

The following example demonstrates the FileUpload control and its properties. The form has a **FileUpload** control along with a save button and a label control for displaying the file name, file type, and file length.

In the design view, the form looks as follows:



The content file code is as given:

```
<body>
  <form id="form1" runat="server">

    <div>
      <h3> File Upload:</h3>
      <br />
      <asp:FileUpload ID="FileUpload1" runat="server" />
      <br /><br />
      <asp:Button ID="btnsave" runat="server" onclick="btnsave_Click"  Text="Save"
style="width:85px" />
      <br /><br />
      <asp:Label ID="lblmessage" runat="server" />
    </div>

  </form>
</body>
```

The code behind the save button is as given:

```
protected void btnsave_Click(object sender, EventArgs e)
{
  StringBuilder sb = new StringBuilder();

  if (FileUpload1.HasFile)
  {
    try
    {
      sb.AppendFormat(" Uploading file: {0}", FileUpload1.FileName);

      //saving the file
      FileUpload1.SaveAs("<c:\\SaveDirectory>" + FileUpload1.FileName);

      //Showing the file information
      sb.AppendFormat("<br/> Save As: {0}",  FileUpload1.PostedFile.FileName);
      sb.AppendFormat("<br/> File type: {0}",    FileUpload1.PostedFile.ContentType);
      sb.AppendFormat("<br/> File length: {0}",  FileUpload1.PostedFile.ContentLength);
      sb.AppendFormat("<br/> File name: {0}",  FileUpload1.PostedFile.FileName);

    }catch (Exception ex)
    {
      sb.Append("<br/> Error <br/>");
      sb.AppendFormat("Unable to save file <br/> {0}", ex.Message);
    }
  }
  else
  {
    lblmessage.Text = sb.ToString();
  }
}
```

Note the following:

- The StringBuilder class is derived from System.IO namespace, so it needs to be included.
- The try and catch blocks are used for catching errors, and display the error message.

CALENDER CONTROL

The calendar control is a functionally rich web control, which provides the following capabilities:

- **Displaying one month at a time**
- **Selecting a day, a week or a month**
- **Selecting a range of days**
- **Moving from month to month**
- **Controlling the display of the days programmatically**

FAQ: DESCIRBE THE CALENDER CONTROL WITH AN EXAMPLE

The basic syntax of a calendar control is:

```
<asp:Calender ID = "Calendar1" runat = "server">
</asp:Calender>
```

Properties and Events of the Calendar Control

The calendar control has many properties and events, using which you can customize the actions and display of the control. The following table provides some important properties of the Calendar control:

Properties	Description
Caption	Gets or sets the caption for the calendar control.
CaptionAlign	Gets or sets the alignment for the caption.
CellPadding	Gets or sets the number of spaces between the data and the cell border.
CellSpacing	Gets or sets the space between cells.

DayHeaderStyle	Gets the style properties for the section that displays the day of the week.
DayNameFormat	Gets or sets format of days of the week.
DayStyle	Gets the style properties for the days in the displayed month.
FirstDayOfWeek	Gets or sets the day of week to display in the first column.
NextMonthText	Gets or sets the text for next month navigation control. The default value is >.
NextPrevFormat	Gets or sets the format of the next and previous month navigation control.
OtherMonthDayStyle	Gets the style properties for the days on the Calendar control that are not in the displayed month.
PrevMonthText	Gets or sets the text for previous month navigation control. The default value is <.
SelectedDate	Gets or sets the selected date.
SelectedDates	Gets a collection of DateTime objects representing the selected dates.
SelectedDayStyle	Gets the style properties for the selected dates.
SelectionMode	Gets or sets the selection mode that specifies whether the user can select a single day, a week or an entire month.
SelectMonthText	Gets or sets the text for the month selection element in the selector column.

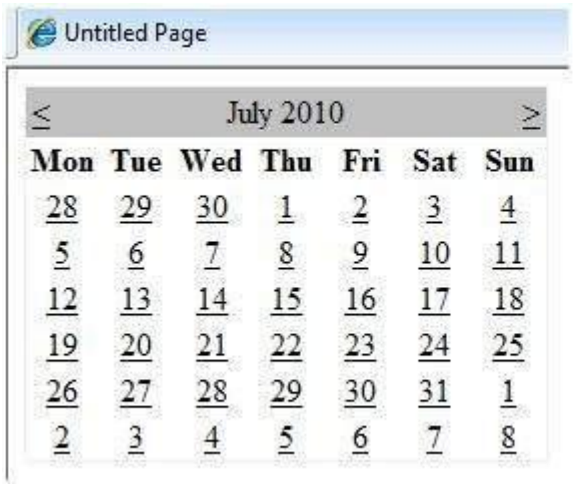
SelectorStyle	Gets the style properties for the week and month selector column.
SelectWeekText	Gets or sets the text displayed for the week selection element in the selector column.
ShowDayHeader	Gets or sets the value indicating whether the heading for the days of the week is displayed.
ShowGridLines	Gets or sets the value indicating whether the gridlines would be shown.
ShowNextPrevMonth	Gets or sets a value indicating whether next and previous month navigation elements are shown in the title section.
ShowTitle	Gets or sets a value indicating whether the title section is displayed.
TitleFormat	Gets or sets the format for the title section.
Titlestyle	Get the style properties of the title heading for the Calendar control.
TodayDayStyle	Gets the style properties for today's date on the Calendar control.
TodaysDate	Gets or sets the value for today's date.
UseAccessibleHeader	Gets or sets a value that indicates whether to render the table header <th> HTML element for the day headers instead of the table data <td> HTML element.
VisibleDate	Gets or sets the date that specifies the month to display.
WeekendDayStyle	Gets the style properties for the weekend dates on the Calendar control.

The Calendar control has the following three most important events that allow the developers to program the calendar control. They are:

Events	Description
SelectionChanged	It is raised when a day, a week or an entire month is selected.
DayRender	It is raised when each data cell of the calendar control is rendered.
VisibleMonthChanged	It is raised when user changes a month.

Working with the Calendar Control

Putting a bare-bone calendar control without any code behind file provides a workable calendar to a site, which shows the months and days of the year. It also allows navigation to next and previous months.



Mon	Tue	Wed	Thu	Fri	Sat	Sun
28	29	30	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	1
2	3	4	5	6	7	8

Calendar controls allow the users to select a single day, a week, or an entire month. This is done by using the SelectionMode property. This property has the following values:

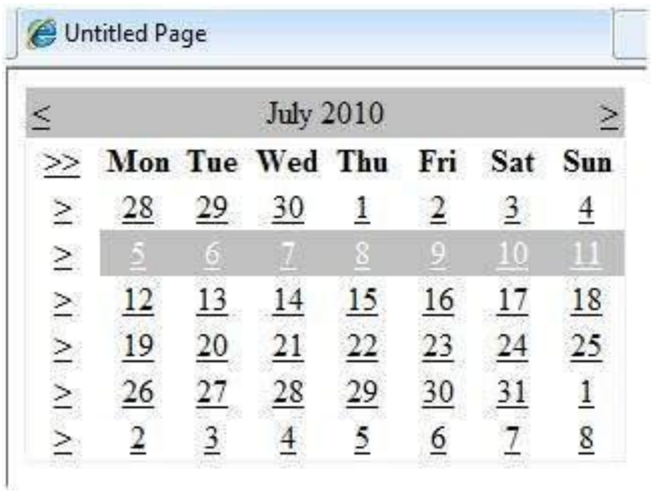
Properties	Description
Day	To select a single day.

DayWeek	To select a single day or an entire week.
DayWeekMonth	To select a single day, a week, or an entire month.
None	Nothing can be selected.

The syntax for selecting days:

```
<asp:Calender ID = "Calendar1" runat = "server" SelectionMode="DayWeekMonth">
</asp:Calender>
```

When the selection mode is set to the value DayWeekMonth, an extra column with the > symbol appears for selecting the week, and a >> symbol appears to the left of the days name for selecting the month.



Example

The following example demonstrates selecting a date and displays the date in a label:

The content file code is as follows:

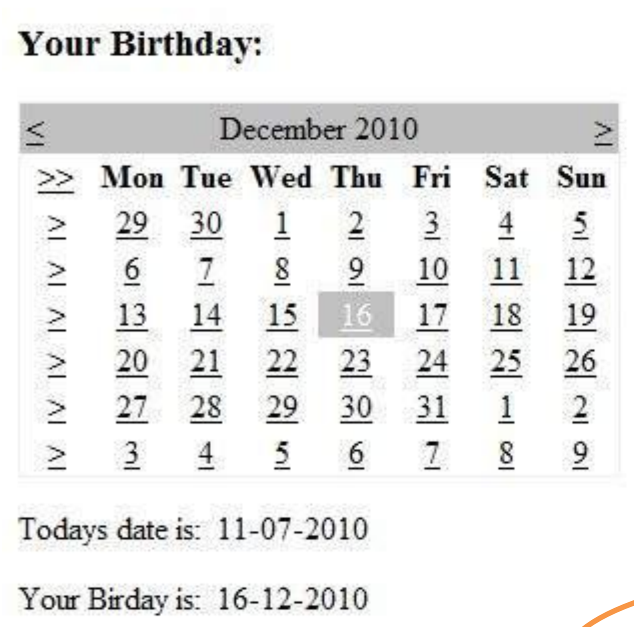
```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
Inherits="calendardemo._Default" %>

<!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>
      Untitled Page
    </title>
  </head>

  <body>
    <form id="form1" runat="server">

      <div>
        <h3> Your Birthday:</h3>
        <asp:Calendar ID="Calendar1" runat="server  SelectionMode="DayWeekMonth"
onselectionchanged="Calendar1_SelectionChanged">
        </asp:Calendar>
      </div>

      <p>Todays date is:
        <asp:Label ID="lblday" runat="server"></asp:Label>
      </p>

      <p>Your Birday is:
        <asp:Label ID="lblbday" runat="server"></asp:Label>
      </p>

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

The event handler for the event SelectionChanged:

```
protected void Calendar1_SelectionChanged(object sender, EventArgs e)
{
  lblday.Text = Calendar1.TodaysDate.ToShortDateString();
  lblbday.Text = Calendar1.SelectedDate.ToShortDateString();
}
```

When the file is run, it should produce the following output:



Your Birthday:

<			December 2010				>
>>	Mon	Tue	Wed	Thu	Fri	Sat	Sun
>	29	30	1	2	3	4	5
>	6	7	8	9	10	11	12
>	13	14	15	16	17	18	19
>	20	21	22	23	24	25	26
>	27	28	29	30	31	1	2
>	3	4	5	6	7	8	9

Todays date is: 11-07-2010

Your Birday is: 16-12-2010

FAQ: DESCRIBE ADROTATOR WITH A SUITABE EXAMPLES

ADROTATOR CONTROL

The AdRotator control randomly selects banner graphics from a list, which is specified in an external XML schedule file. This external XML schedule file is called the advertisement file.

The AdRotator control allows you to specify the advertisement file and the type of window that the link should follow in the AdvertisementFile and the Target property respectively.

The basic syntax of adding an AdRotator is as follows:

```
<asp:AdRotator  runat = "server" AdvertisementFile = "adfile.xml"  Target =  "_blank" />
```

Before going into the details of the AdRotator control and its properties, let us look into the construction of the advertisement file.

The Advertisement File

The advertisement file is an XML file, which contains the information about the advertisements to be displayed.

Extensible Markup Language (XML) is a W3C standard for text document markup. It is a text-based markup language that enables you to store data in a structured format by using meaningful tags. The term 'extensible' implies that you can extend your ability to describe a document by defining meaningful tags for the application.

XML is not a language in itself, like HTML, but a set of rules for creating new markup languages. It is a meta-markup language. It allows developers to create custom tag sets for special uses. It structures, stores, and transports the information.

Following is an example of XML file:

```
<BOOK>
  <NAME> Learn XML </NAME>
  <AUTHOR> Samuel Peterson </AUTHOR>
  <PUBLISHER> NSS Publications </PUBLISHER>
  <PRICE> $30.00</PRICE>
</BOOK>
```

Like all XML files, the advertisement file needs to be a structured text file with well-defined tags delineating the data. There are the following standard XML elements that are commonly used in the advertisement file:

Element	Description
Advertisements	Encloses the advertisement file.
Ad	Delineates separate ad.
ImageUrl	The path of image that will be displayed.
NavigateUrl	The link that will be followed when the user clicks the ad.
AlternateText	The text that will be displayed instead of the picture if it cannot be displayed.
Keyword	Keyword identifying a group of advertisements. This is used for filtering.
Impressions	The number indicating how often an advertisement will appear.
Height	Height of the image to be displayed.

Width	Width of the image to be displayed.

Apart from these tags, customs tags with custom attributes could also be included. The following code illustrates an advertisement file ads.xml:

```
<Advertisements>
  <Ad>
    <ImageUrl>rose1.jpg</ImageUrl>
    <NavigateUrl>http://www.1800flowers.com</NavigateUrl>
    <AlternateText>
      Order flowers, roses, gifts and more
    </AlternateText>
    <Impressions>20</Impressions>
    <Keyword>flowers</Keyword>
  </Ad>

  <Ad>
    <ImageUrl>rose2.jpg</ImageUrl>
    <NavigateUrl>http://www.babybouquets.com.au</NavigateUrl>
    <AlternateText>Order roses and flowers</AlternateText>
    <Impressions>20</Impressions>
    <Keyword>gifts</Keyword>
  </Ad>

  <Ad>
    <ImageUrl>rose3.jpg</ImageUrl>
    <NavigateUrl>http://www.flowers2moscow.com</NavigateUrl>
    <AlternateText>Send flowers to Russia</AlternateText>
    <Impressions>20</Impressions>
    <Keyword>russia</Keyword>
  </Ad>

  <Ad>
    <ImageUrl>rose4.jpg</ImageUrl>
    <NavigateUrl>http://www.edibleblooms.com</NavigateUrl>
    <AlternateText>Edible Blooms</AlternateText>
    <Impressions>20</Impressions>
    <Keyword>gifts</Keyword>
  </Ad>
```

```
</Advertisements>
```

Properties and Events of the AdRotator Class

The AdRotator class is derived from the WebControl class and inherits its properties. Apart from those, the AdRotator class has the following properties:

Properties	Description
AdvertisementFile	The path to the advertisement file.
AlternateTextFeild	The element name of the field where alternate text is provided. The default value is AlternateText.
DataMember	The name of the specific list of data to be bound when advertisement file is not used.
DataSource	Control from where it would retrieve data.
DataSourceID	Id of the control from where it would retrieve data.
Font	Specifies the font properties associated with the advertisement banner control.
ImageUrlField	The element name of the field where the URL for the image is provided. The default value is ImageUrl.
KeywordFilter	For displaying the keyword based ads only.
NavigateUrlField	The element name of the field where the URL to navigate to is provided. The default value is NavigateUrl.

Target	The browser window or frame that displays the content of the page linked.
UniqueID	Obtains the unique, hierarchically qualified identifier for the AdRotator control.

Following are the important events of the AdRotator class:

Events	Description
AdCreated	It is raised once per round trip to the server after creation of the control, but before the page is rendered
DataBinding	Occurs when the server control binds to a data source.
DataBound	Occurs after the server control binds to a data source.
Disposed	Occurs when a server control is released from memory, which is the last stage of the server control lifecycle when an ASP.NET page is requested
Init	Occurs when the server control is initialized, which is the first step in its lifecycle.
Load	Occurs when the server control is loaded into the Page object.
PreRender	Occurs after the Control object is loaded but prior to rendering.
Unload	Occurs when the server control is unloaded from memory.

Working with AdRotator Control

Create a new web page and place an AdRotator control on it.

```
<form id="form1" runat="server">
  <div>
    <asp:AdRotator ID="AdRotator1" runat="server" AdvertisementFile  ="~/ads.xml"
onadcreated="AdRotator1_AdCreated" />
  </div>
</form>
```

The ads.xml file and the image files should be located in the root directory of the web site.

Try to execute the above application and observe that each time the page is reloaded, the ad is changed.

****** END OF UNIT – III ******

FAQ: WHAT IS ADO.NET?

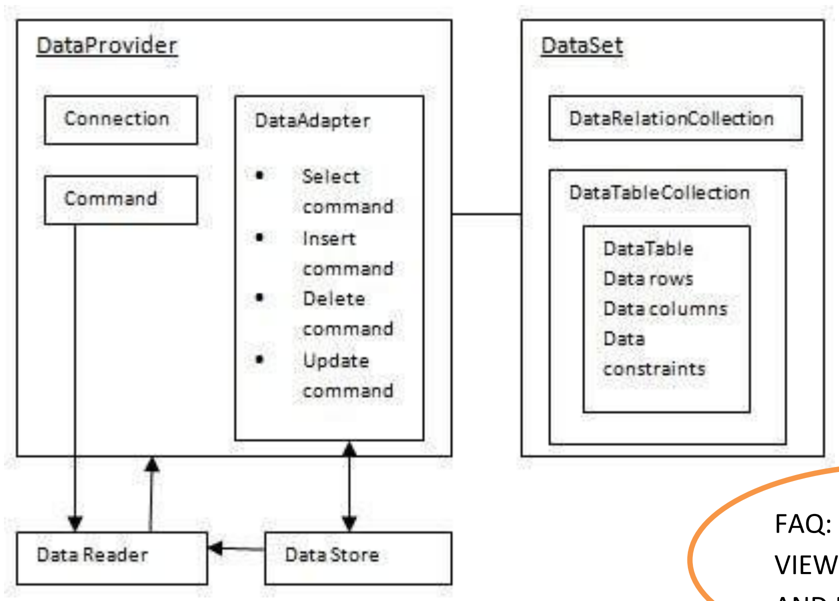
****** UNIT – IV ******

ADO.NET

ADO.NET is a set of computer software components that programmers can use to access data and data services from a database. ... **ADO.NET** is sometimes considered an evolution of ActiveX Data Objects (**ADO**) technology, ct.

ADO.NET provides a bridge between the front end controls and the back end database. The ADO.NET objects encapsulate all the data access operations and the controls interact with these objects to display data, thus hiding the details of movement of data.

The following figure shows the ADO.NET objects at a glance:



FAQ: DEFINE GRID VIEW IN ASP.NET AND DATASET

The **GridView** control displays the values of a data source in a table. Each column represents a field, while each row represents a record. The **GridView** control supports the following features: Binding to data source controls, such as SqlDataSource

The DataSet Class

The **dataset** represents a subset of the database. It does not have a continuous connection to the database. To update the database a reconnection is required. The DataSet contains DataTable objects and DataRelation objects. The DataRelation objects represent the relationship between two tables.

Following table shows some important properties of the DataSet class:

Properties	Description
CaseSensitive	Indicates whether string comparisons within the data tables are case-sensitive.
Container	Gets the container for the component.
DataSetName	Gets or sets the name of the current data set.
DefaultViewManager	Returns a view of data in the data set.
DesignMode	Indicates whether the component is currently in design mode.
EnforceConstraints	Indicates whether constraint rules are followed when attempting any update operation.
Events	Gets the list of event handlers that are attached to this component.
ExtendedProperties	Gets the collection of customized user information associated with the DataSet.
HasErrors	Indicates if there are any errors.
IsInitialized	Indicates whether the DataSet is initialized.
Locale	Gets or sets the locale information used to compare strings within the table.
Namespace	Gets or sets the namespace of the DataSet.

Prefix	Gets or sets an XML prefix that aliases the namespace of the DataSet.
Relations	Returns the collection of DataRelation objects.
Tables	Returns the collection of DataTable objects.

The following table shows some important methods of the DataSet class:

Methods	**Description**
AcceptChanges	Accepts all changes made since the DataSet was loaded or this method was called.
BeginInit	Begins the initialization of the DataSet. The initialization occurs at run time.
Clear	Clears data.
Clone	Copies the structure of the DataSet, including all DataTable schemas, relations, and constraints. Does not copy any data.
Copy	Copies both structure and data.
CreateDataReader()	Returns a DataTableReader with one result set per DataTable, in the same sequence as the tables appear in the Tables collection.
CreateDataReader(DataTable[])	Returns a DataTableReader with one result set per DataTable.

EndInit	Ends the initialization of the data set.
Equals(Object)	Determines whether the specified Object is equal to the current Object.
Finalize	Free resources and perform other cleanups.
GetChanges	Returns a copy of the DataSet with all changes made since it was loaded or the AcceptChanges method was called.
GetChanges(DataRowState)	Gets a copy of DataSet with all changes made since it was loaded or the AcceptChanges method was called, filtered by DataRowState.
GetDataSetSchema	Gets a copy of XmlSchemaSet for the DataSet.
GetObjectData	Populates a serialization information object with the data needed to serialize the DataSet.
GetType	Gets the type of the current instance.
GetXML	Returns the XML representation of the data.
GetXMLSchema	Returns the XSD schema for the XML representation of the data.
HasChanges()	Gets a value indicating whether the DataSet has changes, including new, deleted, or modified rows.

HasChanges(DataRowState)	Gets a value indicating whether the DataSet has changes, including new, deleted, or modified rows, filtered by DataRowState.
IsBinarySerialized	Inspects the format of the serialized representation of the DataSet.
Load(IDataReader, LoadOption, DataTable[])	Fills a DataSet with values from a data source using the supplied IDataReader, using an array of DataTable instances to supply the schema and namespace information.
Load(IDataReader, LoadOption, String[])	Fills a DataSet with values from a data source using the supplied IDataReader, using an array of strings to supply the names for the tables within the DataSet.
Merge()	Merges the data with data from another DataSet. This method has different overloaded forms.
ReadXML()	Reads an XML schema and data into the DataSet. This method has different overloaded forms.
ReadXMLSchema(0)	Reads an XML schema into the DataSet. This method has different overloaded forms.
RejectChanges	Rolls back all changes made since the last call to AcceptChanges.
WriteXML()	Writes an XML schema and data from the DataSet. This method has different overloaded forms.

WriteXMLSchema()	Writes the structure of the DataSet as an XML schema. This method has different overloaded forms.

The DataTable Class

The DataTable class represents the tables in the database. It has the following important properties; most of these properties are read only properties except the PrimaryKey property:

Properties	**Description**
ChildRelations	Returns the collection of child relationship.
Columns	Returns the Columns collection.
Constraints	Returns the Constraints collection.
DataSet	Returns the parent DataSet.
DefaultView	Returns a view of the table.
ParentRelations	Returns the ParentRelations collection.
PrimaryKey	Gets or sets an array of columns as the primary key for the table.
Rows	Returns the Rows collection.

The following table shows some important methods of the **DataTable** class:

Methods	Description
AcceptChanges	Commits all changes since the last AcceptChanges.
Clear	Clears all data from the table.
GetChanges	Returns a copy of the DataTable with all changes made since the AcceptChanges method was called.
GetErrors	Returns an array of rows with errors.
ImportRows	Copies a new row into the table.
LoadDataRow	Finds and updates a specific row, or creates a new one, if not found any.
Merge	Merges the table with another DataTable.
NewRow	Creates a new DataRow.
RejectChanges	Rolls back all changes made since the last call to AcceptChanges.
Reset	Resets the table to its original state.
Select	Returns an array of DataRow objects.

The **DataRow** Class

The DataRow object represents a row in a table. It has the following important properties:

Properties	Description
HasErrors	Indicates if there are any errors.
Items	Gets or sets the data stored in a specific column.
ItemArrays	Gets or sets all the values for the row.
Table	Returns the parent table.

The following table shows some important methods of the DataRow class:

Methods	Description
AcceptChanges	Accepts all changes made since this method was called.
BeginEdit	Begins edit operation.
CancelEdit	Cancels edit operation.
Delete	Deletes the DataRow.
EndEdit	Ends the edit operation.
GetChildRows	Gets the child rows of this row.
GetParentRow	Gets the parent row.
GetParentRows	Gets parent rows of DataRow object.

RejectChanges	Rolls back all changes made since the last call to AcceptChanges.

The DataAdapter Object

The DataAdapter object acts as a mediator between the DataSet object and the database. This helps the Dataset to contain data from multiple databases or other data source.

The DataReader Object

The DataReader object is an alternative to the DataSet and DataAdapter combination. This object provides a connection oriented access to the data records in the database. These objects are suitable for read-only access, such as populating a list and then breaking the connection.

DbCommand and DbConnection Objects

The DbConnection object represents a connection to the data source. The connection could be shared among different command objects.

The DbCommand object represents the command or a stored procedure sent to the database from retrieving or manipulating data.

Example

So far, we have used tables and databases already existing in our computer. In this example, we will create a table, add column, rows and data into it and display the table using a GridView object.

The source file code is as given:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
Inherits="createdatabase._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>
         Untitled Page
      </title>
```

```
  </head>

  <body>
    <form id="form1" runat="server">

      <div>
        <asp:GridView ID="GridView1" runat="server">
        </asp:GridView>
      </div>

    </form>
  </body>

</html>
```

The code behind file is as given:

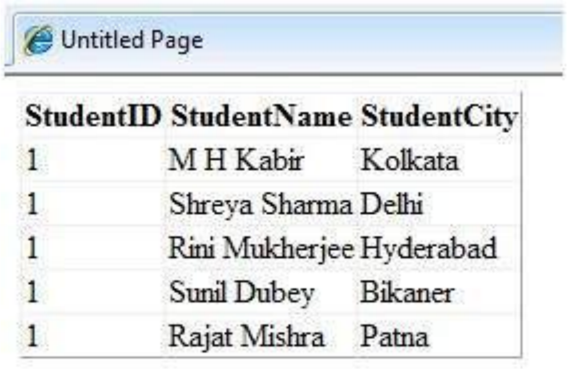
```
namespace createdatabase
{
  public partial class _Default : System.Web.UI.Page
  {
    protected void Page_Load(object sender, EventArgs e)
    {
      if (!IsPostBack)
      {
        DataSet ds = CreateDataSet();
        GridView1.DataSource = ds.Tables["Student"];
        GridView1.DataBind();
      }
    }

    private DataSet CreateDataSet()
    {
      //creating a DataSet object for tables
      DataSet dataset = new DataSet();

      // creating the student table
      DataTable Students = CreateStudentTable();
      dataset.Tables.Add(Students);
      return dataset;
    }
    private DataTable CreateStudentTable()
    {
      DataTable Students = new DataTable("Student");

      // adding columns
      AddNewColumn(Students, "System.Int32", "StudentID");
      AddNewColumn(Students, "System.String", "StudentName");
      AddNewColumn(Students, "System.String", "StudentCity");

      // adding rows
      AddNewRow(Students, 1, "M H Kabir", "Kolkata");
      AddNewRow(Students, 1, "Shreya Sharma", "Delhi");
      AddNewRow(Students, 1, "Rini Mukherjee", "Hyderabad");
      AddNewRow(Students, 1, "Sunil Dubey", "Bikaner");
      AddNewRow(Students, 1, "Rajat Mishra", "Patna");

      return Students;
    }

    private void AddNewColumn(DataTable table, string columnType, string  columnName)
    {
      DataColumn column = table.Columns.Add(columnName,  Type.GetType(columnType));
    }

    //adding data into the table
    private void AddNewRow(DataTable table, int id, string name, string city)
    {
      DataRow newrow = table.NewRow();
      newrow["StudentID"] = id;
      newrow["StudentName"] = name;
      newrow["StudentCity"] = city;
      table.Rows.Add(newrow);
    }
  }
}
```

When you execute the program, observe the following:

- The application first creates a data set and binds it with the grid view control using the DataBind() method of the GridView control.

- The Createdataset() method is a user defined function, which creates a new DataSet object and then calls another user defined method CreateStudentTable() to create the table and add it to the Tables collection of the data set.
- The CreateStudentTable() method calls the user defined methods AddNewColumn() and AddNewRow() to create the columns and rows of the table as well as to add data to the rows.

When the page is executed, it returns the rows of the table as shown:



StudentID	StudentName	StudentCity
1	M H Kabir	Kolkata
1	Shreya Sharma	Delhi
1	Rini Mukherjee	Hyderabad
1	Sunil Dubey	Bikaner
1	Rajat Mishra	Patna

FAQ: EXPLAIN ADO.NET TECNOLOGY WITH SIUTABLE EXAMPLES

ASP.NET allows the following sources of data to be accessed and used:

- Databases (e.g., Access, SQL Server, Oracle, MySQL)
- XML documents
- Business Objects
- Flat files

ASP.NET hides the complex processes of data access and provides much higher level of classes and objects through which data is accessed easily. These classes hide all complex coding for connection, data retrieving, data querying, and data manipulation.

ADO.NET is the technology that provides the bridge between various ASP.NET control objects and the backend data source. In this tutorial, we will look at data access and working with the data in brief.

Retrieve and display data

It takes two types of data controls to retrieve and display data in ASP.NET:

- **A data source control** - It manages the connection to the data, selection of data, and other jobs such as paging and caching of data etc.

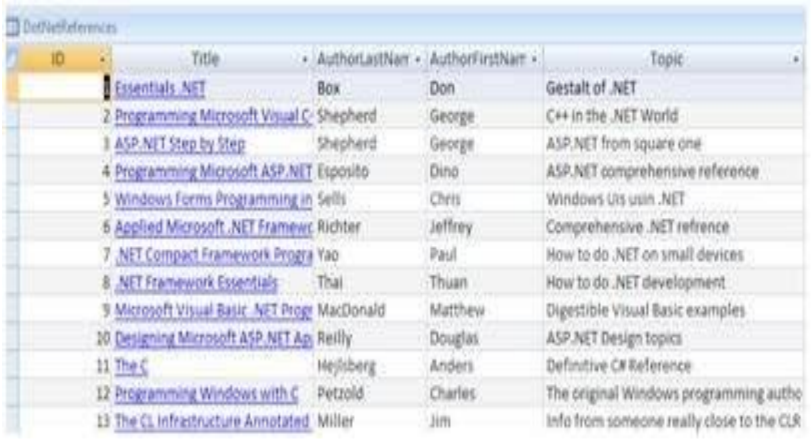
- **A data view control -** It binds and displays the data and allows data manipulation.

We will discuss the data binding and data source controls in detail later. In this section, we will use a SqlDataSource control to access data and a GridView control to display and manipulate data in this chapter.

We will also use an Access database, which contains the details about .Net books available in the market. Name of our database is ASPDotNetStepByStep.mdb and we will use the data table DotNetReferences.

The table has the following columns: ID, Title, AuthorFirstName, AuthorLastName, Topic, and Publisher.

Here is a snapshot of the data table:

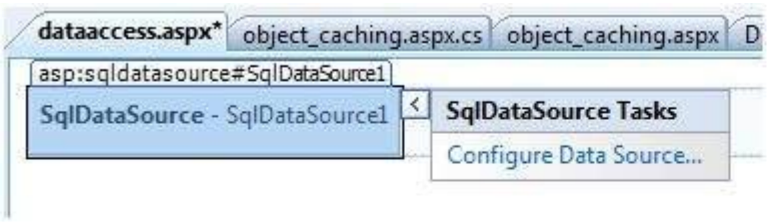


DotNetReferences

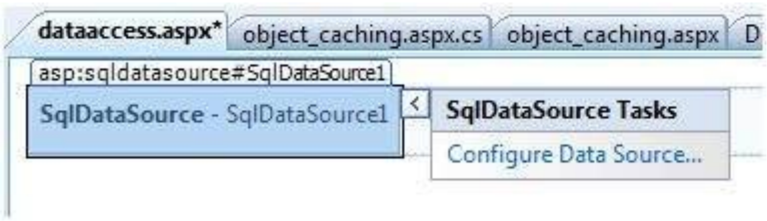
ID	Title	AuthorLastNam	AuthorFirstNam	Topic
1	Essentials .NET	Box	Don	Gestalt of .NET
2	Programming Microsoft Visual C	Shepherd	George	C++ in the .NET World
3	ASP.NET Step by Step	Shepherd	George	ASP.NET from square one
4	Programming Microsoft ASP.NET	Esposito	Dino	ASP.NET comprehensive reference
5	Windows Forms Programming in	Sells	Chris	Windows UIs usin .NET
6	Applied Microsoft .NET Framewc	Richter	Jeffrey	Comprehensive .NET refrence
7	.NET Compact Framework Progra	Yao	Paul	How to do .NET on small devices
8	.NET Framework Essentials	Thai	Thuan	How to do .NET development
9	Microsoft Visual Basic .NET Progr	MacDonald	Matthew	Digestible Visual Basic examples
10	Designing Microsoft ASP.NET Ap	Reilly	Douglas	ASP.NET Design topics
11	The C	Hejlsberg	Anders	Definitive C# Reference
12	Programming Windows with C	Petzold	Charles	The original Windows programming autho
13	The CL Infrastructure Annotated	Miller	Jim	Info from someone really close to the CLR

Let us directly move to action, take the following steps:

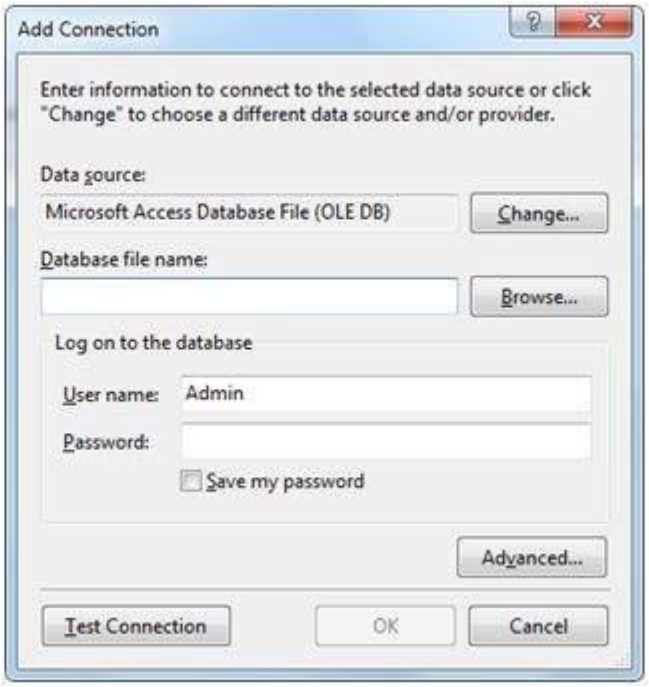
(1) Create a web site and add a SqlDataSourceControl on the web form.



(2) Click on the Configure Data Source option.



(3) Click on the New Connection button to establish connection with a database.


Add Connection

Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.

Data source: Microsoft Access Database File (OLE DB)

Database file name:

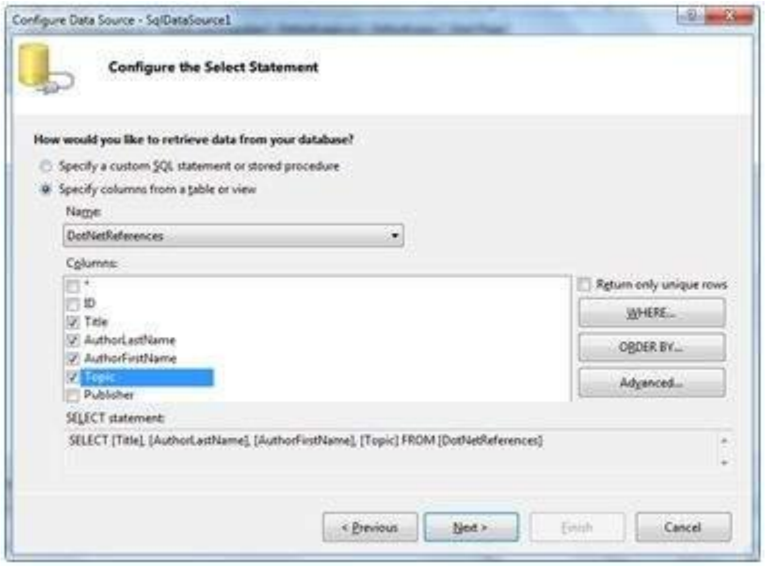
Log on to the database

User name: Admin

Password:

☐ Save my password

(4) Once the connection is set up, you may save it for further use. At the next step, you are asked to configure the select statement:


Configure Data Source - SqlDataSource1

Configure the Select Statement

How would you like to retrieve data from your database?

○ Specify a custom SQL statement or stored procedure
● Specify columns from a table or view

Name: DotNetReferences

Columns:

☐ *
☐ ID
☑ Title
☑ AuthorLastName
☑ AuthorFirstName
☑ Topic
☐ Publisher

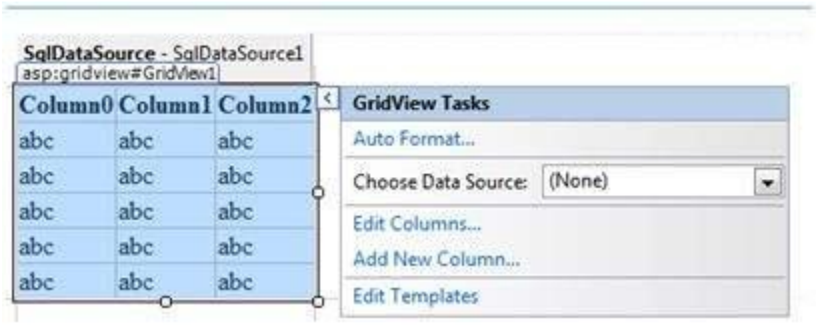
☐ Return only unique rows

SELECT statement:

```
SELECT [Title], [AuthorLastName], [AuthorFirstName], [Topic] FROM [DotNetReferences]
```

(5) Select the columns and click next to complete the steps. Observe the WHERE, ORDER BY, and the Advanced buttons. These buttons allow you to provide the where clause, order by clause, and specify the insert, update, and delete commands of SQL respectively. This way, you can manipulate the data.

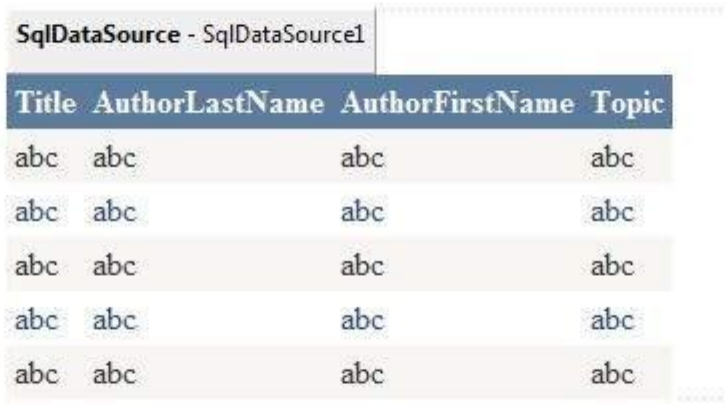
(6) Add a GridView control on the form. Choose the data source and format the control using AutoFormat option.



Column0	Column1	Column2
abc	abc	abc
abc	abc	abc
abc	abc	abc
abc	abc	abc
abc	abc	abc

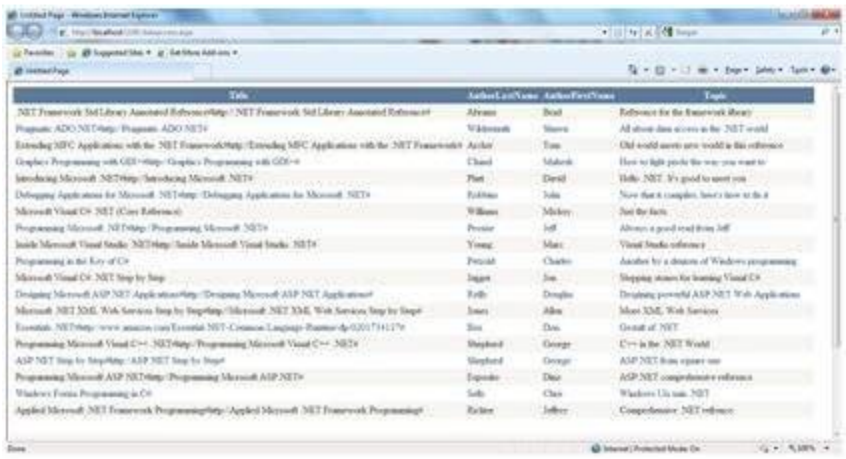
Choose Data Source: (None)

(7) After this the formatted GridView control displays the column headings, and the application is ready to execute.



Title	AuthorLastName	AuthorFirstName	Topic
abc	abc	abc	abc
abc	abc	abc	abc
abc	abc	abc	abc
abc	abc	abc	abc
abc	abc	abc	abc

(8) Finally execute the application.



Title	AuthorLastName	AuthorFirstName	Topic
.NET Framework Std Library Annotated Reference#http://.NET Framework Std Library Annotated Reference#	Abrams	Brad	Reference for the framework library
Pragmatic ADO.NET#http://Pragmatic ADO.NET#	Wildermuth	Shawn	All about data access in the .NET world
Extending MFC Applications with the .NET Framework#http://Extending MFC Applications with the .NET Framework#	Archer	Tom	Old world meets new world in this reference
Graphics Programming with GDI+#http://Graphics Programming with GDI+#	Chand	Mahesh	How to light pixels the way you want to
Introducing Microsoft .NET#http://Introducing Microsoft .NET#	Platt	David	Hello .NET. It's good to meet you
Debugging Applications for Microsoft .NET#http://Debugging Applications for Microsoft .NET#	Robbins	John	Now that it compiles, here's how to do it
Microsoft Visual C# .NET (Core Reference)	Williams	Mickey	Just the facts
Programming Microsoft .NET#http://Programming Microsoft .NET#	Prosise	Jeff	Always a good read from Jeff
Inside Microsoft Visual Studio .NET#http://Inside Microsoft Visual Studio .NET#	Young	Marc	Visual Studio reference
Programming in the Key of C#	Petzold	Charles	Another by a demon of Windows programming
Microsoft Visual C# .NET Step by Step	Jagger	Jon	Stepping stones for learning Visual C#
Designing Microsoft ASP.NET Applications#http://Designing Microsoft ASP.NET Applications#	Reilly	Douglas	Designing powerful ASP.NET Web Applications
Microsoft .NET XML Web Services Step by Step#http://Microsoft .NET XML Web Services Step by Step#	Jones	Allen	More XML Web Services
Essentials .NET#http://www.amazon.com/Essential-NET-Common-Language-Runtime/dp/0201734117#	Box	Don	Gestalt of .NET
Programming Microsoft Visual C++ .NET#http://Programming Microsoft Visual C++ .NET#	Shepherd	George	C++ in the .NET World
ASP.NET Step by Step#http://ASP.NET Step by Step#	Shepherd	George	ASP.NET from square one
Programming Microsoft ASP.NET#http://Programming Microsoft ASP.NET#	Esposito	Dino	ASP.NET comprehensive reference
Windows Forms Programming in C#	Sells	Chris	Windows UIs with .NET
Applied Microsoft .NET Framework Programming#http://Applied Microsoft .NET Framework Programming#	Richter	Jeffrey	Comprehensive .NET reference

The content file code is as given:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="dataaccess.aspx.cs"
  Inherits="datacaching.WebForm1" %>

<!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>
      Untitled Page
    </title>
  </head>

  <body>
    <form id="form1" runat="server">
      <div>

        <asp:SqlDataSource ID="SqlDataSource1" runat="server"
          ConnectionString= "<%$
ConnectionStrings:ASPDotNetStepByStepConnectionString%>"
          ProviderName= "<%$ ConnectionStrings:
            ASPDotNetStepByStepConnectionString.ProviderName %>"
          SelectCommand="SELECT [Title], [AuthorLastName],
            [AuthorFirstName], [Topic] FROM [DotNetReferences]">
        </asp:SqlDataSource>
```

```
         <asp:GridView ID="GridView1" runat="server"
           AutoGenerateColumns="False" CellPadding="4"
           DataSourceID="SqlDataSource1" ForeColor="#333333"
           GridLines="None">
           <RowStyle BackColor="#F7F6F3" ForeColor="#333333" />

           <Columns>
             <asp:BoundField DataField="Title" HeaderText="Title"
               SortExpression="Title" />
             <asp:BoundField DataField="AuthorLastName"
               HeaderText="AuthorLastName" SortExpression="AuthorLastName" />
             <asp:BoundField DataField="AuthorFirstName"
               HeaderText="AuthorFirstName" SortExpression="AuthorFirstName" />
             <asp:BoundField DataField="Topic"
               HeaderText="Topic" SortExpression="Topic" />
           </Columns>
           <FooterStyle BackColor="#5D7B9D"
             Font-Bold="True" ForeColor="White" />
           <PagerStyle BackColor="#284775"
             ForeColor="White" HorizontalAlign="Center" />
           <SelectedRowStyle BackColor="#E2DED6"
             Font-Bold="True" ForeColor="#333333" />
           <HeaderStyle BackColor="#5D7B9D" Font-Bold="True"
             ForeColor="White" />
           <EditRowStyle BackColor="#999999" />
           <AlternatingRowStyle BackColor="White" ForeColor="#284775" />
         </asp:GridView>
       </div>
     </form>
   </body>
</html>
```

A data source control interacts with the data-bound controls and hides the complex data binding processes. These are the tools that provide data to the data bound controls and support execution of operations like insertions, deletions, sorting, and updates.

Each data source control wraps a particular data provider-relational databases, XML documents, or custom classes and helps in:

- Managing connection
- Selecting data
- Managing presentation aspects like paging, caching, etc.
- Manipulating data

There are many data source controls available in ASP.NET for accessing data from SQL Server, from ODBC or OLE DB servers, from XML files, and from business objects.

Based on type of data, these controls could be divided into two categories:

- Hierarchical data source controls
- Table-based data source controls

The data source controls used for hierarchical data are:

- **XMLDataSource** - It allows binding to XML files and strings with or without schema information.
- **SiteMapDataSource** - It allows binding to a provider that supplies site map information.

The data source controls used for tabular data are:

Data source controls	**Description**
SqlDataSource	It represents a connection to an ADO.NET data provider that returns SQL data, including data sources accessible via OLEDB and ODBC.
ObjectDataSource	It allows binding to a custom .Net business object that returns data.
LinqdataSource	It allows binding to the results of a Linq-to-SQL query (supported by ASP.NET 3.5 only).
AccessDataSource	It represents connection to a Microsoft Access database.

Data Source Views

Data source views are objects of the DataSourceView class. Which represent a customized view of data for different data operations such as sorting, filtering, etc.

The DataSourceView class serves as the base class for all data source view classes, which define the capabilities of data source controls.

The following table provides the properties of the DataSourceView class:

Properties	Description
CanDelete	Indicates whether deletion is allowed on the underlying data source.
CanInsert	Indicates whether insertion is allowed on the underlying data source.
CanPage	Indicates whether paging is allowed on the underlying data source.
CanRetrieveTotalRowCount	Indicates whether total row count information is available.
CanSort	Indicates whether the data could be sorted.
CanUpdate	Indicates whether updates are allowed on the underlying data source.
Events	Gets a list of event-handler delegates for the data source view.
Name	Name of the view.

The following table provides the methods of the DataSourceView class:

Methods	Description
CanExecute	Determines whether the specified command can be executed.
ExecuteCommand	Executes the specific command.

ExecuteDelete	Performs a delete operation on the list of data that the DataSourceView object represents.
ExecuteInsert	Performs an insert operation on the list of data that the DataSourceView object represents.
ExecuteSelect	Gets a list of data from the underlying data storage.
ExecuteUpdate	Performs an update operation on the list of data that the DataSourceView object represents.
Delete	Performs a delete operation on the data associated with the view.
Insert	Performs an insert operation on the data associated with the view.
Select	Returns the queried data.
Update	Performs an update operation on the data associated with the view.
OnDataSourceViewChanged	Raises the DataSourceViewChanged event.
RaiseUnsupportedCapabilitiesError	Called by the RaiseUnsupportedCapabilitiesError method to compare the capabilities requested for an ExecuteSelect operation against those that the view supports.

The SqlDataSource Control

FAQ: WHAT IS OLEDB?

The SqlDataSource control represents a connection to a relational database such as SQL Server or Oracle database, or data accessible through OLEDB(Object Linking Embedding Database) or Open Database Connectivity (ODBC). Connection to data is made through two important properties ConnectionString and ProviderName.

The following code snippet provides the basic syntax of the control:

```
<asp:SqlDataSource runat="server" ID="MySqlSource"
  ProviderName='<%$ ConnectionStrings:LocalNWind.ProviderName  %>'
  ConnectionString='<%$ ConnectionStrings:LocalNWind %>'
  SelectionCommand= "SELECT * FROM EMPLOYEES" />

<asp:GridView ID="GridView1" runat="server" DataSourceID="MySqlSource" />
```

Configuring various data operations on the underlying data depends upon the various properties (property groups) of the data source control.

The following table provides the related sets of properties of the SqlDataSource control, which provides the programming interface of the control:

Property Group	Description
DeleteCommand, DeleteParameters, DeleteCommandType	Gets or sets the SQL statement, parameters, and type for deleting rows in the underlying data.
FilterExpression, FilterParameters	Gets or sets the data filtering string and parameters.
InsertCommand, InsertParameters, InsertCommandType	Gets or sets the SQL statement, parameters, and type for inserting rows in the underlying database.
SelectCommand, SelectParameters, SelectCommandType	Gets or sets the SQL statement, parameters, and type for retrieving rows from the underlying database.
SortParameterName	Gets or sets the name of an input parameter that the command's stored procedure will use to sort data.
UpdateCommand,	Gets or sets the SQL statement, parameters, and type for updating rows in the underlying data store.

UpdateParameters, UpdateCommandType	

FAQ: DEFINE CONNECTIONSTRING, PROVIDER NAME AND SQL COMMAND

The following code snippet shows a data source control enabled for data manipulation:

```
<asp:SqlDataSource runat="server" ID= "MySqlSource"
  ProviderName='<%$ ConnectionStrings:LocalNWind.ProviderName  %>'
  ConnectionString=' <%$ ConnectionStrings:LocalNWind %>'
  SelectCommand= "SELECT * FROM EMPLOYEES"
  UpdateCommand= "UPDATE EMPLOYEES SET LASTNAME=@lame"
  DeleteCommand= "DELETE FROM EMPLOYEES WHERE EMPLOYEEID=@eid"
  FilterExpression= "EMPLOYEEID > 10">
  .....
  .....
</asp:SqlDataSource>
```

The ObjectDataSource Control

The **ObjectDataSource** Control enables user-defined classes to associate the output of their methods to data bound controls. The programming interface of this class is almost same as the SqlDataSource control.

Following are two important aspects of binding business objects:

- The bindable class should have a default constructor, it should be stateless, and have methods that can be mapped to select, update, insert, and delete semantics.
- The object must update one item at a time, batch operations are not supported.

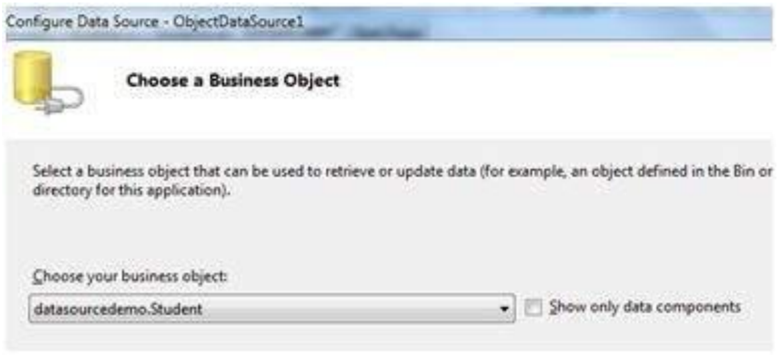
Let us go directly to an example to work with this control. The student class is the class to be used with an object data source. This class has three properties: a student id, name, and city. It has a default constructor and a GetStudents method for retrieving data.

The student class:

```
public class Student
{
  public int StudentID { get; set; }
  public string Name { get; set; }
  public string City { get; set; }

  public Student()
  { }

  public DataSet GetStudents()
  {
    DataSet ds = new DataSet();
    DataTable dt = new DataTable("Students");

    dt.Columns.Add("StudentID", typeof(System.Int32));
    dt.Columns.Add("StudentName", typeof(System.String));
    dt.Columns.Add("StudentCity", typeof(System.String));
    dt.Rows.Add(new object[] { 1, "M. H. Kabir", "Calcutta" });
    dt.Rows.Add(new object[] { 2, "Ayan J. Sarkar", "Calcutta" });
    ds.Tables.Add(dt);

    return ds;
  }
}
```

Take the following steps to bind the object with an object data source and retrieve data:

- Create a new web site.
- Add a class (Students.cs) to it by right clicking the project from the Solution Explorer, adding a class template, and placing the above code in it.
- Build the solution so that the application can use the reference to the class.
- Place an object data source control in the web form.
- Configure the data source by selecting the object.



Configure Data Source - ObjectDataSource1

Choose a Business Object

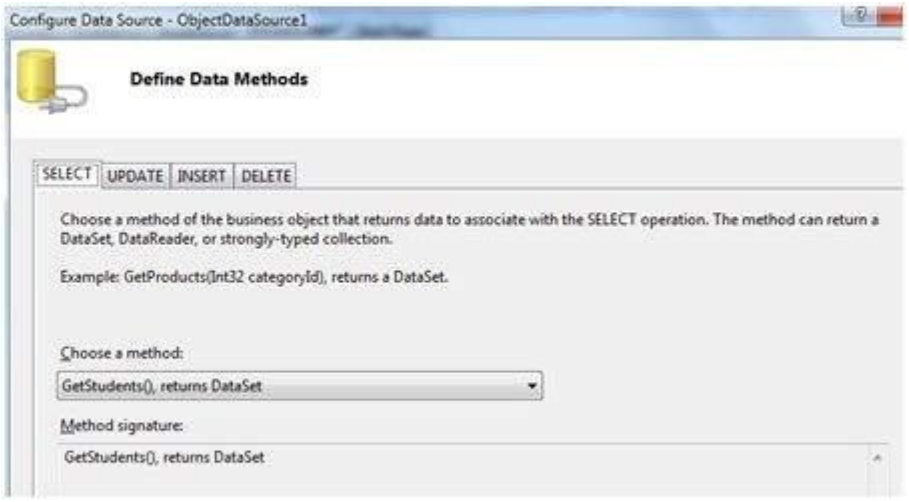
Select a business object that can be used to retrieve or update data (for example, an object defined in the Bin or
directory for this application).

Choose your business object:

datasourcedemo.Student

Show only data components

- Select a data method(s) for different operations on data. In this example, there is only one method.



Configure Data Source - ObjectDataSource1

Define Data Methods

SELECT | UPDATE | INSERT | DELETE

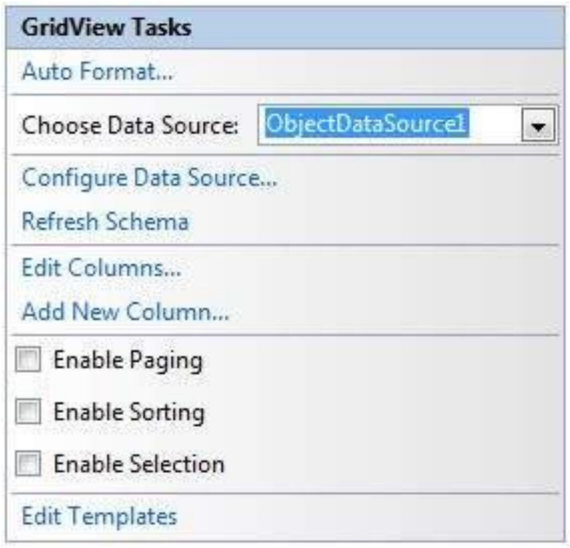
Choose a method of the business object that returns data to associate with the SELECT operation. The method can return a DataSet, DataReader, or strongly-typed collection.

Example: GetProducts(Int32 categoryId), returns a DataSet.

Choose a method: GetStudents(), returns DataSet

Method signature: GetStudents(), returns DataSet

- Place a data bound control such as grid view on the page and select the object data source as its underlying data source.



GridView Tasks

Auto Format...

Choose Data Source: ObjectDataSource1

Configure Data Source...

Refresh Schema

Edit Columns...

Add New Column...

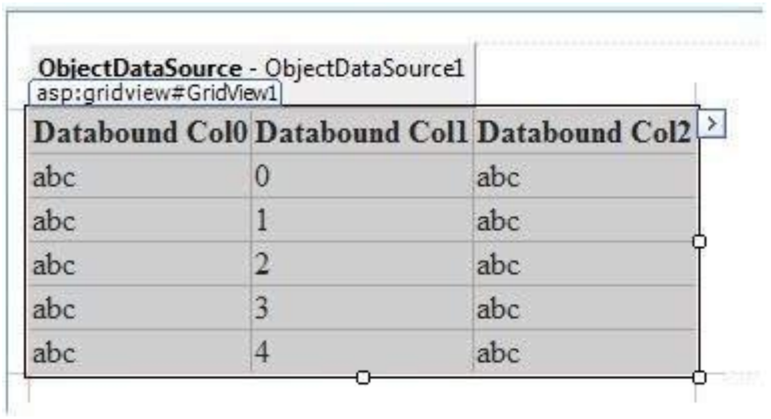
Enable Paging

Enable Sorting

Enable Selection

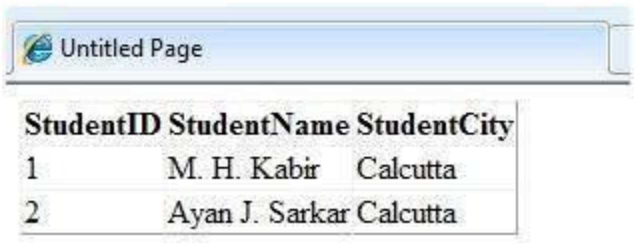
Edit Templates

- At this stage, the design view should look like the following:



Databound Col0	Databound Col1	Databound Col2
abc	0	abc
abc	1	abc
abc	2	abc
abc	3	abc
abc	4	abc

- Run the project, it retrieves the hard coded tuples from the students class.



StudentID	StudentName	StudentCity
1	M. H. Kabir	Calcutta
2	Ayan J. Sarkar	Calcutta

The AccessDataSource Control

The AccessDataSource control represents a connection to an Access database. It is based on the SqlDataSource control and provides simpler programming interface. The following code snippet provides the basic syntax for the data source:

```
<asp:AccessDataSource ID="AccessDataSource1 runat="server"
  DataFile="~/App_Data/ASPDotNetStepByStep.mdb" SelectCommand="SELECT * FROM
[DotNetReferences]">
</asp:AccessDataSource>
```

The AccessDataSource control opens the database in read-only mode. However, it can also be used for performing insert, update, or delete operations. This is done using the ADO.NET commands and parameter collection.

Updates are problematic for Access databases from within an ASP.NET application because an Access database is a plain file and the default account of the ASP.NET application might not have the permission to write to the database file.

****** END OF UNIT- IV ******

FAQ: DEFINE DATA BINDING

****** UNIT – V ******

DATA BINDING

Every ASP.NET web form control inherits the **DataBind** method from its parent Control class, which gives it an inherent capability to bind data to at least one of its properties. This is known as **simple data binding** or **inline data binding**.

Simple data binding involves attaching any collection (item collection) which implements the IEnumerable interface, or the DataSet and DataTable classes to the DataSource property of the control.

On the other hand, some controls can bind records, lists, or columns of data into their structure through a DataSource control. These controls derive from the BaseDataBoundControl class. This is called **declarative data binding**.

The data source controls help the data-bound controls implement functionalities such as, sorting, paging, and editing data collections.

The BaseDataBoundControl is an abstract class, which is inherited by two more abstract classes:

FAQ: EXPLAIN THE DIFFERENT TYPES OF DATABINDINGIN DETAIL

- DataBoundControl
- HierarchicalDataBoundControl

The abstract class DataBoundControl is again inherited by two more abstract classes:

- ListControl
- CompositeDataBoundControl

The controls capable of simple data binding are derived from the ListControl abstract class and these controls are:

- BulletedList
- CheckBoxList
- DropDownList
- ListBox
- RadioButtonList

The controls capable of declarative data binding (a more complex data binding) are derived from the abstract class CompositeDataBoundControl. These controls are:

- DetailsView
- FormView
- GridView
- RecordList

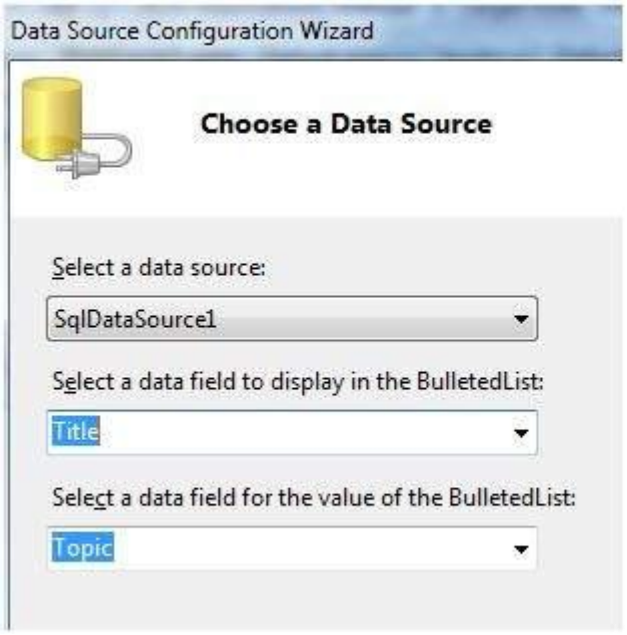
Simple Data Binding

Simple data binding involves the read-only selection lists. These controls can bind to an array list or fields from a database. Selection lists takes two values from the database or the data source; one value is displayed by the list and the other is considered as the value corresponding to the display.

Let us take up a small example to understand the concept. Create a web site with a bulleted list and a SqlDataSource control on it. Configure the data source control to retrieve two values from your database (we use the same DotNetReferences table as in the previous chapter).

Choosing a data source for the bulleted list control involves:

- Selecting the data source control
- Selecting a field to display, which is called the data field
- Selecting a field for the value

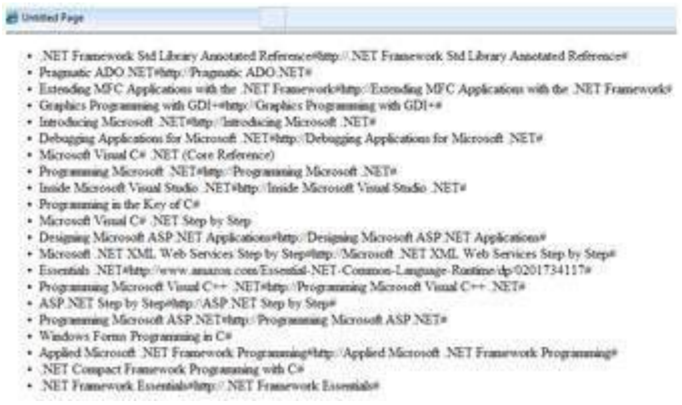


Select a data source: SqlDataSource1

Select a data field to display in the BulletedList: Title

Select a data field for the value of the BulletedList: Topic

When the application is executed, check that the entire title column is bound to the bulleted list and displayed.



- .NET Framework Std Library Annotated Reference#http://.NET Framework Std Library Annotated Reference#
- Pragmatic ADO.NET#http://Pragmatic ADO.NET#
- Extending MFC Applications with the .NET Framework#http://Extending MFC Applications with the .NET Framework#
- Graphics Programming with GDI+#http://Graphics Programming with GDI+#
- Introducing Microsoft .NET#http://Introducing Microsoft .NET#
- Debugging Applications for Microsoft .NET#http://Debugging Applications for Microsoft .NET#
- Microsoft Visual C# .NET (Core Reference)
- Programming Microsoft .NET#http://Programming Microsoft .NET#
- Inside Microsoft Visual Studio .NET#http://Inside Microsoft Visual Studio .NET#
- Programming in the Key of C#
- Microsoft Visual C# .NET Step by Step
- Designing Microsoft ASP.NET Applications#http://Designing Microsoft ASP.NET Applications#
- Microsoft .NET XML Web Services Step by Step#http://Microsoft .NET XML Web Services Step by Step#
- Essentials .NET#http://www.amazon.com/Essential-NET-Common-Language-Runtime/dp/0201734117#
- Programming Microsoft Visual C++ .NET#http://Programming Microsoft Visual C++ .NET#
- ASP.NET Step by Step#http://ASP.NET Step by Step#
- Programming Microsoft ASP.NET#http://Programming Microsoft ASP.NET#
- Windows Forms Programming in C#
- Applied Microsoft .NET Framework Programming#http://Applied Microsoft .NET Framework Programming#
- .NET Compact Framework Programming with C#
- .NET Framework Essentials#http://.NET Framework Essentials#

Declarative Data Binding

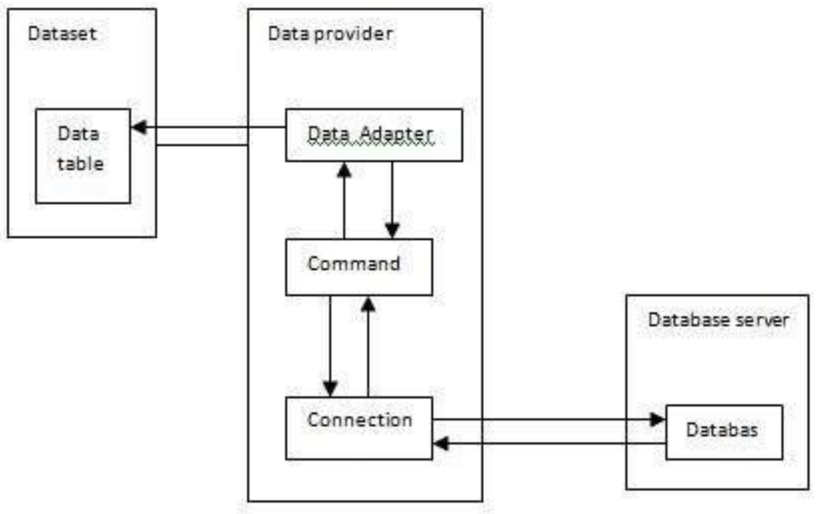
We have already used declarative data binding in the previous tutorial using GridView control. The other composite data bound controls capable of displaying and manipulating data in a tabular manner are the DetailsView, FormView, and RecordList control.

In the next tutorial, we will look into the technology for handling database, i.e, ADO.NET.

However, the data binding involves the following objects:

- A dataset that stores the data retrieved from the database.
- The data provider, which retrieves data from the database by using a command over a connection.
- The data adapter that issues the select statement stored in the command object; it is also capable of update the data in a database by issuing Insert, Delete, and Update statements.

Relation between the data binding objects:



Example

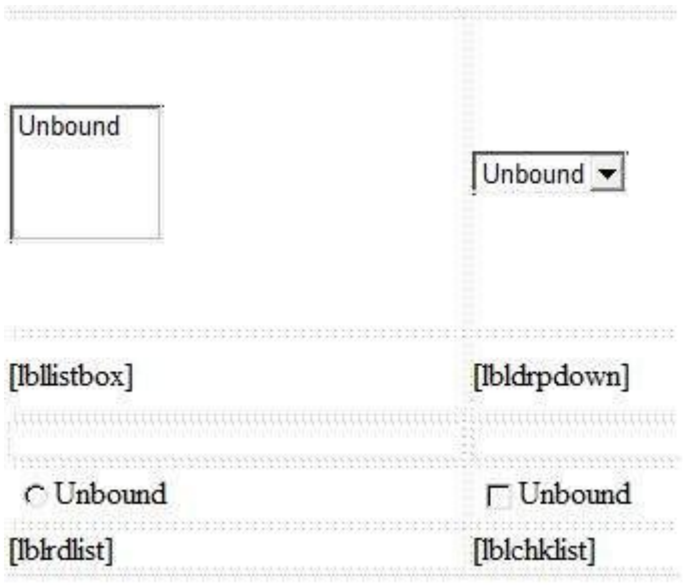
Let us take the following steps:

Step (1) : Create a new website. Add a class named booklist by right clicking on the solution name in the Solution Explorer and choosing the item ‘Class’ from the ‘Add Item’ dialog box. Name it as booklist.cs.

```
using System;
using System.Data;
using System.Configuration;
using System.Linq;

using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;

using System.Xml.Linq;

namespace databinding
{
  public class booklist
  {
    protected String bookname;
    protected String authorname;
    public booklist(String bname, String aname)
    {
      this.bookname = bname;
      this.authorname = aname;

    }

    public String Book
    {
      get
      {
        return this.bookname;
      }
      set
```

```
        {
          this.bookname = value;
        }
      }

      public String Author
      {
        get
        {
          return this.authorname;
        }
        set
        {
          this.authorname = value;
        }
      }
    }
}
```

Step (2) : Add four list controls on the page a list box control, a radio button list, a check box list, and a drop down list and four labels along with these list controls. The page should look like this in design view:



The source file should look as the following:

```
<form id=”form1” runat=”server”>
  <div>
   <table style=”width: 559px”>
     <tr>
       <td style=”width: 228px; height: 157px;”>
         <asp:ListBox ID=”ListBox1” runat=”server” AutoPostBack=”True”
           OnSelectedIndexChanged=”ListBox1_SelectedIndexChanged”>
         </asp:ListBox>
       </td>

       <td style=”height: 157px”>
         <asp:DropDownList ID=”DropDownList1” runat=”server”
           AutoPostBack=”True”
 OnSelectedIndexChanged=”DropDownList1_SelectedIndexChanged”>
         </asp:DropDownList>
       </td>
     </tr>

     <tr>
       <td style=”width: 228px; height: 40px;”>
         <asp:Label ID=”lbllistbox” runat=”server”></asp:Label>
       </td>

       <td style=”height: 40px”>
         <asp:Label ID=”lbldrpdown” runat=”server”>
         </asp:Label>
       </td>
     </tr>

     <tr>
       <td style=”width: 228px; height: 21px”>
       </td>

       <td style=”height: 21px”>
       </td>
     </tr>

     <tr>
       <td style=”width: 228px; height: 21px”>
         <asp:RadioButtonList ID=”RadioButtonList1” runat=”server”
```

```
            AutoPostBack=”True”
OnSelectedIndexChanged=”RadioButtonList1_SelectedIndexChanged”>
          </asp:RadioButtonList>
        </td>

        <td style=”height: 21px”>
          <asp:CheckBoxList ID=”CheckBoxList1” runat=”server”
            AutoPostBack=”True”
OnSelectedIndexChanged=”CheckBoxList1_SelectedIndexChanged”>
          </asp:CheckBoxList>
        </td>
      </tr>

      <tr>
        <td style=”width: 228px; height: 21px”>
          <asp:Label ID=”lblrdlist” runat=”server”>
          </asp:Label>
        </td>

        <td style=”height: 21px”>
          <asp:Label ID=”lblchklist” runat=”server”>
          </asp:Label>
        </td>
      </tr>
    </table>

  </div>
</form>
```

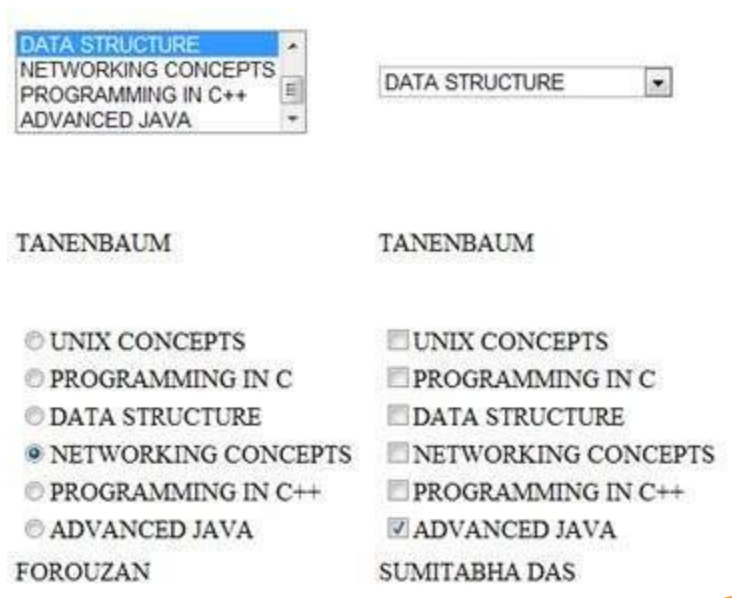
Step (3) : Finally, write the following code behind routines of the application:

```
public partial class _Default : System.Web.UI.Page
{
  protected void Page_Load(object sender, EventArgs e)
  {
    Ilist bklist = createbooklist();

    if (!this.IsPostBack)
    {
      this.ListBox1.DataSource = bklist;
      this.ListBox1.DataTextField = “Book”;
      this.ListBox1.DataValueField = “Author”;

      this.DropDownList1.DataSource = bklist;
      this.DropDownList1.DataTextField = “Book”;
      this.DropDownList1.DataValueField = “Author”;

      this.RadioButtonList1.DataSource = bklist;
      this.RadioButtonList1.DataTextField = “Book”;
      this.RadioButtonList1.DataValueField = “Author”;

      this.CheckBoxList1.DataSource = bklist;
      this.CheckBoxList1.DataTextField = “Book”;
      this.CheckBoxList1.DataValueField = “Author”;

      this.DataBind();
   }
 }

protected Ilist createbooklist()
{
  ArrayList allbooks = new ArrayList();
  booklist bl;

  bl = new booklist(“UNIX CONCEPTS”, “SUMITABHA DAS”);
  allbooks.Add(bl);

  bl = new booklist(“PROGRAMMING IN C”, “RICHI KERNIGHAN”);
  allbooks.Add(bl);

  bl = new booklist(“DATA STRUCTURE”, “TANENBAUM”);
  allbooks.Add(bl);

  bl = new booklist(“NETWORKING CONCEPTS”, “FOROUZAN”);
  allbooks.Add(bl);

  bl = new booklist(“PROGRAMMING IN C++”, “B. STROUSTROUP”);
  allbooks.Add(bl);

  bl = new booklist(“ADVANCED JAVA”, “SUMITABHA DAS”);
  allbooks.Add(bl);
```

```
        return allbooks;
    }

    protected void ListBox1_SelectedIndexChanged(object sender, EventArgs e)
    {
        this.lbllistbox.Text = this.ListBox1.SelectedValue;
    }

    protected void DropDownList1_SelectedIndexChanged(object sender, EventArgs e)
    {
        this.lbldrpdown.Text = this.DropDownList1.SelectedValue;
    }

    protected void RadioButtonList1_SelectedIndexChanged(object sender, EventArgs e)
    {
        this.lblrdlist.Text = this.RadioButtonList1.SelectedValue;
    }

    protected void CheckBoxList1_SelectedIndexChanged(object sender, EventArgs e)
    {
        this.lblchklist.Text = this.CheckBoxList1.SelectedValue;
    }
}
```

Observe the following:

- The booklist class has two properties: bookname and authorname.
- The createbooklist method is a user defined method that creates an array of booklist objects named allbooks.
- The Page_Load event handler ensures that a list of books is created. The list is of Ilist type, which implements the Ienumerable interface and capable of being bound to the list controls. The page load event handler binds the Ilist object 'bklist' with the list controls. The bookname property is to be displayed and the authorname property is considered as the value.
- When the page is run, if the user selects a book, its name is selected and displayed by the list controls whereas the corresponding labels display the author name, which is the corresponding value for the selected index of the list control.


DATA STRUCTURE
NETWORKING CONCEPTS
PROGRAMMING IN C++
ADVANCED JAVA

DATA STRUCTURE

TANENBAUM TANENBAUM

○ UNIX CONCEPTS
○ PROGRAMMING IN C
○ DATA STRUCTURE
● NETWORKING CONCEPTS
○ PROGRAMMING IN C++
○ ADVANCED JAVA

FOROUZAN

☐ UNIX CONCEPTS
☐ PROGRAMMING IN C
☐ DATA STRUCTURE
☐ NETWORKING CONCEPTS
☐ PROGRAMMING IN C++
☑ ADVANCED JAVA

SUMITABHA DAS

FAQ: EXPLAIN SINGLE VALUE DATA BINDING

SINGLE VALUE DATA BINDING

Simple control **data binding** refers to the process of **binding** a **single value** to a property of a control-for instance, **binding** form controls like textboxes, checkboxes, radio buttons, or selected **values** of list controls to **individual data** or object **values**.

Asp.net single value data binding example – using variable

asp.net single value data binding – using variable

DataBindingExample.aspx

```
<%@ Page Language=”C#” AutoEventWireup=”true”
```

```
CodeFile=”DataBindingExample.aspx.cs” Inherits=”DataBindingExample” %>
```

```
<!DOCTYPE html>
```

```
<html xmlns=”http://www.w3.org/1999/xhtml”>
```

```
<head runat=”server”>
    <title>asp.net single value data binding example: using variable</title>
</head>
<body>
    <form id=”form1” runat=”server”>
    <div>
        <h2 style=”color:Teal”>Example: Single Value Data Binding [Using Variable]</h2>
        <asp:Label ID=”Label1” runat=”server” Font-Size=”Large” Font-Italic=”true”
ForeColor=”Crimson”>
            User ID: <%# UserID %><br />
            UserName: <%# UserName %><br />
            City: <%# City %>
        </asp:Label>
    </div>
    </form>
</body>
</html>
```

DataBindingExample.aspx.cs

```
using System;
using System.Collections;
using System.Configuration;
using System.Data;
using System.Linq;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Xml.Linq;

public partial class DataBindingExample : System.Web.UI.Page
{
    public int UserID;
    public string UserName;
    public string City;

    protected void Page_Load(object sender, EventArgs e)
    {
        UserID = 1;
        UserName = "Jones";
    City = "Rome";

  }
}
```

DESCRIBE REPEATED-VALUE DATABINDINGIN DETAIL

Repeated-Value Data Binding

To use **repeated-value binding**, you link one of these controls to a **data** source (such as a field in a **data** table). When you call **DataBind()**, the control automatically creates a full list using all the corresponding **values**.

Although using simple data binding is optional, repeated-value binding is so useful that almost every ASP.NET application will want to use it somewhere.

Repeated-value data binding works with the ASP.NET list controls (and the rich data controls described in the next chapter). To use repeated-value binding, you link one of these controls to a data source (such as a field in a data table). When you call DataBind(), the control automatically creates a full list using all the corresponding values. This saves you from writing code that loops through the array or data table and manually adds elements to a control.

Repeated-value binding can also simplify your life by supporting advanced formatting and template options.

***** END OF UNIT – V *****

******* ALLTHE BEST *******