

**NEW SYLLABUS
CBCS PATTERN**

**T.Y. B.B.A. (C.A.)
SEMESTER - VI**



Dot Net Framework

Dr. Anjali P. Kalkar

Prof. Nutan P. Joshi

Prof. Piyush Dixit

Syllabus ...

1. Introduction to DOT NET FRAMEWORK	[05 Hrs]
1.1 What is Framework?	
1.2 Architecture of Dot Net Framework	
1.2.1 Common Language Runtime	
1.2.2 Common Type System (CTS)	
1.2.3 Common Language Specification (CLS)	
1.2.3 JIT Compilers	
1.2.3 Base Class Library	
1.3 IDE (Integrated Development Environment)	
1.4 Event Driven Programming	
2. Introduction to VB.Net	[11 Hrs]
2.1 Basics of VB.Net	
2.1.1 Operators	
2.1.2 Data Types	
2.1.3 Control Structures	
2.2 Build Windows Applications	
2.2.1 Controls: Form, TextBox, Button, Label, CheckBox, ListBox, ComboBox, RadioButton, DateTimePicker, MonthCalender, Timer, Progressbar, Scrollbar, PictureBox, ImageBox, ImageList, TreeView, ListView, Toolbar, StatusBar, Datagridview	
2.2.2 Menus and PopUp Menu	
2.2.3 Predefined Dialog Controls: Color, Save, File, Open, Font	
2.2.4 DialogBox - InputBox(), MessageBox, MsgBox()	
3. Introduction to C#	[12 Hrs]
3.1 Language Fundamentals	
3.1.1 Data type and Control Constructs	
3.1.2 Value and Reference Types, Boxing	
3.1.3 Arrays	
3.1.4 String Class and its Various Operations	
3.1.5 Functions	
3.2 Object Oriented Concepts	
3.2.1 Defining Classes and Objects	
3.2.2 Access Modifiers	
3.2.3 Constructors	
3.2.4 Inheritance	
3.2.5 Interface	
3.2.6 Abstract Class	
3.2.7 Method Overloading and Overriding	
3.2.8 Delegates	

4. Introduction to ASP.NET

[10 Hrs]

- 4.1 What is ASP.NET?
- 4.2 ASP.NET Page Life Cycle
- 4.3 Architecture of ASP.NET
- 4.4 Forms, WebPages, HTML Forms
- 4.5 Request and Response in Non-ASP.NET Pages
- 4.6 Using ASP.NET Server Controls
- 4.7 Overview of Control Structures
- 4.8 Functions
- 4.9 HTML Events
 - 4.9.1 ASP.NET Web Control Events
 - 4.9.2 Event Driven Programming and Postback
- 4.10 Introduction to Web forms
 - 4.10.1 Web Controls
 - 4.10.2 Server Controls
 - 4.10.3 Client Controls
 - 4.10.4 Navigation Controls
 - 4.10.5 Validations
 - 4.10.6 Master Page
 - 4.10.7 State Management Techniques

5. Architecture of Ado.Net

[10 Hrs]

- 5.1 Basics of Ado.net
 - 5.1.1 Connection Object
 - 5.1.2 Command Object
 - 5.1.3 Dataset
 - 5.1.4 Data Table
 - 5.1.5 Data Reader Object
 - 5.1.6 Data Adapter Object
- 5.2 Datagridview and Data Binding: Insert, Update, Delete Records
- 5.3 Navigation Using Data Source



Contents ...

1. Introduction to DOT NET FRAMEWORK	1.1 - 1.17
2. Introduction to VB.Net	2.1 - 2.50
3. Introduction to C#	3.1 - 3.62
4. Introduction to ASP.NET	4.1 - 4.122
5. Architecture of Ado.Net	5.1 - 5.45
Solved Question Papers	P.1 - P.6

■ ■ ■

Introduction to DOT NET FRAMEWORK

Learning Objectives ...

Students will be able to:

- Get a basic knowledge about framework and architecture of .Net.
- Build and run applications on Windows.
- Develop Form-based applications, Web-based applications, and Web services.

1.1 WHAT IS FRAMEWORK?

- Framework is a collection of reusable classes which represent software code and design that can be recycled for various application domains.
- Framework is an essential component of windows operating system, which helps in creating applications by integrating different programming languages such as c#, VB, J# & visual C++.
- A framework can be a collection of libraries and technologies consist of many reusable classes and line of code.
- .Net framework is a way that helps one to build software economically.
- .Net framework provides a bridge to make sure the interoperability between applications which are created using various languages such as C#, Visual Basic.

Benefits of .Net Framework:

[S-23, W-22]

1. **Consistent programming model:** User can able to create programs for performing different tasks such as database applications as well as reading and writing in files.
2. **Cross-platform support:** .Net applications enables interoperability between multiple windows operating system.
3. **Language Interoperability:** Code written in different languages interact with each other, it leads to improves efficiency and reusability of code.

Objectives of .Net framework:**[W-22]**

- .Net introduced a unified programming Environment:** All .Net enabled languages compiles to "Microsoft intermediate language" before being assembled into platform-specific machine code.
- .Net committed developers to object oriented technologies:** .Net implements object oriented programming paradigm; everything in .Net is contained in an object.
- .Net simplified windows programming:** With .Net, APIs are replaced with hierarchy of object providing access to many commonly needed windows features.
- .Net provides Security:** By using .Net one can limit malicious programs from doing their damage.
- .Net enhanced web-based development:** Until .Net many web-based development was done using scripting languages, but .Net enables the power of compiled, desktop development to the internet.

1.2 ARCHITECTURE OF DOT NET FRAMEWORK**[S-22, W-22]****1.2.1 Common Language Runtime (CLR)****1.2.2 Common Type System (CTS)****1.2.3 Common Language Specification (CLS)**

- .Net is tiered, modular and hierachal.
 - .Net is a software platform, not an operating system.
- This platform is used to compile and execute programs written by using .Net compatible languages.

.Net platform composed of different components like .Net compatible languages, Common Languages Specification (CLS), Common Type System (CTS), .Net Framework Class Library (FCL), Common Language Runtime (CLR) and Operating System (OS).

VB.NET	C#.NET	Jscript.Net	.Net Languages			
Common Language Specification(CLS)						
Common Type System(CTS)						
.NET Framework Class Library(FCL)						
ASP .NET Web forms, XML	Windows Form		Console			
ADO.NET	.NET Remoting					
Common Language Runtime(CLR)						
Operating System						

.NET Framework Architecture

1. .Net Compatible Languages:

- .Net has come up with various programming languages which help to integrate the application designed in different technologies.
- It intended to develop the application in different languages and integrates them together.
- Different Compatible Languages such as VB.NET, C#.NET, Jscript.NET, Visual C++, .NET are available.

2. Common Language Specification:

- Microsoft .Net design set of rules that have to be followed by every language.
- For programming languages to communicate effectively it is necessary that every language should adhere to the Common Language specification (CLS).
- CLS defines a minimum set of features that a language must implement before it is considered to be .Net compliant.
- Whenever the object of the class is to be declared rule simply states that new keywords must be used.

For example: When there is an establishment of SQL server Connection, user need to use SQL connection class and should name space as System.Data.SqlClient

3. Common Type System:

[S-22, W-22]

- It specifies how types are declared, used and managed in the runtime.
- It is runtimes support for cross-language integration.
- CTS is also called as super set of CLS.
- Multiple Programming Languages are supported on a single CLR and have the ability to reuse the FCL, it is necessary that type of each programming language must be compatible.
- This Binary compatibility between languages type is called as Common Type System (CTS).
- .Net Framework Common Type System (CTS) defines the entire core data types and data mechanism used in .Net programs.
- It consists of all numeric, string and Boolean value types, it also defines the object, core data storage unit in .Net

CTS divides data object into two general categories of type:

- (a) Value Types
- (b) Reference Types

(a) Value Types: Value type contains their data and instances of value type are allocated on the stack, they cannot be null and must always contain some data.

There are three general value types:

- (i) Build in types
- (ii) User defined type
- (iii) Enumeration Type

i) **Build in Types:** Primitive data types defined in .Net such as Byte, Int16, Int32 and Int64, Single, Double, Decimal, Char, Boolean, Date time etc. Value type can be defined in VB.NET as

```
Dim a As Integer
Dim b As Boolean=false
Value type can be defined in C#.Net
Int a;
Bool b=false;
```

ii) **User Defined Type:** User defines types are also called as structure, structures are composed of other data types.

In VB.Net define structure as:

```
Structure Student
    Dim a as integer
    Dim b as integer
End Structure
-Create object-instance of structure as bellowed-
Dim obj1 As New Student
```

In C#.Net defines structure as-

```
Struct Student
{
    Int no;
    Int n1, n2;
}
```

Object-instance creation is as below-

```
Student s1=new Student();
```

i) **Enumeration Type:** It provides a list of choices for developers using class.

Example:

In VB.Net

```
Enum color as integer
```

Red

Blue

Green

Black

Yellow

White

```
End Enum
```

In C#.Net

```
Enum color: int {Red, Blue, Green, Black, Yellow, White};
```

Reference Type: It contains reference where data is stored are allocated on the heap and reference of data is stored on a stack.

Reference types i.e. object, interfaces and pointers are so called because they contains references to heap based objects and can be null.

These types are passed by reference, it means when we pass an object into function address of or pointer to the object is passed not a copy of the object.

4. .NET Framework Class Library (FCL):

- .NET intended to design RAD (Rapid Application Development), it concentrate more on to the business requirement rather than common functionality.
- Common functionalities like sort data, searching data, string manipulation, data functions, database connectivity etc. are pre-defined in .NET framework.
- In this FCL Layer, all .NET class libraries are maintained and can be used by user to create the application.
- Examples of class libraries such as System. Object, System.IO, System.XML, ADO.NET classes etc.

5. Common Language Runtime (CLR):**[S-22]**

- CLR is the execution engine of .Net and also called as the heart of .NET architecture.
- It takes the responsibility for execution of .NET Application, Memory Management, Garbage Collection, Thread Management.
- It provides execution environment to .Net Program.
- Out of the compiled .Net program is not an executable file but it is a Microsoft Intermediate Language (MSIL) which is low level set of instruction understandable by CLR.
- CLR has responsibility to translate this Intermediate code (MSIL) into executable code.
- This translation is done by JIT (Just In-Time Compilers).

CLR provides following Execution Support Services like:

(a) Garbage Collection: Unlike C++ CLR supports automatic lifetime management for all .NET objects.

CLR Garbage Collector can detect when objects are no longer being referenced and perform garbage collection to reclaim unused memory.

(b) Exception Handling: In .Net, CLR supports a standard exception handling mechanism that work across all languages, allocating every program to use a common error handling mechanism.

(c) Security Support: CLR performs various security checks at runtime to ensure the code is safe to execute.

CLR also supports code access security, declarative and imperative security checks.

(d) Debugging support: CLR Provides Support for debugging through an API that the vendors can use to develop a debugger.

This API contains support for controlling program execution, setting breakpoints, exceptions, control flow etc.

1.2.4 JIT Compilers

- Compiled program in .NET is a code called as Microsoft Intermediate Language (MSIL, which is a Low Level set of instruction understood by CLR).
- This MSIL is not an executable file hence it is necessary to turn MSIL into an executable code called as Native Code or machine code using JIT (Just In-Time) Compilers.
- JIT Compilers converts MSIL into native code that it can execute on the target Operating System.
- This native code is understood by operating system and further executes by it.
- The full MSIL code was not translated into native code, JIT compiler converts MSIL code into native code on demand basis as each part of the program is needed.
- JIT Compiler dynamically compiles code that is optimized for the target machine.

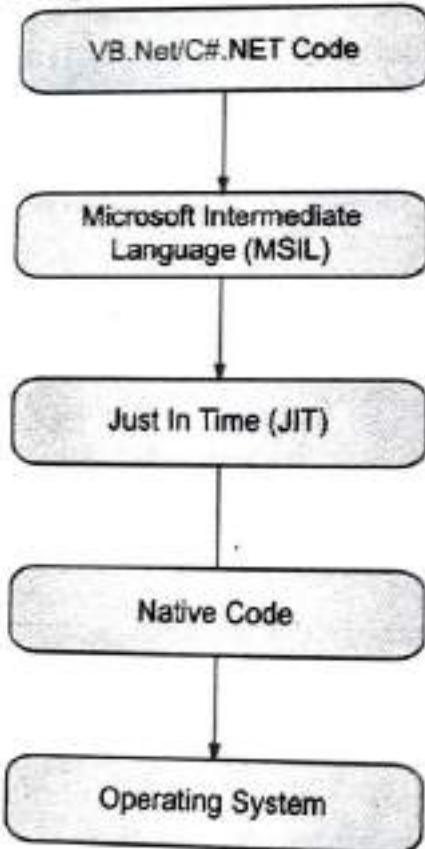


Fig. 1.1

Fig. 1.1 above shows the execution of code by using JIT Compiler. User can implement code in any .NET specific language. The code is compiled and generates MSIL code called an Assembly. JIT converts MSIL into native code also called as machine code which is a machine understandable code, it is then executed.

2.5 Base Class Library (BCL)-Core Set

BCL is smaller library which contains the most essential features that a program just could not do without.

- BCL includes only those classes that are an absolute must for supporting applications on the framework.
- BCL provides the most foundational types and utility functionality which is base of all other.
- BCL aims to provide general implementation without bias of any workload

1.3 IDE (INTEGRATED DEVELOPMENT ENVIRONMENT)

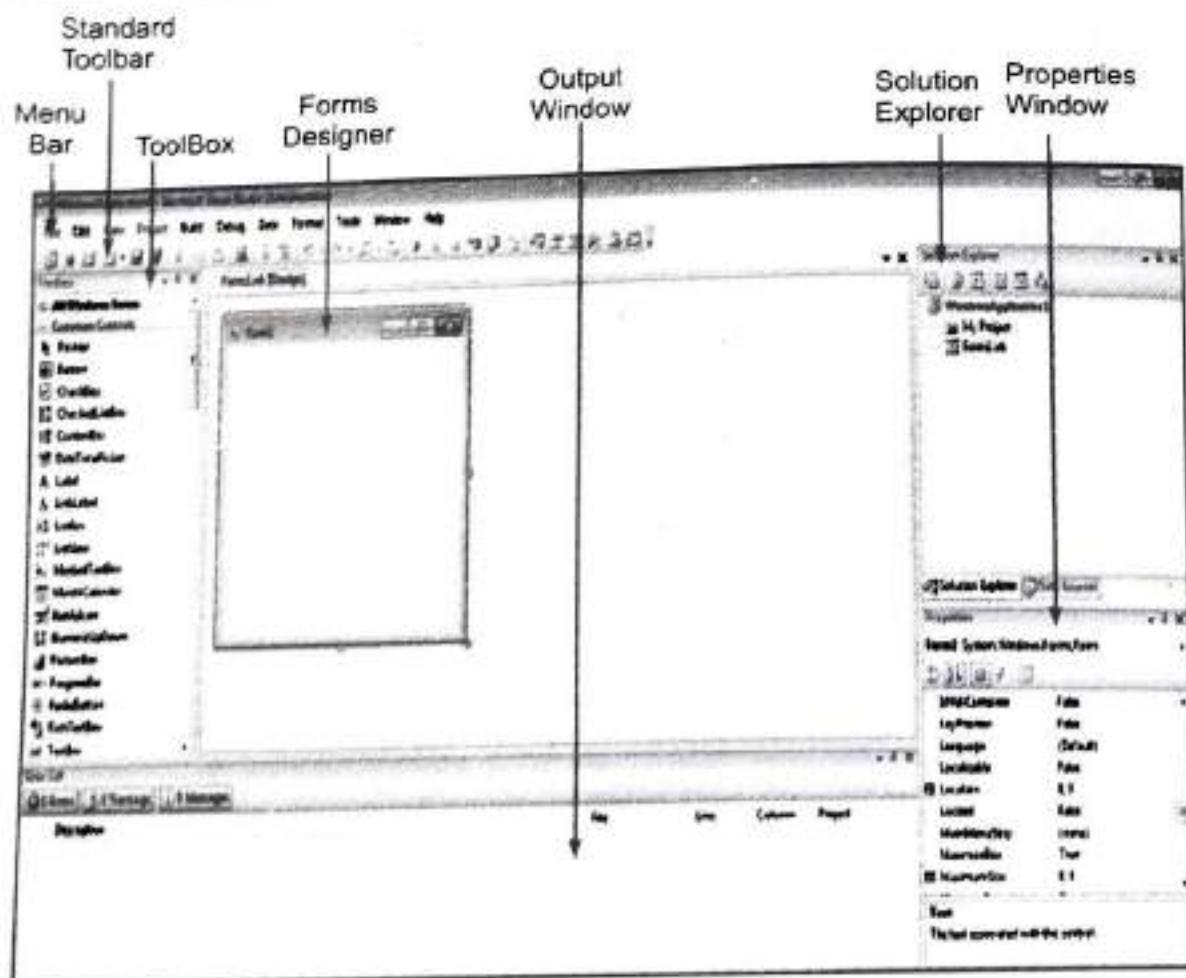
- IDEs, are software platforms that provide programmers and developers with a comprehensive set of tools for software development in single product specifically in the .NET framework.
- IDEs are built to work with specific applications platforms.
- It removes barriers of SDLC.
- .Net IDEs are used to program code for a specific platform and have integrated features specifically designed for use within these platform including capabilities to compile, debug or intelligently complete code automatically.
- An Integrated Development Environment (IDE) is software that facilitates application development.
- IDE consist of windows for visual design of forms, code-editing windows, menus and toolbars providing access to commands and features.
- It also consists of toolboxes containing control for use on the forms and windows providing properties and information about forms.

In the context of .NET based applications, Visual Studio is the most Common IDE having following key features:

- Single IDE for all .Net applications hence no switching is required.
- Single .NET solution for an application which has been built on code written in multiple languages.
- Integrated debugger that works at source and machine level.
- Provides customizable environment.

Visual Studio IDE:

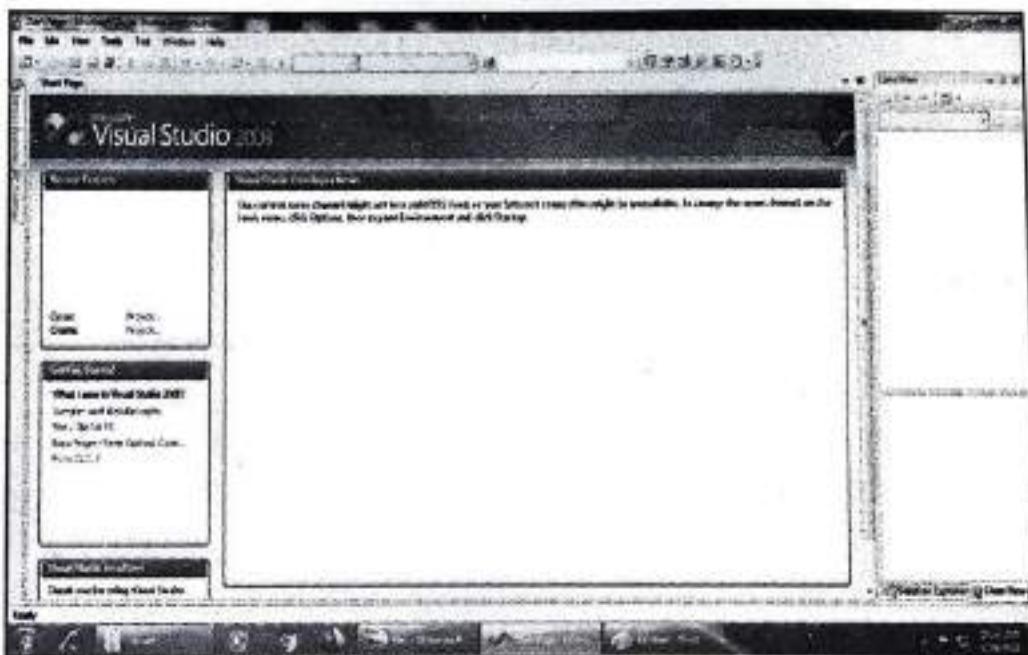
- Visual Studio is a powerful and customizable programming environment that contains all the tools you need to build programs quickly and efficiently. It offers a set of tools that help you write and modify the code for your programs, and also detect and correct errors in your programs.
- Before you start learning more about VB.NET programming, it is important to understand the development environment and identify some of the frequently using programming tools in Visual Studio IDE.



- Visual Basic.NET IDE is built out of a collection of different windows. Some windows are used for writing code, some for designing interfaces, and others for getting a general overview of files or classes in your application.
- Visual Studio organizes your work in projects and solutions. A solution can contain more than one project, such as a DLL and an executable that references that DLL. From the following chapters you will learn how to use these Visual Studio features for your programming needs.

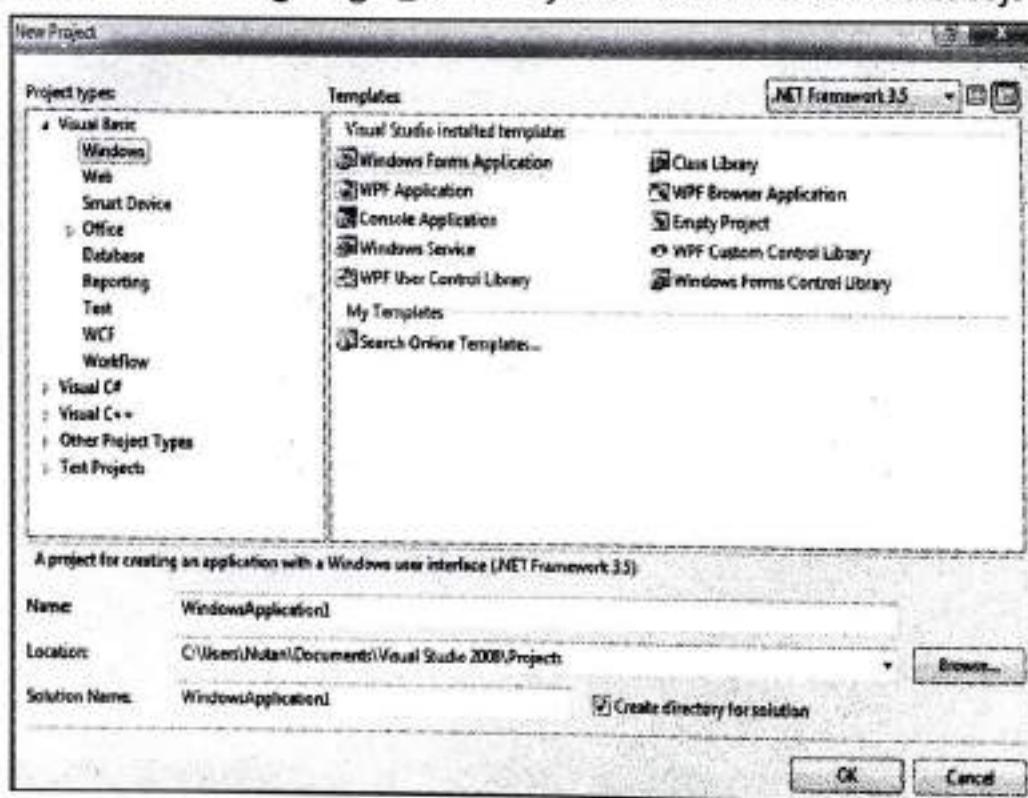
1. Starting Page Window:

- When we open VB.NET from start -> Programs -> Microsoft Visual Studio.NET -> the window that is displayed first is the start page as shown below:
- The start page allows us to select from the most recent projects with which we worked:



2. New Project Dialogue:

- The New Project Dialogue box like the one which is used to create a new project specifying it allow us to name the project and also specify its location on the disk where it is saved.
- The default location on the hard disk where all the projects are saved is, C:\DocumentsandSettings\login_name\MyDocuments\VisualStudioProjects.

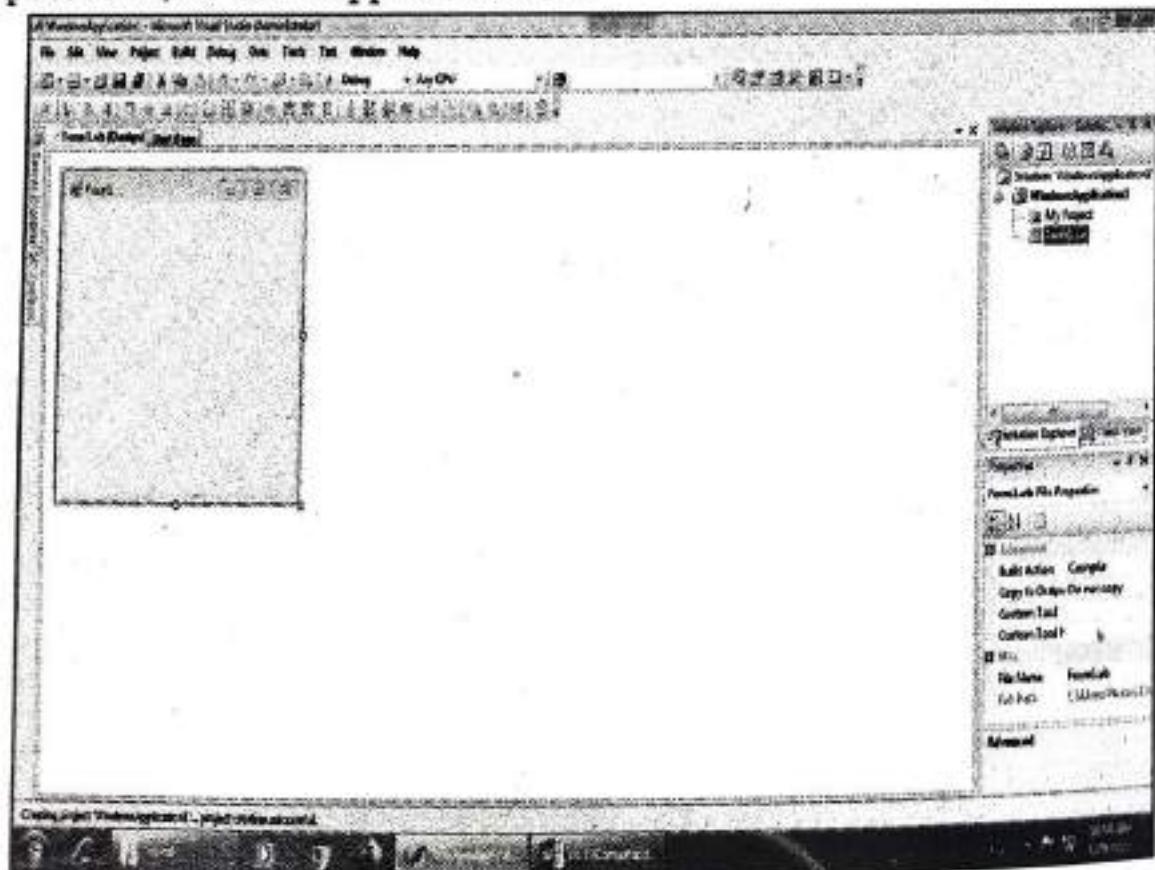


Following are different templates under Project Types and their use.

Windows Application:

This template allows creating standard windows based applications.

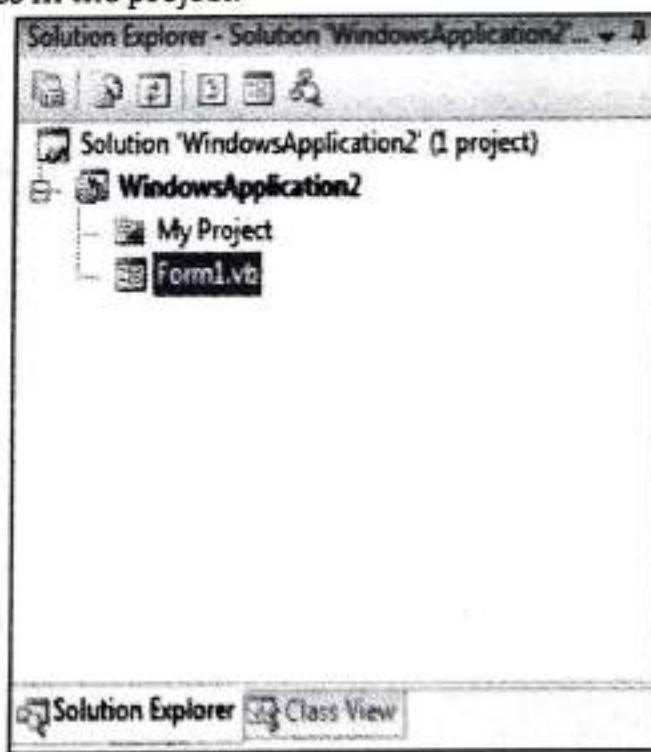
- **Class Library:** Class libraries are those that provide functionality similar to ActiveX and DLL by creating classes that access other applications.
- **Windows Control Library:** This allows creating our own windows controls. Also called as **User Controls**, where you group some controls, add it to the toolbox and make it available to other projects.
- **ASP.NET Web Application:** This allows to create web-based applications using IIS. We can create web pages, rich web applications and web services.
- **ASP.NET Web Service:** Allows creating XML Web Services.
- **Web Control Library:** Allows creating User-defined controls for the Web. Similar, to user defined windows controls but these are used for Web.
- **Console Application:** A new kind of application in Visual Studio.NET. They are command line based applications.
- **Windows Service:** These run continuously regardless of the user interaction. They are designed for special purpose and once written, will keep running and come to an end only when the system is shut down.
- **Other:** This template is to develop other kinds of applications like enterprise applications, database applications etc.





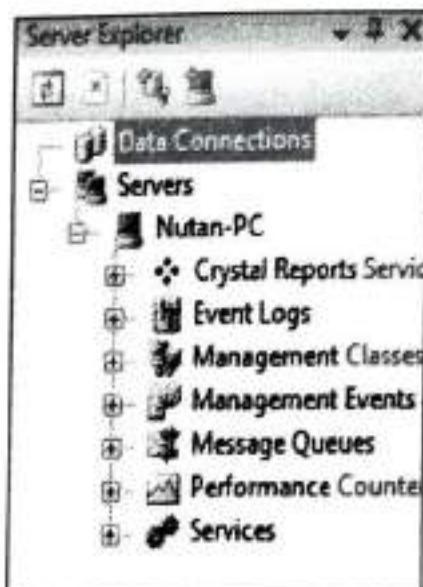
3. Solution Explorer Window:

- The Solution Explorer window gives an overview of the solution we are working with and lists all the files in the project.



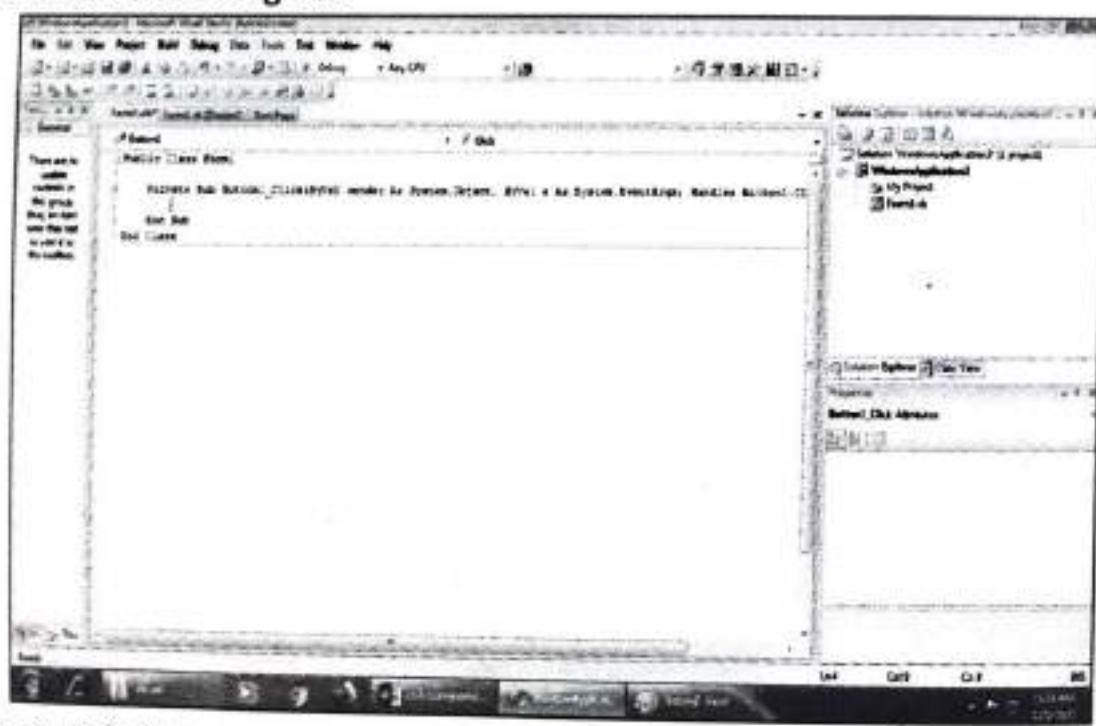
4. Server Explorer Window:

- The Server Explorer window is a great tool that provides "drag and drop" features and helps us work with databases in an easy graphical environment.



5. Code Designer Window:

- Code Designer Window allows us to edit and write code. This is the window that opens when we double-click on a form or any control of form.
- The left box allows us to select the objects code we are working with and the right box allows us to select the part of code that we want to work. Also notice the "+" and "-" boxes in the code designer.



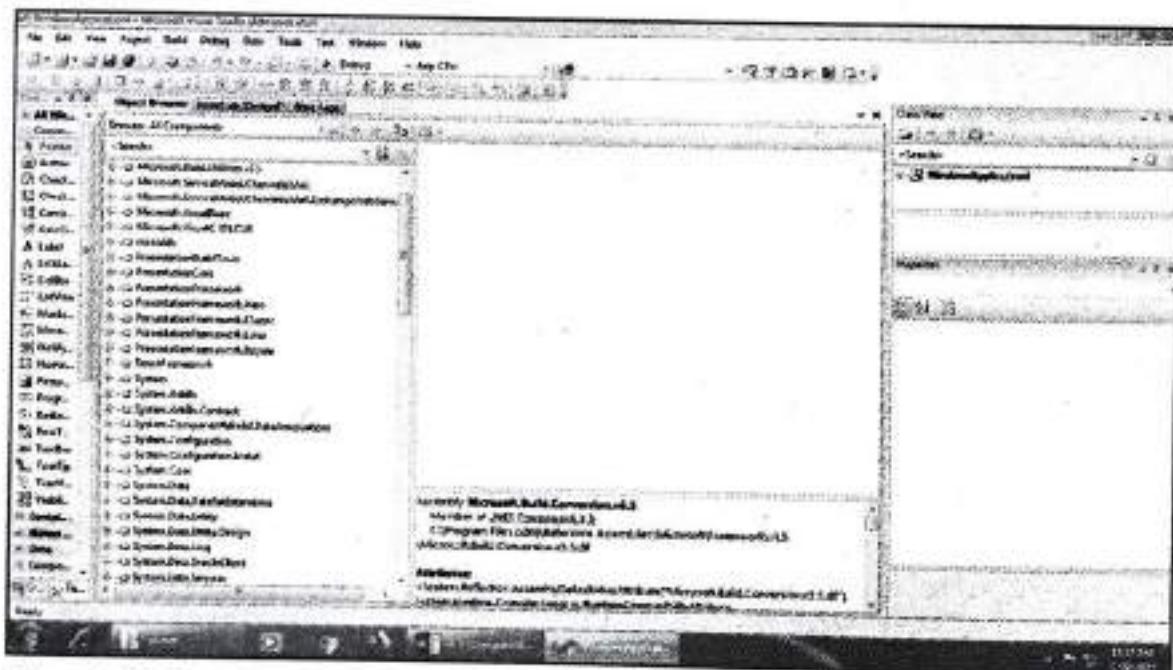
6. Property Window:

- The properties window allows us to set properties for various objects at design time.
- If user want to change the font, font size, backcolor, name, text that appears on a button, textbox etc you can do that in this window.



7. Object Explorer Window:

- The Object Explorer window allows us to view all the members of an object at once.
- It lists all the objects in our code and gives us access to them.
- You can view the object explorer window by selecting View -> object browser from the main menu.



8. Toolbox Window:

- The toolbox window is the window that gives us access to all controls, components etc.

- Toolbox uses tabs to divide its contents into categories. These tabs marked Data, Components, Windows Forms and General.
- The Data tab displays tools for creating datasets and making data connections, the windows forms tab displays tools for adding controls to forms, the general tab is left empty by default.
- The clipboard ring tab displays recent items stored in the clipboard and allows us to select from them.



1.4 EVENT DRIVEN PROGRAMMING

[S-23]

- The concept of an Event Driven Programming is simply that enables you to write code that executes in response to events generated by specific actions.
- Events are the actions that objects can generate.
- Objects (i.e. users) generate or initiate some events in the program.
- Event driven programming fosters the dynamic interaction between users and computers.
- User interact with the controls on form, as per the actions taken by user some of code is executed in order to response those actions.
- User actions cause's events such as Click Event, Load Event, Double-Click Event, Key Press Even, Mouse Move Event and many more.
- For Example, Demonstration to handle click event on button in VB.Net

- Private Sub Button1_Click (ByVal Sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click

```

    Dim a,b As Integer
    a=TextBox1.Text
    b=TextBox2.Text
    If (a>b) Then
        TextBox3.Text=a.ToString
    Else
        TextBox3.Text=b.ToString
    End If
End Sub
Event driven programming

```

- Events are basically a user action like key press, clicks, mouse movements, etc., or some occurrence like system generated notifications. Applications need to respond to events when they occur.
- Clicking on a button, or entering some text in a text box, or clicking on a menu item, all are examples of events. An event is an action that calls a function or may cause another event. Event handlers are functions that tell how to respond to an event.
- VB.Net is an event-driven language. There are mainly two types of events:
 - Mouse events
 - Keyboard events

Handling Mouse Events:

- Mouse events occur with mouse movements in forms and controls. Following are the various mouse events related with a Control class:
 - **MouseDown:** It occurs when a mouse button is pressed.
 - **MouseEnter:** It occurs when the mouse pointer enters the control.
 - **MouseHover:** It occurs when the mouse pointer hovers over the control.
 - **MouseLeave:** It occurs when the mouse pointer leaves the control.
 - **MouseMove:** It occurs when the mouse pointer moves over the control.
 - **MouseUp:** It occurs when the mouse pointer is over the control and the mouse button is released.
 - **MouseWheel:** It occurs when the mouse wheel moves and the control has focus. The event handlers of the mouse events get an argument of type **MouseEventArgs**. The **MouseEventArgs** object is used for handling mouse events. It has the following properties:
 - **Buttons:** Indicates the mouse button pressed.
 - **Clicks:** Indicates the number of clicks.
 - **Delta:** Indicates the number of detents the mouse wheel rotated.
 - **X:** Indicates the x-coordinate of mouse click.
 - **Y:** Indicates the y-coordinate of mouse click.

Example:

```

Public Class Form1
Private Sub Form1_Load(sender As Object, e As EventArgs)
    //Set the caption bar text of the form.
    Me.Text = "tutorialspsont.com"
End Sub
Handles MyBase.Load

```

```
Private Sub txtID_MouseEnter(sender As Object, e As EventArgs)_
    Handles txtID.MouseEnter
    //code for handling mouse enter on ID textbox
    txtID.BackColor = Color.CornflowerBlue
    txtID.ForeColor = Color.White
End Sub

Private Sub txtID_MouseLeave(sender As Object, e As EventArgs) _
    Handles txtID.MouseLeave
    //code for handling mouse leave on ID textbox
    txtID.BackColor = Color.White
    txtID.ForeColor = Color.Blue
End Sub

Private Sub txtName_MouseEnter(sender As Object, e As EventArgs) ...
    Handles txtName.MouseEnter
    //code for handling mouse enter on Name textbox
    txtName.BackColor = Color.CornflowerBlue
    txtName.ForeColor = Color.White
End Sub

Private Sub txtName_MouseLeave(sender As Object, e As EventArgs) ...
    Handles txtName.MouseLeave
    //code for handling mouse leave on Name textbox
    txtName.BackColor = Color.White
    txtName.ForeColor = Color.Blue
End Sub

Private Sub txtAddress_MouseEnter(sender As Object, e As EventArgs) ...
    Handles txtAddress.MouseEnter
    //code for handling mouse enter on Address textbox
    txtAddress.BackColor = Color.CornflowerBlue
    txtAddress.ForeColor = Color.White
End Sub

Private Sub txtAddress_MouseLeave(sender As Object, e As EventArgs) ...
    Handles txtAddress.MouseLeave
    //code for handling mouse leave on Address textbox
    txtAddress.BackColor = Color.White
    txtAddress.ForeColor = Color.Blue
End Sub

Private Sub Button1_Click(sender As Object, e As EventArgs) ...
    Handles Button1.Click
    MsgBox("Thank you " & txtName.Text & ", for your kind cooperation")
End Sub
End Class
```

Summary

- Framework is a collection of reusable classes which represent software code and design that can be recycled for various application domains.
- A framework can be a collection of libraries and technologies consist of many reusable classes and line of code.
- .Net platform composed of different components like .Net compatible languages, Common Languages Specification (CLS), Common Type System (CTS), .Net Framework Class Library (FCL), Common Language Runtime (CLR) and Operating System (OS).
- .NET Framework Architecture
 - .Net Compatible Languages
 - Common Language Specification
 - Common Type System
 - .NET Framework Class Library (FCL)
 - Common Language Runtime (CLR)
- Compiled program in .NET is a code called as Microsoft Intermediate Language (MSIL, which is a Low Level set of instruction understood by CLR).
- This MSIL is not an executable file hence it is necessary to turned MSIL into an executable code called as Native Code or machine code using JIT (Just In-Time) Compilers.
- IDEs, are software platforms that provide programmers and developers with a comprehensive set of tools for software development in single product specifically in the .NET framework.
- The concept of an Event Driven Programming is simply that enables you to write code that executes in response to events generated by specific actions.

Practice Questions

1. What are the different .NET compatible languages?
2. What is .Net Framework? Explain the components of .Net Framework.
3. What is the use of CLR?
4. What does FCL provide to the user?
5. What is CTS?
6. Differentiate between Value Type and Reference Type.
7. What is MSIL?
8. State the Role of JIT.
9. Explain the detailed architecture of .NET Framework.
10. Enlist and explain various objectives of .Net Framework.
11. State various advantages of .Net.



2...

Introduction to VB.Net

Learning Objectives ...

Students will be able to:

- Design, formulate, and construct applications with VB.NET.
- Integrate variables and constants into calculations applying VB.NET.
- Determine logical alternatives with VB.NET decision structures.
- Implement lists and loops with VB.NET controls and iteration.
- Separate operations into appropriate VB.NET procedures and functions.
- Assemble multiple forms, modules, and menus into working VB.NET solutions.
- Create VB.NET programs using multiple array techniques.

1 INTRODUCTION

Visual Basic .NET (VB.NET) is an object-oriented computer language that can be viewed as an evolution of Microsoft's Visual Basic (VB) implemented on the Microsoft .NET Framework.

VB.NET programming is very easy to understand and can be used to rapidly design and develop the application.

VB.NET application goes through the following stages as shown below:

1. Source Code (VB.NET application)
2. Net compiler
3. MSIL (Microsoft Intermediate Language)
4. CLR (Common Language Runtime)
5. Output

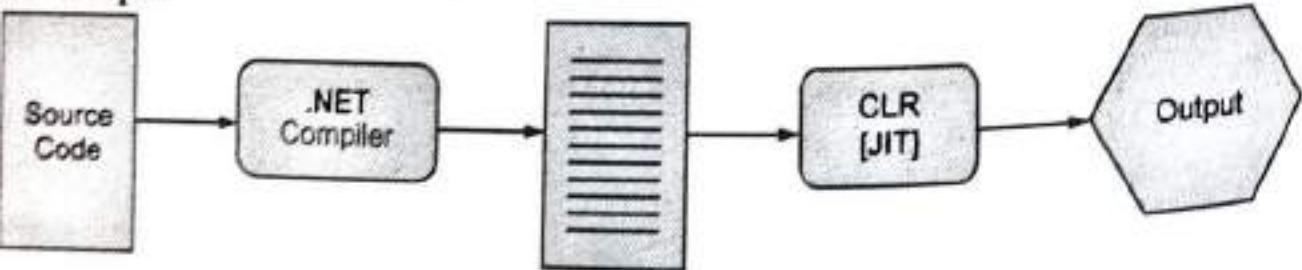


Fig. 2.1
(2.1)

Features of VB.NET:

- Following are the key features of VB.NET
 1. Powerful Windows-based applications
 2. Building web-based Applications
 3. Simplified Deployment
 4. Powerful, flexible, simplified Data Access
 5. Improved coding
 6. Direct Access to the platform
 7. Full object-oriented constructs
 8. XML Web Services
 9. Mobile Applications

2.2 BASICS OF VB.Net

[S-22]

2.2.1 Operators

- VB.Net provides many built-in operators that allow us to manipulate data.

- An operator performs a function on one or more operands.

- Available operators are as below:

1. Arithmetic Operators
2. Comparison Operators
3. Logical/Bitwise Operators
4. Concatenation Operators

1. Arithmetic Operators:

- Arithmetic Operators are used to perform operations that involve calculation of Numeric Values

Operator	Use
$^$	Exponentiation
-	Negation
*	Multiplication
/	Division
Mod	Modulus Arithmetic
+	Addition
-	Subtraction

Example 1:

1. Create a new console application project from new project dialog box.
2. Select the module1 and write the following code

```

imports System.Console
Module Module1
    Sub Main ()
        Dim X, Y, Z As Integer
        X=30
        Y=20
        Z=X+Y
        WriteLine ("Addition " & "=" & Z)
        Z=X-Y
    End Sub
End Module

```

```

    WriteLine ("Subtraction" & "=" & Z)
    Read()
End Sub
End Module

```

Output:

Addition=50
Subtraction=10

[S-22]

Comparison Operators:

Comparison Operator compares Operands and returns a logical value based on whether the comparison is true or not.

Operator	Use
=	Equality
<>	Inequality
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
=	Equality

Example 2:

[S-23]

```

imports System.Console
Module Module1
    Dim X, Y As Integer
    WriteLine ("enter values for X and Y")
    X=Val (ReadLine ())
    Y=Val (ReadLine ())
    If X=Y Then
        WriteLine ("X is equal to Y")
    Else If X<Y Then
        WriteLine ("X is less than Y")
    Else If X>Y Then
        WriteLine ("X is greater than Y")
    End If
    Read ()
End Sub
End Module

```

Output:

Enter value for X and Y
10 20
X is less than Y

3. Logical/Bitwise Operators:

- Logical Operators compare Boolean expression and return a Boolean result.
- These Operators return a true or false result over a conditional expression.

Operator	Use
NOT	Negation
AND	Conjunction
OR	Disjunction
AndAlso	Conjunction
OrElse	Disjunction

Example 3:

```

import System.Console
Module Module1
Sub Main()
    Dim a As Integer=10
    Dim b As Integer=8
    Dim c As Integer=6
    Dim first,second,third As Boolean
    First=a>b AndAlso b>c
    Second=b>a AndAlso b>c
    WriteLine(first)
    WriteLine(second)
    WriteLine(third)
    read()
End Sub
End Module

```

Output:

True
False
False

4. Concatenation Operators:

[S-23]

- Concatenation operators join multiple strings into a single string.
- Concatenation operators are as below:

Operator	Use
+	String Concatenation
&	String Concatenation

Note: The (&) operator always makes sure that both operands are string, while the + operator finds the overload that matches the operands.

Example 4:

```

import System.Console
Module Module1
Sub Main()
Dim str1, str2, str3, str4 As String
Str1="Welcome"
Str2="Students"
Str3=str1+str2
WriteLine(str3)
Str4=str1&str2
WriteLine(str4)
Read()
End Sub
End Module

```

Output:

WelcomeStudents
WelcomeStudents

2.2 Data Types and Variables**[S-23, 22, W-22]**

Available Data Types in VB.NET along with their size, type, description is as bellowed:

Datatype	Size	Description	Type
Byte	1	8-bit unsigned integer	System.Byte
Char	2	16-bit Unicode character	System.Char
Integer	4	32-bit signed integer	System.Int32
Double	8	64-bit floating point variable	System.Double
Long	8	64-bit signed integer	System.Int64
Short	2	16-bit signed integer	System.Int16
Single	4	32-bit floating point variable	System.Single
String	Varies	Non-numeric Type	System.String
Decimal	16	128-bit floating point variable	System.Decimal
Date	8	Date Type	System.Date
Boolean	2	Non-Numeric Type	System.Boolean
Object	4	Non-Numeric Type	System.Object

Data type Conversion/Type Casting:

A Process of converting a value of one datatype to another is called as conversion or casting.

Conversions are made in two ways:

1. Implicit Conversion
2. Explicit Conversion

1. Implicit Conversion:

- It converts data from a data type with a narrower range of possible values to a data type with a wider range of possible values.

```
Module Module1
    Sub Main ()
        Dim n1 As Integer=100
        Dim n2 As Integer=75
        Dim total As Long
        Total=n1+n2
        Console.WriteLine("Total is:" &total)
        Console.ReadLine()
    End Sub
End Module
```

2. Explicit Conversion:

- It converts data from a wider range data type to a narrower range data type.
- In this conversion CLR checks for data compatibility at runtime.
- Explicit conversion requires explicitly calling a conversion function.

Operator	Use
CBool	Converts String or numeric expression to Boolean
Cbyte	Converts numeric expression to Byte
CChar	Take a string argument and return first character of the string as a Char data type
CDate	Convert any valid representation of a date or time
Cint	Convert any numeric expression in the range of integer
CShort	Convert any numeric expression to short
CDec	Convert evaluated numeric expression to Decimal
CDbl	Convert any expression that can be evaluated to a number in the range of a Double to Double

```
Module Module1
    Sub Main ()
        Dim num As Integer
        Dim mark As Decimal=34.75
        num=CInt(mark)
        Console.WriteLine("Converted value is:" &num)
        Console.ReadLine()
    End Sub
End Module
```

riables:

It is an entity that holds things having following six properties:

- Name:** It used to identify variable, a name can start with an alphabetic character and can be followed by alphabetic character, numeric, formatting or combined character.
- Address:** Variable Address refer the memory location at which variable value is stored.
- Type:** It is nothing but Data Type which determines possible value type that variable can assume.
- Value:** It is generally R-value of variable which exactly the content of the memory location at the specified address of the variable.
- Scope:** Where exactly that variable is visible to the code in a program.
- Lifetime:** It describes when and for how long particular variable exists.

2.2.3 Statements and Control Structures

[W-22]

To enables decision making ability .NET supports various decision-making statements.

if then statement

Syntax:

```
if condition then
    Statements
End if
```

Example:

```
Private Sub Button1_Click(ByVal Sender As System.Object, ByVal e As
System.EventArgs)_Handles Button1.Click
Dim a As Integer=30
if i=30 then
    MessageBox.Show("Value is match")
End if
End Sub
```

if then else statement

Syntax:

```
if condition then
    statements
else
    statements
end if
```

Example:

```
Private Sub Button1_Click (ByVal Sender As System.Object, ByVal e As
System.EventArgs)_Handles Button1.click
Dim i as integer=30
```

```

if i<50 then
    MessageBox.Show("The value is below limit")
else
    MessageBox.Show("The value is above limit")
End if
End Sub

```

[S-22]

3. else if Statement

Syntax:

```

if condition then
    Statements
else if condition then
    Statements
else
    Statements
End if

```

Example:

```

Private Sub Button1_Click (ByVal Sender As System.Object, ByVal e As
System.EventArgs)_Handles Button1.Click
Dim iAs Integer=30
if i<30 then
    MessageBox.Show("Range of Value is Smaller")
else if i=30 then
    MessageBox.Show("Range of Value is equal")
else
    MessageBox.Show("Range of Value is Larger")
End if
End Sub

```

4. Select Case

- Select case statements execute one of several groups of statements depending on the value of an expression.

Syntax:

```

Select expression
    Case expressionlist
        Statements
    Case else
        Statements
End Select

```

Example:

```

Module Module1
    Sub Main()
        Dim grade As Char
        grade="B"
    End Sub

```

```

Select grade
Case "A"
Console.WriteLine("Excellent")
Case "B", "C"
Console.WriteLine("Well done")
Case "D"
Console.WriteLine("You Passed")
Case "F"
Console.WriteLine("Better try again")
Case Else
Console.WriteLine("Invalid grade")
End Select
Console.ReadLine()
End Sub
End Module

```

Loops:**1. For Loop:**

- For loop enable us to execute a series of expressions multiple numbers of times.
- For loop in VB.NET needs a loop index which counts the number of loop iterations the loop executes.

Syntax:

```

For index=start to end step stepvalue
    Statements
Next

```

Example:

```

Module Module1
Sub Main ()
Dim a As Byte
For a=10 to 20
    Console.WriteLine("Value of a:", a)
Next
Console.ReadLine()
End Sub
End Module

```

2. For each...Next loop:

- With the help of for each Next loop one can use to execute a block of code for each element in the collection instead of executing the block a set number of times.

Syntax:

```

For each element in group
    Statements
Next element

```

Example:

```
Module Module1
Sub Main ()
Dim Arr(3),j As Integer
Arr(0)=0
Arr(1)=1
Arr(2)=2
Arr(3)=3
For each j in Arr
    Console.WriteLine(j)
Next j
    Console.ReadLine()
End Sub
End Module
```

3. While Loop:

- While loop keeps executing until the condition against which it tests remains true.

Syntax:

```
While Condition
Statements
End While
```

Example:

```
Module Module1
Sub Main ()
Dim a As Integer=10
While a<20
    Console.WriteLine("Value of a:", a)
    a=a+1
End While
    Console.ReadLine()
End Sub
End Module
```

4. Do.... While loop:

- The Do...While loop can be used to execute a fixed block of statements indefinite number of times.
- Do loop keeps executing its statements while or until the condition is true.
- Two keywords, while and until can be used with the do loop.
- Do loop also supports an exit Do statement which makes the loop to exit at any moment.

Syntax:

```
Do while/until condition
Statements
Loop
```

Example:

```

Module Module1
Sub Main ()
    Dim a As Integer=10
    Do
        Console.WriteLine("Value of a:", a)
        a=a+1
    Loop While (a<20)
    Console.ReadLine ()
End Sub
End Module

```

2.3 BUILD WINDOWS APPLICATIONS**Creating windows application:****Steps to create Windows Application:**

1. To create an application based on windows forms, select the new project from file menu, the new project dialog box will appear on screen, select the windows application and give it a new name "winwelcome" and click on OK button of new project dialog box.
2. Now new form will appear on screen with name form1.
Suppose we want when we click on a button, we will get some text in textbox like "Welcome from visual basic .NET".
3. Drag a textbox and a button control on form1 from toolbox.
4. Set the text property of Button1 as "ClickMe"

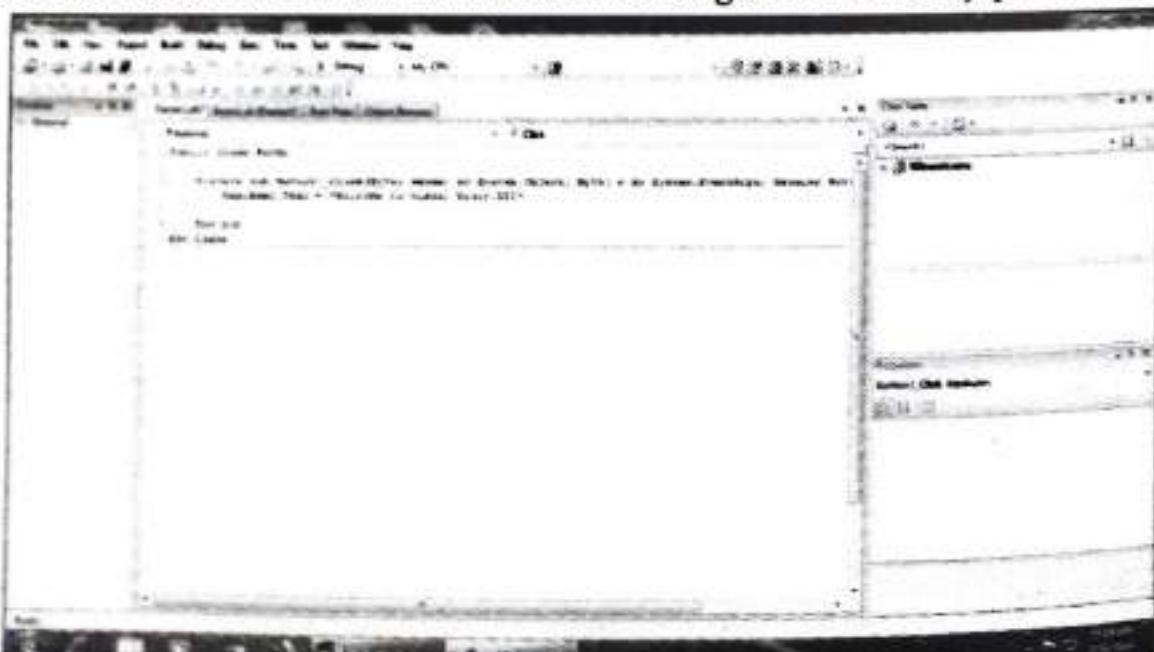
Double Click the button on design time, the code window will appear on screen

```

Public Class Form1
    Private Sub Button1_Click (ByVal Sender asSystem.Object, ByVal e
        AsSystem.EventArgs) Handles Button1.Click
        //Add your code here...
    End Sub
End Class

```

To place the text, we want a textbox control on the form, when user click on the button textbox will shown the desired message that actually put in the code.



Save your application and to run the project first selecting the winv properties from project menu and set startup form as form1.



Run the project by pressing F5 or click on start icon on toolbar. The application starts and when you click on "Click Me" button the message show out on screen as:

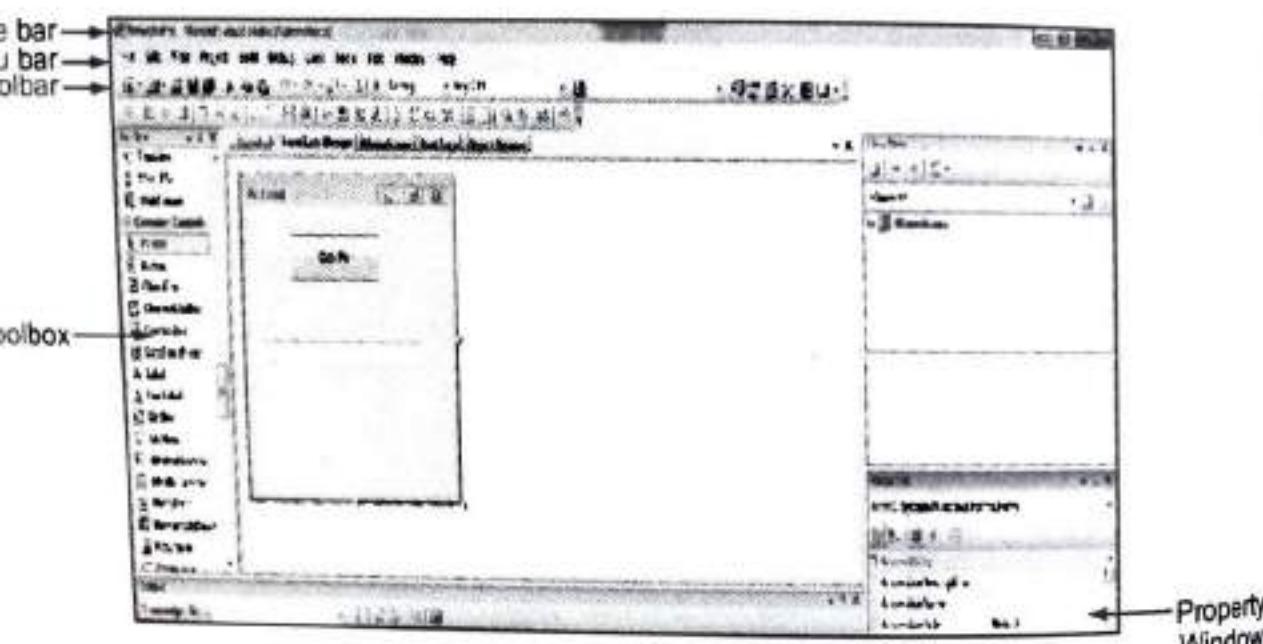
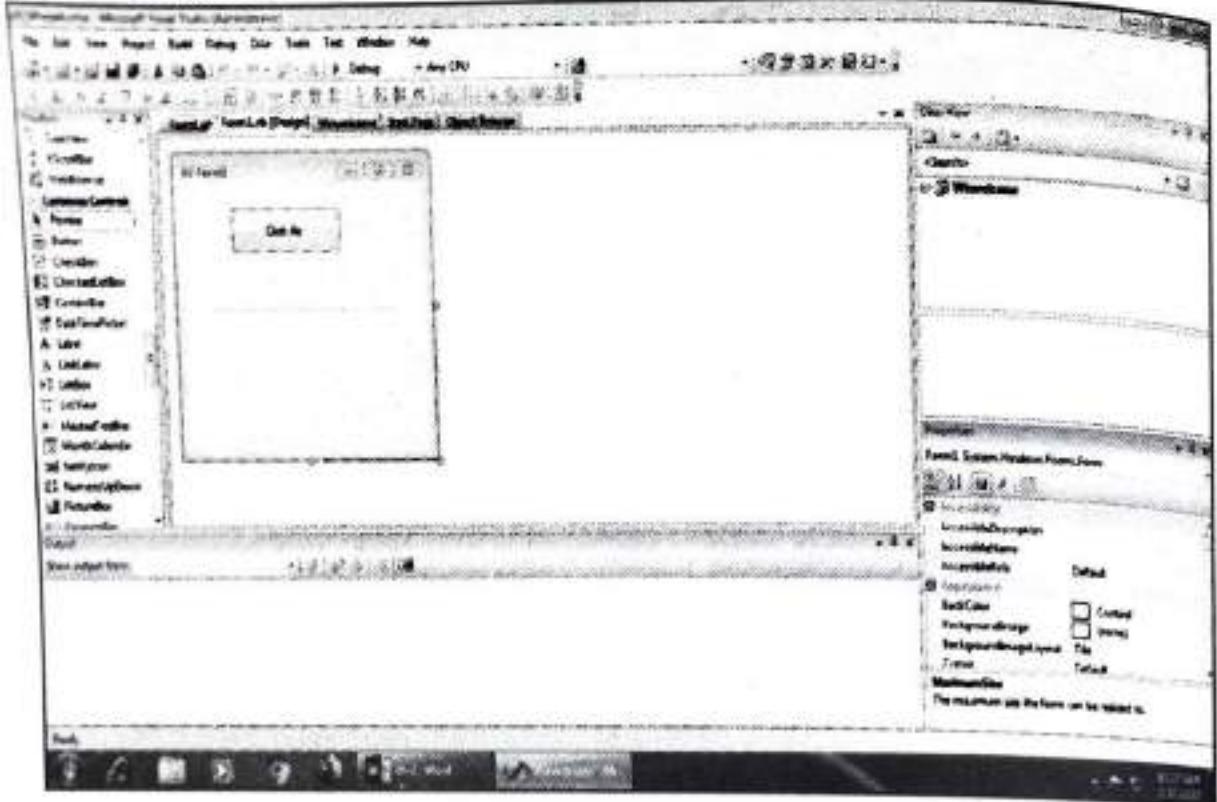


2.3.1 Controls in VB.NET

1. Windows Forms:

- Forms represent the window that will appear in your application.
- In VB.NET working environment, user can develop forms visually, controls and other items from the toolbox.
- VB.NET support for windows forms is in the System.Windows.Forms namespace and the form class is System.Windows.Forms.form
- The form class itself is based on the control class, which means that form has a lot of the methods and properties that controls do.
- Form is a kind of control which acts as another control holder which is used in the Visual Basic Integrated Development Environment (IDE) which handles several aspects of form.

Form Designer window in VB.Net is as:



Properties, Methods and Events:

The following are the most important list of properties related to a form. And these properties can be set or read while the application is being executed.

Properties	Description
BackColor	It is used to set the background color for the form.
BackgroundImage	It is used to set the background image of the form.
Cursor	It is used to set the cursor image when it hovers over the form.

contd..

AllowDrop	Using the AllowDrop control in a form, it allows whether to drag and drop on the form.
Font	It is used to get or set the font used in a form.
Locked	It determines whether the form is locked or not.
FormBorderStyle	It is used to set or get border style in a form.
Text	It is used to set the title for a form window.
MinimizeBox	MinimizeBox It is used to display the minimum option on the title bar of the form.
IsMDIChild	It is used to authenticate whether a form is a container of a Multiple Document Interface (MDI) child form.
Autoscroll	It allows whether to enable auto-scrolling in a form.
MaximizeBox	It is used to display the maximum option on the title bar of the form.
MaximumSize	It is used to set the maximum height and width of the form.
Language	It is used to specify the localized language in a form.
AcceptButton	It is used to set the form button if the enter key is pressed.
Top, Left	It is used to set the top-left corner coordinates of the form in pixel.
Name	It is used to define the name of the form.
MinimumSize	It is used to set the minimum height and width of the form.
Enabled	It uses the True or False value to enable mouse or keyboard events in the form.
TopMost	It uses a Boolean value that represents whether you want to put the window form on top of the other form. By default, it is False.

Form Events:

- The following are the most important list of events related to a form.

Events	Description
Activated	An activated event is found when the user or program activates the form.
Click	A click event is active when the form is clicked.
Closed	A closed event is found before closing the form.
Closing	It exists when a form is closing.
DoubleClick	The DoubleClick event is activated when a user double clicks on the form.

contd. ...

DragDrop	A DragDrop event is activated when a drag and drop operation is performed.
MouseDown	A MouseDown event is activated when the mouse pointer is on the form, and the mouse button is pressed.
GotFocus	A GotFocus event is activated when the form control receives a focus.
HelpButtonClicked	It is activated when a user clicked on the help button.
KeyDown	A KeyDown event is activated when a key is pressed while focusing on the form.
KeyUp	A KeyUp event is activated when a key is released while focusing on the form.
Load	The load event is used to load a form before it is first displayed.
LostFocus	It is activated when the form loses focus.
MouseEnter	A MouseEnter event is activated when the mouse pointer enters the form.
MouseHover	A MouseHover event is activated when the mouse pointer put on the form.
MouseLeave	A MouseLeave event is activated when the mouse pointer leaves the form surface.
Shown	It is activated whenever the form is displayed for the first time.
Scroll	A Scroll event is activated when a form is scrolled through a user or code.
Resize	A Resize event is activated when a form is resized.
Move	A Move event is activated when a form is moved.

Form Methods:

- The following are some of the commonly used methods of the Form class. You can refer to Microsoft documentation for a complete list of methods associated with forms control:

Sr. No.	Method	Description
1.	Activate	Activates the form and gives it focus.
2.	ActivateMdiChild	Activates the MDI child of a form.
3.	AddOwnedForm	Adds an owned form to this form.
4.	BringToFront	Brings the control to the front of the z-order.

contd...

5.	CenterToParent	Centers the position of the form within the bounds of the parent form.
6.	CenterToScreen	Centers the form on the current screen.
7.	Close	Closes the form.
8.	Contains	Retrieves a value indicating whether the specified control is a child of the control.
9.	Focus	Sets input focus to the control.
10.	Hide	Conceals the control from the user.
11.	Refresh	Forces the control to invalidate its client area and immediately redraw itself and any child controls.
12.	Scale (SizeF)	Scales the control and all child controls by the specified scaling factor.
13.	ScaleControl	Scales the location, size, padding, and margin of a control.
14.	ScaleCore	Performs scaling of the form.
15.	Select	Activates the control.
16.	SendToBack	Sends the control to the back of the z-order.
17.	SetAutoScrollMargin	Sets the size of the auto-scroll margins.
18.	SetDesktopBounds	Sets the bounds of the form in desktop coordinates.
19.	SetDesktopLocation	Sets the location of the form in desktop coordinates.
20.	SetDisplayRectLocation	Positions the display window to the specified value.
21.	Show	Displays the control to the user.
22.	ShowDialog	Shows the form as a modal dialog box.

Setting Forms Initial Position:

- We can use a forms StartPosition property to specify its initial position on the screen.
- We assign this property values from the FormStartPosition enumeration.

Possible Property values are as below:

1. **WindowsDefaultBounds:** The form is positioned at the windows default location and has the bounds determined by windows default.
2. **WindowsDefaultLocation:** The form is positioned at the windows default location and has the dimensions specified in the forms size.
3. **CenterScreen:** The form is centered on the current display and has the dimensions specified in the form size.

4. **CenterParent:** The form is centered within the bounds of its parent form.
5. **Manual:** The Location and Size properties of the form will determine its starting position.

Following code shows how you can set a forms startPosition Property:

```
Form1.StartPosition=FormStartPosition.CenterScreen
```

Minimizing and Maximizing a Form:

Basically, forms come with minimizing and maximizing buttons as well as a close box at upper right.

To remove these buttons, we can set the forms ControlBox property to false.

We can also remove the minimizing and maximizing buttons independently, with the MaximizeBox and MinimizeBox properties.

We can set the WindowState property to maximize or minimize them,

- o **FormWindowState.Maximized:** Window is Maximized
- o **FormWindowState.Minimized:** Window is Minimized
- o **FormWindowState.Normal:** Window is set to normal state

Working with Multiple Forms:

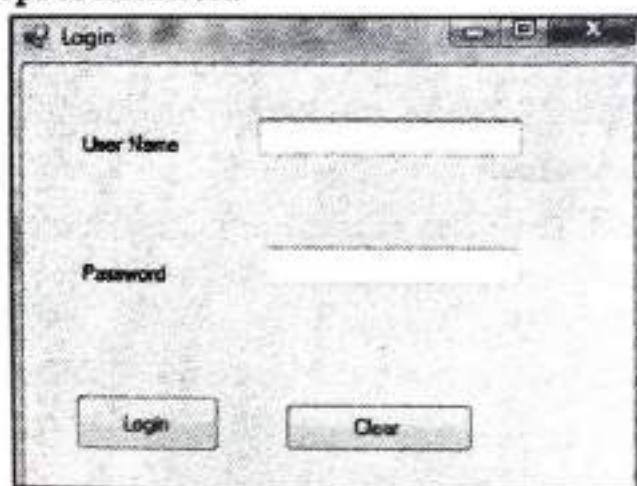
User can deal with multiple forms in single project it is possible for the user to communicate between the forms.

VB.NET Platform provides a choice to the user to decide which form to be executes first and then navigate further between forms.

Let us demonstrate above concept with example stated below:

Create Login form window, once user login successfully the execution control will transfer to the next form where further navigation can be done.

1. First form is Form1 i.e. Login form
2. On the Form1, take two label, two textboxes and two buttons from toolbar and design the form as shown below.
3. Set Password textbox PasswordChar Property as *, so that entered password will be interpret in * representation.



4. Enter code below in a code window:

```

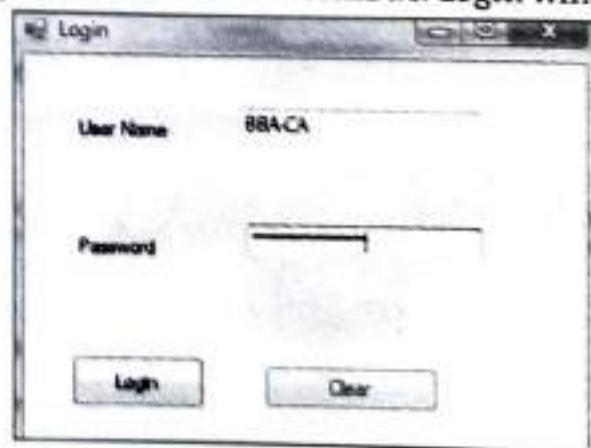
public Class Form1
    private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        if TextBox1.Text = "BBA-CA" And TextBox2.Text = "authentication123" Then
            Me.Hide()
            Form2.ShowDialog()
            Me.Close()
        Else
            MessageBox.Show("Invalid user")
        End If
    End Sub
    private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button2.Click
        TextBox1.Text = ""
        TextBox2.Text = ""
    End Sub
End Class

```

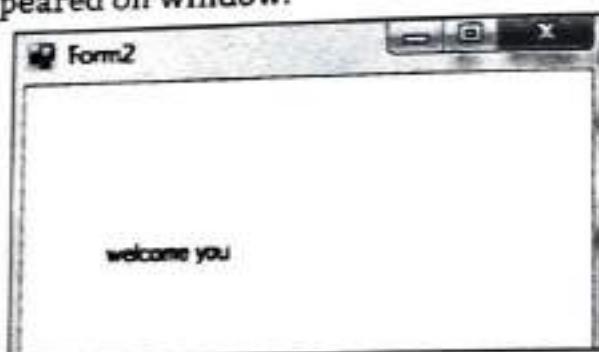
5. When incorrect username and/or password is entered by user then below message box will appears on screen.



6. When run the application, the first Form1 i.e. Login window is appears.

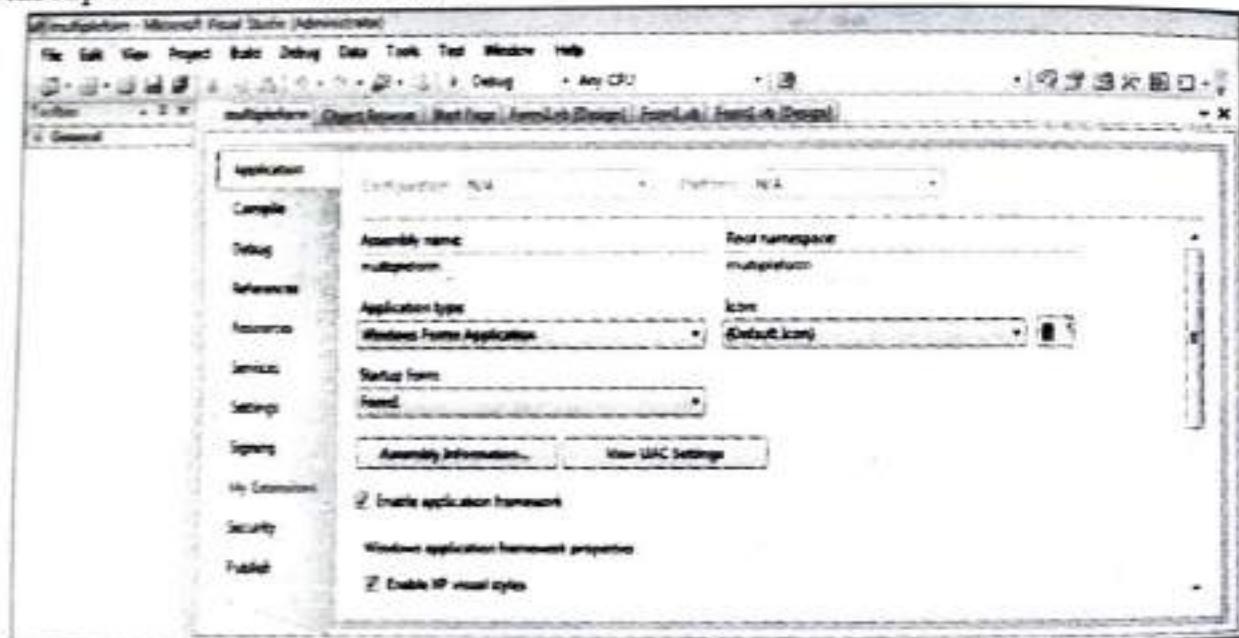


7. Once the correct user name and password is entered, immediately the next form i.e. Form2 will appear on window.



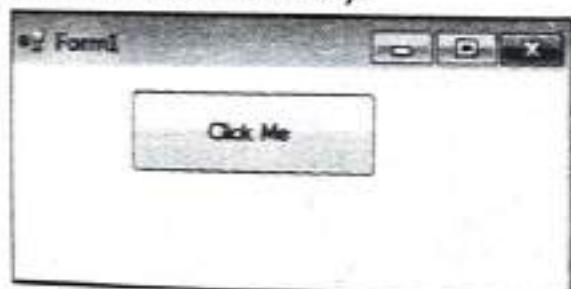
Setting the Start-up Form:

- To execute the program and display the login form first, change the project properties.
- Open menu item project -> Properties -> Select the application tab and change the startup form to Form1 as below:



Adding and Removing Control at Run Time:

- User can add or remove controls on Application at run time, all that we need to do is to use the forms controls collections Add or Remove methods.
- By using following code user can deal with it as:
 - Initially Form window is designed as below without any textbox control on it (we will go to add textbox control at runtime):

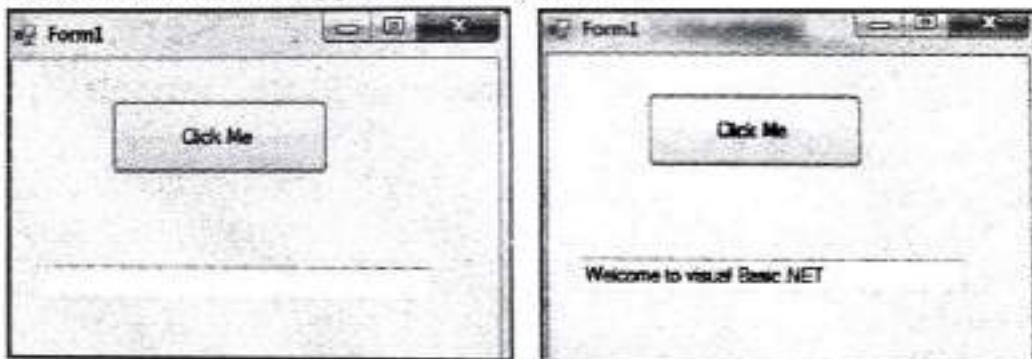


2. add below code on button1_Click() event:

```
public Class Form1
    private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
        System.EventArgs) Handles Button1.Click
        Dim New TextBox As New TextBox()
        New TextBox.Size = New Size(80, 20)
        New TextBox.Location = New Point(90, 80)
        New TextBox.Text = "Welcome to Visual Basic.NET"
        Me.Controls.Add(New TextBox)
    End Sub
End Class
```

3. Run the application:

When the application is run, below window will shown out which contains textbox and a text will appear in it at run time.



Docking and Anchoring Control:

- Docking and anchoring is used to make a control covers an entire client area of the form.
- Anchoring is used to resize controls dynamically with the form.

Docking:

- The rich text box covers complete client area of form, user can dock and anchor it.
- When we dock a control, it adheres to the edges of its container form.
- To dock a control, select the Dock property of that control from the properties window.
- Once user Select the dock property it opens up a small editor from which user can select to which side on the form the control should be docked.

Anchoring:

- To anchor a control, select the anchor property of that control from the properties window.
- Once user Select the anchor property it displays an editor that shows a cross, to set an anchor, click the top, left, right or bottom section of the cross.
- By default, controls are anchored to the top and left side.

2. TextBox Control:

- This control is used to accept input from user or to display text.
- Textboxes can display multiple lines wrap text to the size of control.
- The TextBox is based on the TextBoxBase class which is based on the Control class.
- By default, user can enter up to 2048 characters in a TextBox but if the multiline property is set to True user can enter up to 32KB of text.

TextBox Properties

Property Name	Description
Text	Used to set the text for the textbox.
BackColor	Set the background color of textbox, if set the same as ForeColor property, the text will not be display.
ForeColor	The foreground color of the text box.
Enabled	To disable the textbox, default value is True.
Multiline	By setting this property to True, textbox allows accepting multiple lines of text.
Scrollbars	Allows to add a scrollbar to a Textbox, when the textbox is multiline.
PasswordChar	When the property is set to a specific character representation like say '*', the text that is entered in the TextBox is display as *.
useSystemPasswordChar	Specifies whether system password character should appear in textbox. Values may be false/true.
CharacterCasing	It converts text into uppercase or lowercase. Values are Normal, Upper, Lower.
MaxLength	Specifies maximum number of characters that can be entering in textbox.
ReadOnly	Makes the textbox ReadOnly, doesn't allow entering any text.
Visible	Default value is True, to hide the textbox control set the property to False.

3. RichTextBox Control:

- RichTextboxes are similar to TextBoxes but they provide some advanced features over the standard textboxes.
- It can display, enter, manipulates the text with formatting. It can display font colors, images and links.
- The default event of RichTextBox is the TextChanged event.

4. Label Control:

- Label control is used to display text on the client area of form such as title, paragraph or captions for other control.
- The text displayed by a label control is a text which is set to its text property.

Property	Description
AutoSize	Value indicating that the label will automatically resize to display the entire Text property. Default value is False.
BackgroundImage	The background image displayed on the label.
BorderStyle	Sets the border style for the label.
FlatStyle	Sets the flat style appearance of the label.
Image	The image displayed on the label.
ImageAlign	Align image displayed in the label.
ImageIndex	Zero-based index value of the image object contained in the image list.
TextAlign	Aligns text displayed in the label.

5. LinkLabel Control:

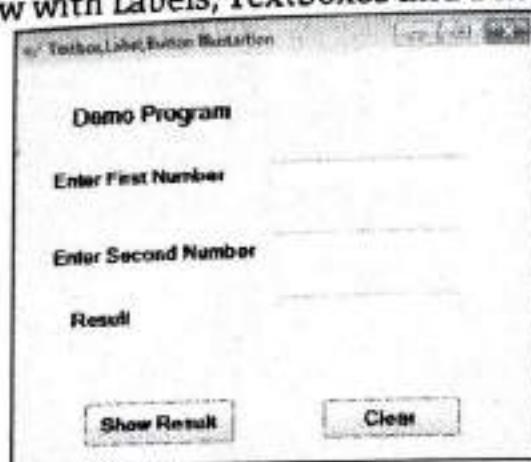
- It is derived from the label control class, which display one or more hyperlinks.
- The control LinkLabel control does not automatically provide actual linkage to anything, nor by default, does it change the color of the link once it has been visited.

6. Button Control:

- A Button is a window control used to initiate an Action.
- From the user's standpoint, a Button is useful when clicked, in which case the user positions the mouse on it and passes one of the mouse buttons.
- When the Button is clicked, it looks as if it is being pushed in and release, whenever the user clicks a button, the click event handler is invoked.
- The Button Class is based on the ButtonBase class which is based on the control class.

Property	Description
FlatStyle	Sets the flat-style appearance of the control, the default value is FlatStyle.Standard.
Image	The Image displayed on the control.
ImageAlign	Aligns image displayed in the label.
ImageIndex	Zero-based index value of the image object contained in the ImageList
ImageList	The image list containing the images displayed in the control, ImageList stores a collection of Image objects.
TextAlign	Align text displayed in the control.

Example 5: To illustrates the implementation of Label, Textbox and Button control: [S-23]
 Maximum Finder among two enter values and display result in third textbox.
 Design the form as below with Labels, Textboxes and Buttons control



```
public Class Form1
private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
Dim a, b As Integer
  a = TextBox1.Text
  b = TextBox2.Text
  if (a > b) Then
    TextBox3.Text = a
  else
    TextBox3.Text = b
  End if
End Sub
private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button2.Click
  TextBox1.Text = ""
  TextBox2.Text = ""
  TextBox3.Text = ""
End Sub
End Class
```

Output:



7. CheckBox Control:

- CheckBox is a small square box that the user can click for an option to select YES/NO or True/False.
- A CheckBox is clicked to select and clicked again to deselect some option, checkbox control is based on textboxbase class which is based on the Control class.

Properties of Checkbox:

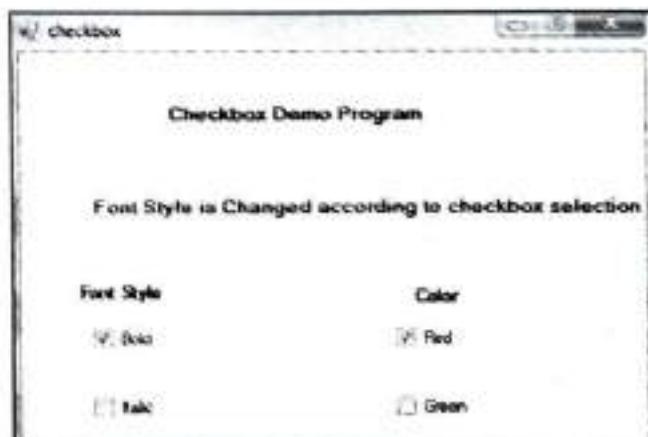
Property	Description
Text	Caption of Checkbox.
Checked	It set to True if user wants the checkbox to be displayed as checked, otherwise default value is False.
CheckState	Default value is Unchecked, if user want a check to appear set it to True.
CheckAlign	Use to set the alignment for the checkbox.

```

public Class checkbox
private Sub CheckBox1_CheckedChanged(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles CheckBox1.CheckedChanged
    Label2.Font = New Font("Microsoft Sans Serif", Label1.Font.Size,
                           Label2.Font.Style Xor FontStyle.Bold)
End Sub
private Sub CheckBox3_CheckedChanged(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles CheckBox3.CheckedChanged
    Label2.Font = New Font("Microsoft Sans Serif", Label1.Font.Size,
                           Label2.Font.Style Xor FontStyle.Italic)
End Sub
private Sub CheckBox2_CheckedChanged(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles CheckBox2.CheckedChanged
    Label2.ForeColor = System.Drawing.Color.Red
End Sub
private Sub CheckBox4_CheckedChanged(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles CheckBox4.CheckedChanged
    Label2.ForeColor = System.Drawing.Color.Green
End Sub
End Class

```

Output:



8. ListBox Control:

- ListBox is a control that allows user to select one or more multiple items from a given list.
- It allows user to add item, remove item etc.

Properties of ListBox:

Property	Description
Sorted	Sort the list of items
Remove	Remove the specified object
RemoveAt	Removes an item with a specified index
Count	Count number of item in a List
Clear	Clear existing list
Insert	Insert an item at specified index
SelectionMode	Allows single or multiple selections
ScrollAlwaysVisible	Default value is set to False, if it is set to True, will display both vertical and horizontal scrollbar always.
MultiColumn	The Default value is set to False, Set it to True if you want the list box to display multiple column.
HorizontalScrollbar	Displays horizontal scrollbar to the listbox.

Example 6: To Illustrate use of Listbox:

```

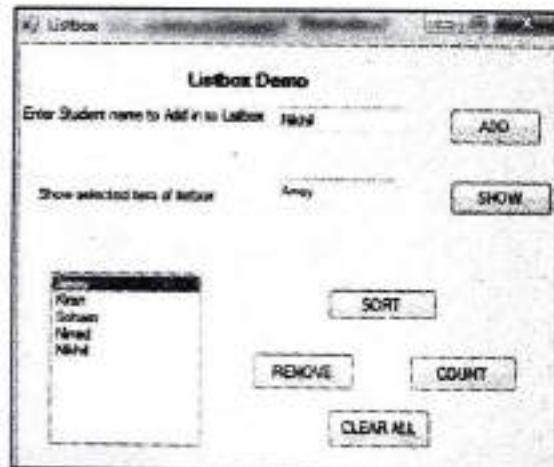
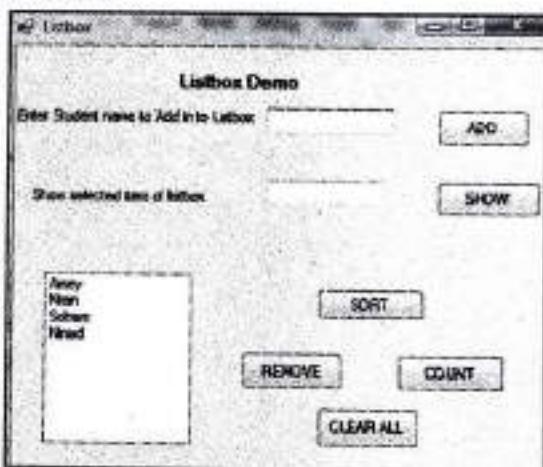
public Class Listbox
    private Sub Listbox_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        ListBox1.Items.Add("Amey")
        ListBox1.Items.Add("Kiran")
        ListBox1.Items.Add("Soham")
        ListBox1.Items.Add("Ninad")
    End Sub
    private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        ListBox1.Items.Add(TextBox1.Text)
    End Sub
    private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button2.Click
        TextBox2.Text = ListBox1.SelectedItem
    End Sub
    private Sub Button3_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button3.Click
        ListBox1.Sorted = True
    End Sub

```

```

private sub Button4_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button4.Click
    ListBox1.Items.RemoveAt(ListBox1.SelectedIndex)
End Sub
private Sub Button5_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button5.Click
    Dim cnt As Integer
    cnt = ListBox1.Items.Count
    MessageBox("number of items in list box:" + cnt.ToString())
End Sub
private Sub Button6_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button6.Click
    ListBox1.Items.Clear()
End Sub
End Class

```



9. ComboBox Control:

[S-23]

- ComboBox is used to display data in a dropdown ComboBox.
- This control is allowed user to select single item from a given list of items.
- ComboBox is made up of two parts, the top part is a textbox that allows user to type in it and second part is a listbox that displays a list of items from which user can select.
- It is a combination of a Textbox and a Listbox.

Properties of ComboBox:

Property	Description
Sorted	Sort the ComboBox items, it is set to False by default
DropDownStyle	Allows user to set the look of ComboBox
Remove	Remove the specified object
RemoveAt	Removes an item with a specified index
Count	Count number of items in a ComboBox list
Clear	Clear existing list from ComboBox
Insert	Insert an item at specified index

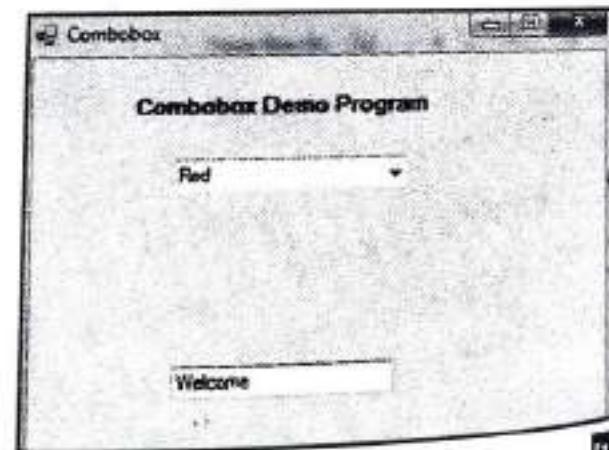
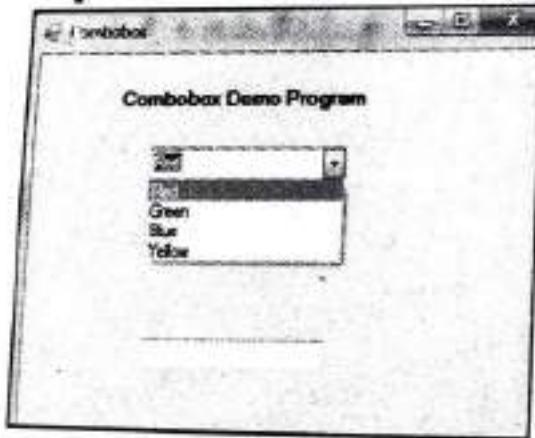
Example 7:

- Design a ComboBox having list of color names, when user select specific color name text in the textbox will appears in that selected color.
- ```

public Class Combobox
 private Sub Combobox_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
 ComboBox1.Items.Add("Red")
 ComboBox1.Items.Add("Green")
 ComboBox1.Items.Add("Blue")
 ComboBox1.Items.Add("Yellow")
 End Sub

 private Sub ComboBox1_SelectedIndexChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles ComboBox1.SelectedIndexChanged
 SelectCase ComboBox1.SelectedIndex
 Case 0
 TextBox1.ForeColor = System.Drawing.Color.Red
 Case 1
 TextBox1.ForeColor = System.Drawing.Color.Green
 Case 2
 TextBox1.ForeColor = System.Drawing.Color.Blue
 Case 3
 TextBox1.ForeColor = System.Drawing.Color.Yellow
 End Select
 End Sub
End Class

```

**Output:****10. RadioButton Control:**

- A radio button is also called an option button which is a circular control, used to select and deselect options.
  - A major difference between checkboxes and radio buttons is user can select only one radio button at a time while in case of checkboxes user can select multiple checkboxes at a time.
- The RadioButtons are mostly used together in group.

**Properties of RadioButton**

| Property   | Description                                                                   |
|------------|-------------------------------------------------------------------------------|
| Checked    | Default value is False, when it set to True RadioButton displayed as checked. |
| CheckAlign | Use to set Alignment for the RadioButton from a predefined list.              |

**Example 8:**

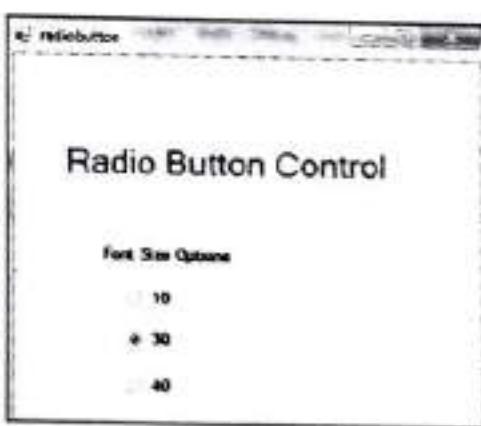
```

public Class radiobutton
 private Sub RadioButton1_CheckedChanged(ByVal sender As System.Object,
 ByVal e As System.EventArgs) Handles RadioButton1.CheckedChanged
 Label1.Font = New Font(Label1.Font.FontFamily, 10)
 End Sub

 private Sub RadioButton2_CheckedChanged(ByVal sender As System.Object,
 ByVal e As System.EventArgs) Handles RadioButton2.CheckedChanged
 Label1.Font = New Font(Label1.Font.FontFamily, 20)
 End Sub

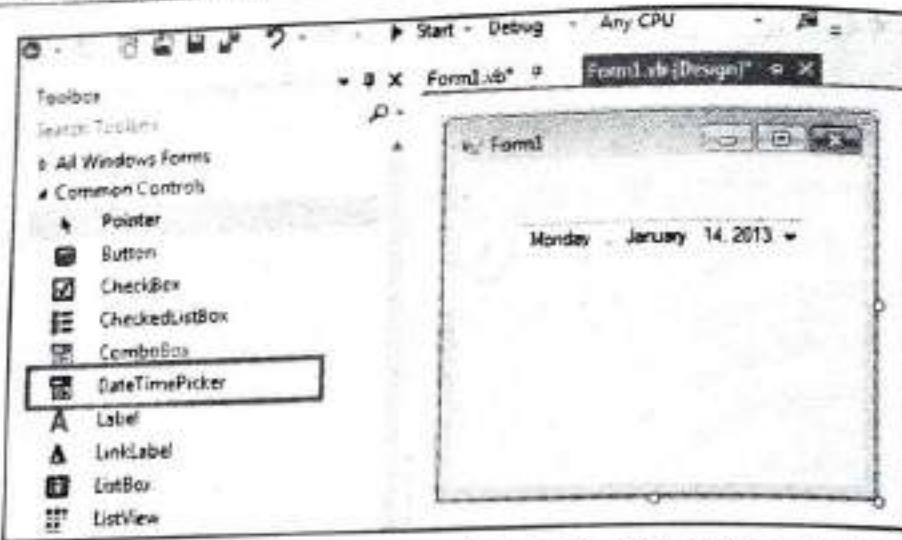
 private Sub RadioButton3_CheckedChanged(ByVal sender As System.Object,
 ByVal e As System.EventArgs) Handles RadioButton3.CheckedChanged
 Label1.Font = New Font(Label1.Font.FontFamily, 25)
 End Sub
End Class

```

**Output:****11. DateTimePicker Control:**

[S-23]

- The DateTimePicker control allows selecting a date and time by editing the displayed values in the control. If you click the arrow in the DateTimePicker control, it displays a month calendar, like a combo box control. The user can make selection by clicking the required date. The new selected value appears in the text box part of the control.



The following are some of the commonly used properties of the DateTimePicker control:

| Sr. No. | Property                     | Description                                                                                                                                         |
|---------|------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| 1.      | BackColor                    | Gets or sets a value indicating the background color of the DateTimePicker control.                                                                 |
| 2.      | BackgroundImage              | Gets or sets the background image for the control.                                                                                                  |
| 3.      | BackgroundImageLayout        | Gets or sets the layout of the background image of the DateTimePicker control.                                                                      |
| 4.      | CalendarFont                 | Gets or sets the font style applied to the calendar.                                                                                                |
| 5.      | CalendarForeColor            | Gets or sets the foreground color of the calendar.                                                                                                  |
| 6.      | CalendarMonthBackgroundColor | Gets or sets the background color of the calendar month.                                                                                            |
| 7.      | CalendarTitleBackColor       | Gets or sets the background color of the calendar title.                                                                                            |
| 8.      | CalendarTitleForeColor       | Gets or sets the foreground color of the calendar title.                                                                                            |
| 9.      | CalendarTrailingForeColor    | Gets or sets the foreground color of the calendar trailing dates.                                                                                   |
| 10.     | Checked                      | Gets or sets a value indicating whether the Value property has been set with a valid date/time value and the displayed value is able to be updated. |

contd...

|     |                   |                                                                                                                                         |
|-----|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| 11. | CustomFormat      | Gets or sets the custom date/time format string.                                                                                        |
| 12. | DropDownAlign     | Gets or sets the alignment of the drop-down calendar on the DateTimePicker control.                                                     |
| 13. | ForeColor         | Gets or sets the foreground color of the DateTimePicker control.                                                                        |
| 14. | Format            | Gets or sets the format of the date and time displayed in the control.                                                                  |
| 15. | MaxDate           | Gets or sets the maximum date and time that can be selected in the control.                                                             |
| 16. | MaximumDateTime   | Gets the maximum date value allowed for the DateTimePicker control.                                                                     |
| 17. | MinDate           | Gets or sets the minimum date and time that can be selected in the control.                                                             |
| 18. | MinimumDateTime   | Gets the minimum date value allowed for the DateTimePicker control.                                                                     |
| 19. | PreferredHeight   | Gets the preferred height of the DateTimePicker control.                                                                                |
| 20. | RightToLeftLayout | Gets or sets whether the contents of the DateTimePicker are laid out from right to left.                                                |
| 21. | ShowCheckBox      | Gets or sets a value indicating whether a check box is displayed to the left of the selected date.                                      |
| 22. | ShowUpDown        | Gets or sets a value indicating whether a spin button control (also known as an up-down control) is used to adjust the date/time value. |
| 23. | Text              | Gets or sets the text associated with this control.                                                                                     |
| 24. | Value             | Gets or sets the date/time value assigned to the control.                                                                               |

**12. MonthCalendar Control:**

- The MonthCalendar control displays a calendar page that enables the user to select one or more dates. The user can scroll to additional months by using arrow buttons located at the top of the control.
- The user selects a date from the MonthCalendar control by clicking the date displayed on the calendar page. By default, the user can select a range of dates. Initially, the user is allowed to select a range of up to seven dates, to change this behavior, use the **MaxSelectionCount** property.

### Properties of MonthCalendar

- FirstDayOfWeek:** The Default value is Sunday. It means week starts with Sunday as the first day and Saturday as last. You can set the first day of the week depending upon your choice by selecting from the predefined list with this property.
- ShowToday:** The Default value is set to true which displays the current date at the bottom of the Calendar. When set it to False will hide the current date.
- ShowTodayCircle:** The Default value is set to true which displays a circle on the current date. Setting it to False will make the circle disappear.
- ShowWeekNumber:** The Default is False. Setting it to True will display the week number of the current week in the 52 week year. That will be displayed toward the left side of the control.

**Example 9:** A Program to illustrate Month Calendar, when user select date from calendar, it will display in the below of form.

```
public Class Form11
 private Sub MonthCalendar1_DateChanged (ByVal sender As System.Object,
 ByVal e As System.Windows.Forms.DateRangeEventArgs) Handles
 MonthCalendar1.DateChanged
 'selected date will display in Label
 Label2.Text = 'Selected date is '+MonthCalendar1.SelectionRange.Start
 End Sub
End Class
```

#### Output:



### 3. Timer Control:

The TimerControl allows you to set a time interval to execute an event after that interval continuously.

This control is useful when you want to execute certain application after a certain interval.

- For example, if user wants to create a backup of your data processing in every hour, user can make a routine which will take the backup and call that routine on Timers event and set time interval for an hour.
- Timer is a component that is added to the tray at design time, as a component it has no parent property.
- The most important property of Timer control is interval.
- Lower range for the Interval property is 1.

#### Properties of Timer Control:

| Property | Description                                     |
|----------|-------------------------------------------------|
| Enabled  | Disable the timer event default value is false  |
| Interval | The Frequency of Elapsed events in milliseconds |

**Example 10:** Design a form with one timer control, two labels and two textboxes and run the code mentioned below:

[S-22]

```

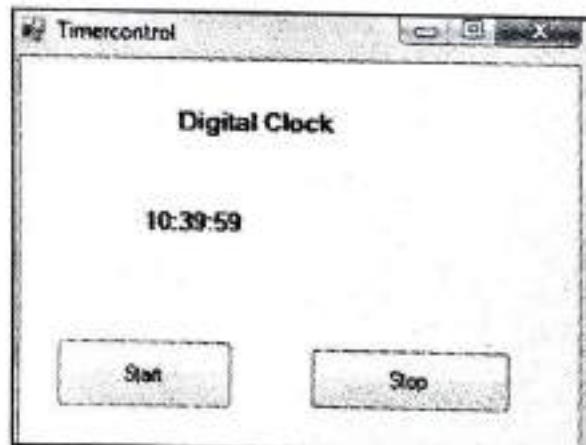
public Class Timercontrol
 Dim num As Integer = 0

 private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
 System.EventArgs) Handles Button1.Click
 Timer1.Enabled = True
 End Sub

 private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
 System.EventArgs) Handles Button2.Click
 Timer1.Enabled = False
 End Sub

 private Sub Timer1_Tick(ByVal sender As Object, ByVal e As
 System.EventArgs) Handles Timer1.Tick
 Label2.Text = Format(DateTime.Now, "hh:mm:ss")
 End Sub
End Class

```



**14. Progressbar Control:**

- Progressbar is a graphical representation to show the progress of any time consuming activity.
- It can be used when user wants to show the progress of activity such as copying large data, moving files etc.

**Properties of Progress bar:**

| Property | Description                                                                                 |
|----------|---------------------------------------------------------------------------------------------|
| Minimum  | It is the initial value to set the starting value of a progress bar.                        |
| Maximum  | It is a possible max value to set for progress bar                                          |
| Value    | This property has to change the value between min and max to show the progress graphically. |

**Example 11: Demonstration of Progressbar.**

```

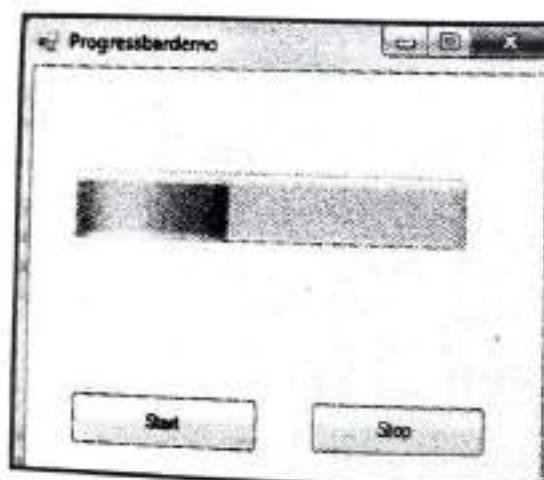
public Class Progressbardemo

 private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
 System.EventArgs) Handles Button1.Click
 Timer1.Enabled = True
 End Sub

 private Sub Timer1_Tick(ByVal sender As Object, ByVal e As
 System.EventArgs) Handles Timer1.Tick
 ProgressBar1.Value += 1
 if ProgressBar1.Value = ProgressBar1.Maximum Then
 Timer1.Enabled = False
 End If
 End Sub

 private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
 System.EventArgs) Handles Button2.Click
 Timer1.Enabled = False
 End Sub
End Class

```



**15. Scrollbar Control:**

- Scrollbar is a Vertical or Horizontal long stripe with an indicator that allows user to select a value between the two ends of the control.
- In VB.NET there are two Scrollbar controls i.e., HScrollBar and VScrollBar
- One end indicates its minimum value where as another end indicates its maximum value.
- The Current value of the control is determined by the position of indicator which is scrolled between minimum and maximum values.
- Users can also implements scrollbars at runtime.

**Properties of Scrollbar:**

| Property | Description                                                  |
|----------|--------------------------------------------------------------|
| Minimum  | Controls the Minimum value, the default value is 0           |
| Maximum  | Controls the Maximum value, the default value is 100         |
| Value    | Indicate current value of control by the indicators position |

**Example 12:**

```

public Class Form1
 private Sub Form1_Load (sender As Object, e As EventArgs) _
 Handles MyBase.Load

 'Create two scroll bars
 Dim hs As HScrollBar
 Dim vs As VScrollBar
 hs = New HScrollBar()
 vs = New VScrollBar()

 'set properties
 hs.Location = New Point(10, 200)
 hs.Size = New Size(175, 15)
 hs.Value = 50
 vs.Location = New Point(200, 30)
 vs.Size = New Size(15, 175)
 hs.Value = 50

 'adding the scroll bars to the form
 Me.Controls.Add(hs)
 Me.Controls.Add(vs)
 End Sub
End Class

```

**16. PictureBox Control:**

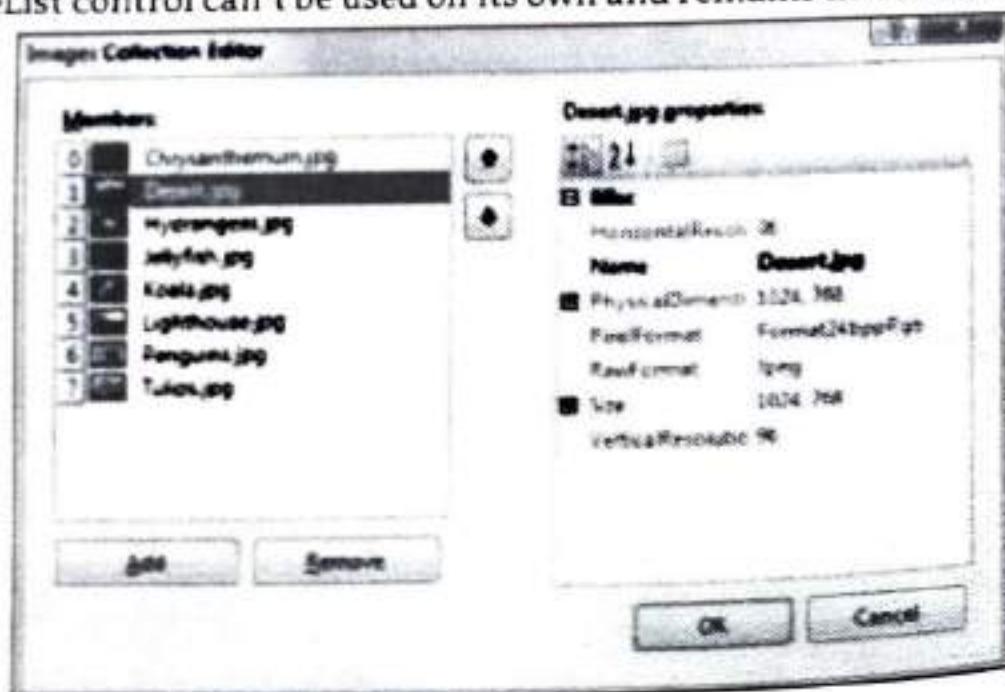
- The PictureBox control allows user to set a picture in it.
- Size Mode is one of the important properties of PictureBox.
- User can set the image as Normal, StretchImage, AutoSize, CenterImage and Zoom.

**Example 13:** To Stretch the image in the PictureBox as per the dimension that we set.

```
public Class pictureBox
 private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
 System.EventArgs) Handles Button1.Click
 PictureBox1.ClientSize = New Size(100, 100)
 PictureBox1.SizeMode = PictureBoxSizeMode.StretchImage
 End Sub
End Class
```

**7. ImageList Control:**

The ImageList is a simple control that stores images used by other controls at runtime. For example, a TreeView control can use icons to identify its nodes. The simplest method of preparing these images is to create an ImageList control and add to it all the icons you need. The ImageList control maintains a series of bitmaps in memory that the TreeView control can access quickly at runtime. Keep in mind that the ImageList control can't be used on its own and remains invisible at runtime.



**18. Tree View Control:**

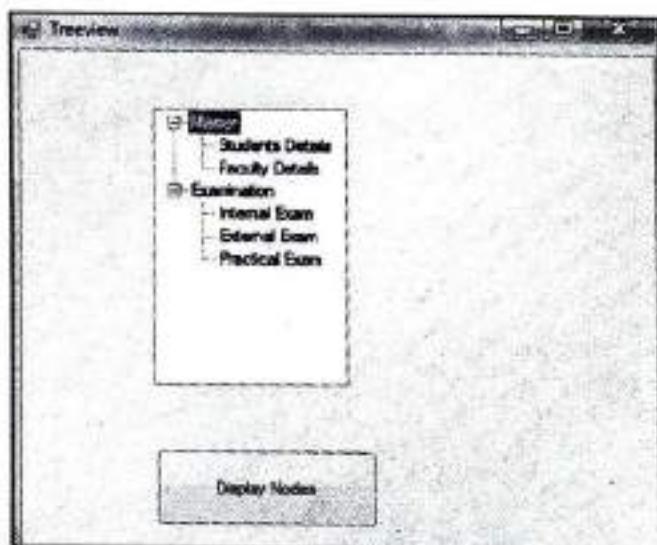
- The tree view control is used to display a hierarchy of nodes.
- User can expand and collapse these nodes by clicking them.
- It is similar to Windows Explorer which displays a tree view in its left pane to list all the folders on the hard disk.
- This control allows user to add nodes to a tree each of which can have sub items.

**Properties of Tree View:**

| Property     | Description                                          |
|--------------|------------------------------------------------------|
| FullPath     | Gets the path from the root node to the current node |
| Index        | Gets the location of the node in the node collection |
| Nodes        | Gets the collection of nodes in the current node.    |
| Parent       | Gets the parent node of the current node             |
| PreNode      | Gets the previous sibling node                       |
| Selectednode | Find out selected node                               |
| Name         | Find out named of node                               |
| Text         | Find out caption of node                             |

**Example 14:** To design Tree view control Application.

```
public Class Treeview
 private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
 System.EventArgs) Handles Button1.Click
 TreeView1.Nodes.Add("Master")
 TreeView1.Nodes(0).Nodes.Add("Students Details")
 TreeView1.Nodes(0).Nodes.Add("Faculty Details")
 TreeView1.Nodes.Add("Examination")
 TreeView1.Nodes(1).Nodes.Add("Internal Exam")
 TreeView1.Nodes(1).Nodes.Add("External Exam")
 TreeView1.Nodes(1).Nodes.Add("Practical Exam")
 End Sub
End Class
```



**19. ListView Control:**

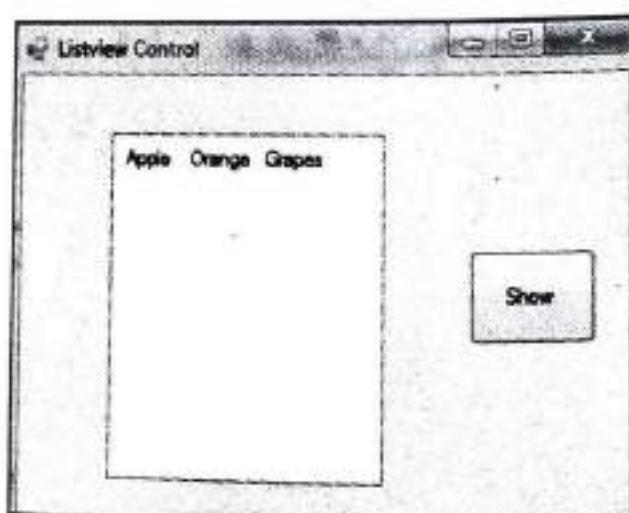
- The ListView control is used to display a list of items.
- It allows user to create a Windows Explorer like interface. Along with tree view control.
- The following are some of the commonly used properties of the ListView control:

| Sr. No. | Property       | Description                                                                                                                              |
|---------|----------------|------------------------------------------------------------------------------------------------------------------------------------------|
| 1.      | Alignment      | Gets or sets the alignment of items in the control.                                                                                      |
| 2.      | AutoArrange    | Gets or sets whether icons are automatically kept arranged.                                                                              |
| 3.      | BackColor      | Gets or sets the background color.                                                                                                       |
| 4.      | Checkboxes     | Gets or sets a value indicating whether a check box appears next to each item in the control.                                            |
| 5.      | CheckedIndices | Gets the indexes of the currently checked items in the control.                                                                          |
| 6.      | CheckedItems   | Gets the currently checked items in the control.                                                                                         |
| 7.      | Columns        | Gets the collection of all column headers that appear in the control.                                                                    |
| 8.      | GridLines      | Gets or sets a value indicating whether grid lines appear between the rows and columns containing the items and subitems in the control. |

**Example 15:** To design List view control.

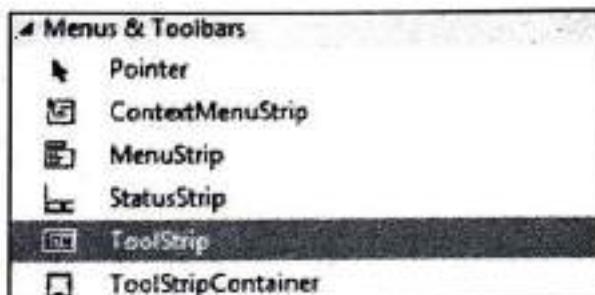
```
public Class Form1
 private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
 System.EventArgs) Handles Button1.Click
 ListView1.Items.Add("Apple")
 ListView1.Items.Add("Orange")
 ListView1.Items.Add("Grapes")
 End Sub
End Class
```

**Output:**

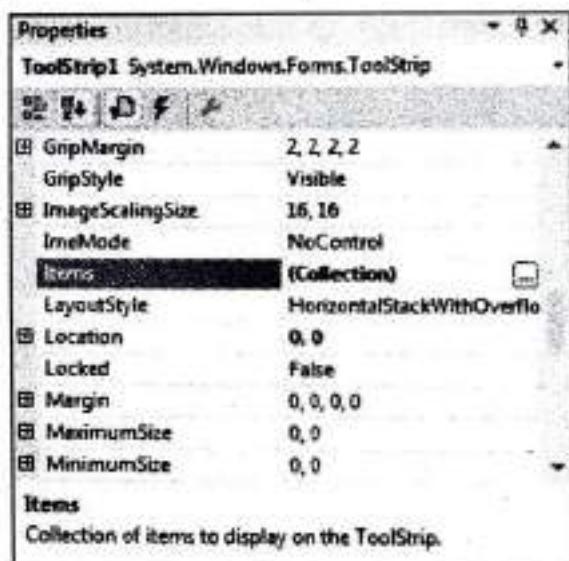


## 20. Toolbar Control:

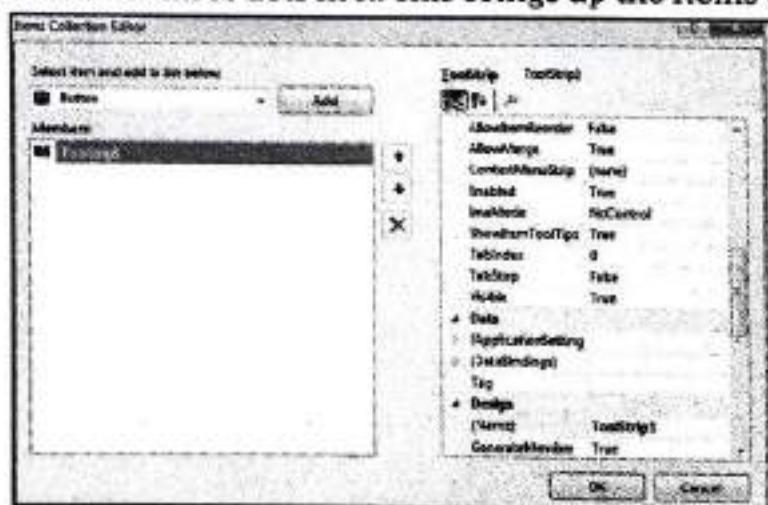
- The toolbar is a very popular and much-used addition to a programme.
- VB.NET lets you add toolbars to your forms, and the process is quite straightforward.
- To add a toolbar to the top of your form, expand the Toolbox and locate the ToolStrip control:



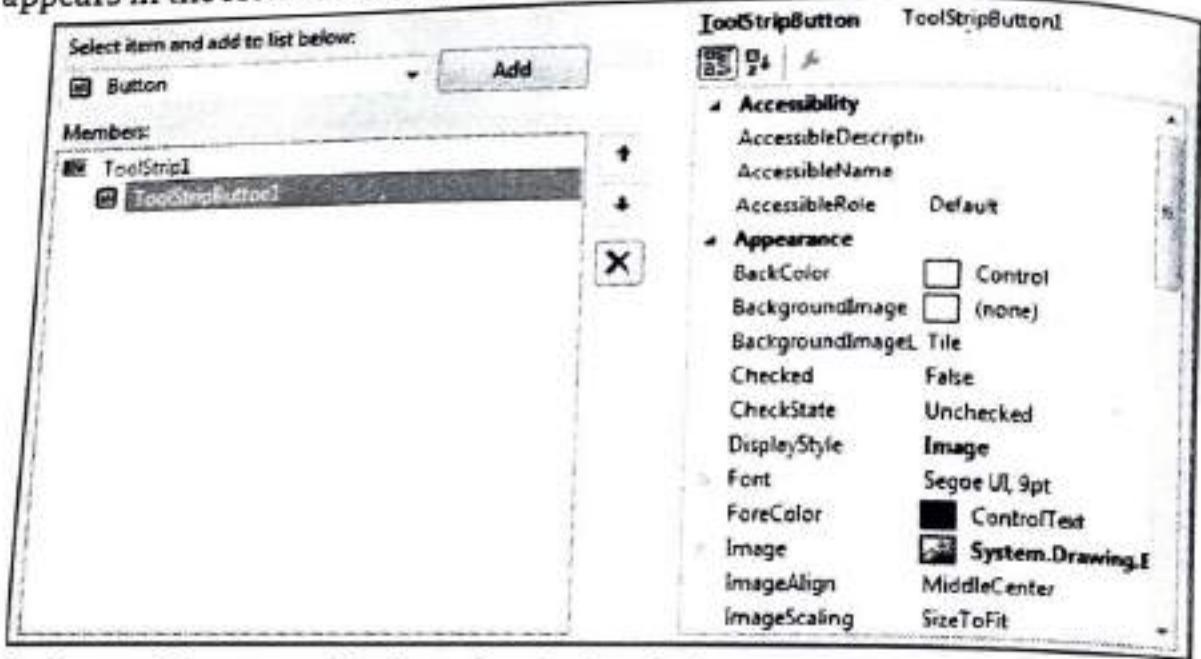
- ToolStrips work by adding buttons and images to them. The button is then clicked, and an action performed.
- Click on your ToolStrip to select it. In the property box for the ToolStrip, you'll notice that it has the default Name of ToolStrip1. We'll keep this Name. But locate the Items (Collection) property:



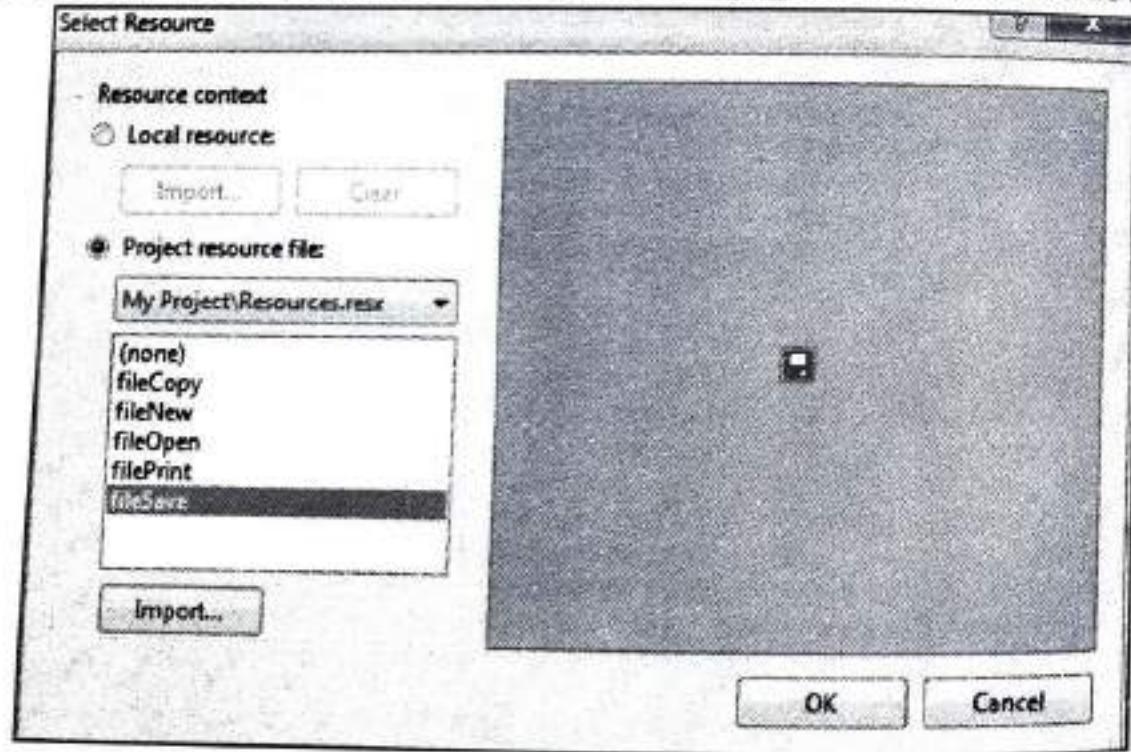
- Click the button with the three dots in it. This brings up the Items Collection Editor:



- To add a new button to your ToolStrip, click the Add button at the top. The button appears in the Members box (ToolStripButton1):



- Click the small button with the 3 dots in it to bring up the Select Resource box:



## .. StatusBar:

A StatusBar control is a combination of StatusBar panels where each panel can be used to display different information. For example, one panel can display current application status and other can display date and other information and so on.

A status bar is commonly used to provide hints for the selected item or information about an action currently being performed on a dialog. Normally, the StatusBar is placed at the bottom of the screen, as it is in MS-Office applications and Paint, but it can be located anywhere you like.

- The StatusBarPanel class contains all the information about the individual panels in the Panels collection. The information that can be set ranges from simple text and alignment of text to icons to be displayed and the style of the panel.

#### StatusBar Properties:

| Name            | Availability | Description                                                                                       |
|-----------------|--------------|---------------------------------------------------------------------------------------------------|
| BackgroundImage | Read/Write   | It is possible to assign an image to the status bar that will be drawn in the background.         |
| Panels          | Read/Write   | This is the collection of panels in the status bar. Use this collection to add and remove panels. |
| ShowPanels      | Read/Write   | If you want to display panels, this property must be set to true.                                 |
| Text            | Read/Write   | Text                                                                                              |

#### Status Bar Events:

| Name       | Description                                                                                                                                                      |
|------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| DrawItem   | Occurs when a panel that has the OwnerDraw style set needs to be redrawn. You must subscribe to this event if you want to draw the contents of a panel yourself. |
| PanelClick | Occurs when a panels is clicked.                                                                                                                                 |

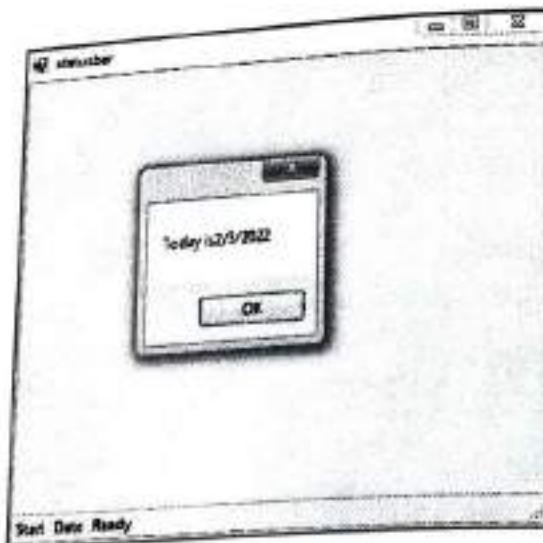
#### Example 16: For StatusBar.

- To understand StatusBar control drag and drop status strip and make text blank and add 3 toolStripStatusLabel, change text property of label and write code on each toolStripLabels click event as shown below-

```

public Class statusbar
 private Sub ToolStripStatusLabel1_Click(ByVal sender As System.Object,
 ByVal e As System.EventArgs) Handles ToolStripStatusLabel1.Click
 MessageBox.Show("Start the Application")
 End Sub
 private Sub ToolStripStatusLabel2_Click(ByVal sender As System.Object,
 ByVal e As System.EventArgs) Handles ToolStripStatusLabel2.Click
 Dim thisDate As Date
 thisDate = Today
 MessageBox.Show("Today is" + thisDate)
 End Sub
 private Sub ToolStripStatusLabel3_Click(ByVal sender As System.Object,
 ByVal e As System.EventArgs) Handles ToolStripStatusLabel3.Click
 MessageBox.Show("System status is optimal")
 End Sub
End Class

```

**Output:****22. DataGridView:**

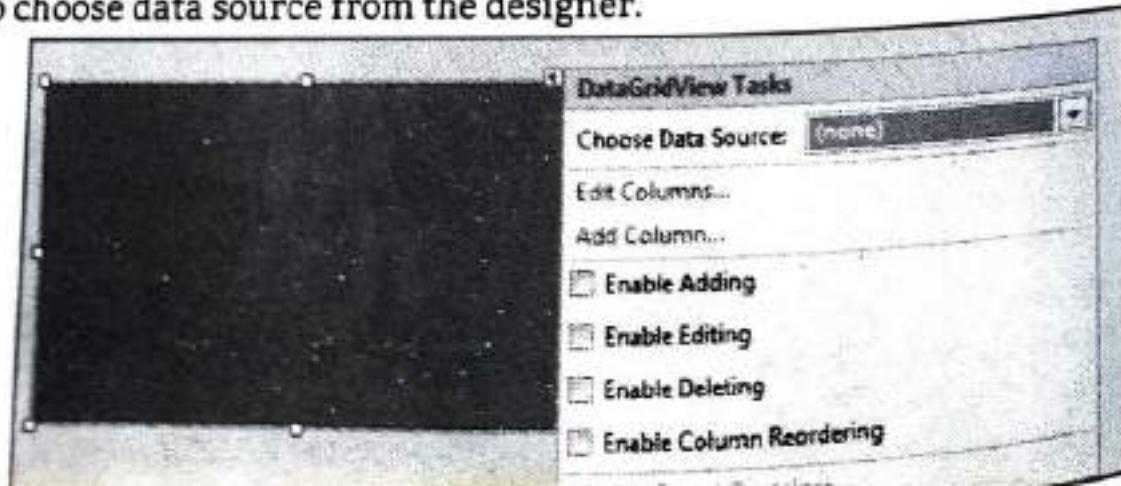
[S-2]

- DataGridView provides a visual interface to data, it is an excellent way to display and allow editing of users data.
- It is accessed with VB.NET code. Data edited in the DataGridView can then be persisted in the database.
- The DataGridView control is basically made of rows and columns. Additionally, every column has its header, where the header text can be changed. The header text can be changed in the designer. It can also be changed programmatically. Vertical and horizontal scrollbars appear automatically whenever they are needed. The scrollbars are used to scroll through the pages to see more content. So a DataGridView control may contain information of multiple pages.

**Choosing Data Source for the DataGridView control**

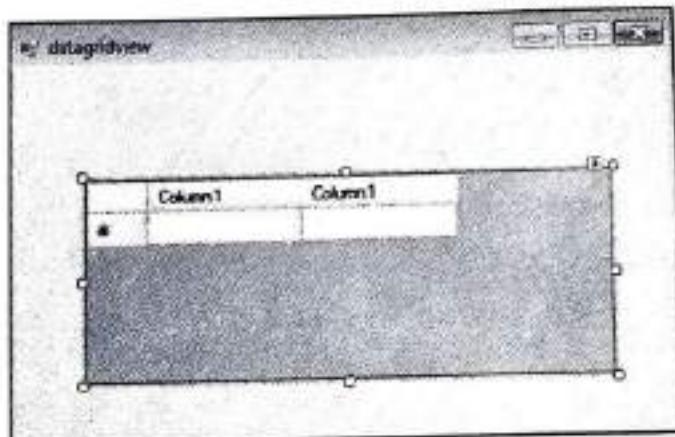
- After adding the DataGridView control on the form, the first thing that you need to do is to choose the data source, either from the designer or in the code, so that the data is automatically shown from the database.

Choosing data source from the designer: The following screenshot demonstrates how to choose data source from the designer.



- Choosing data source in the code:** The following code shows you how to add a data source.

```
private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
 DataGridView1.DataSource = db.data("select * from student", "stud")
End Sub
```



### Properties of DataGridView control:

#### 1. Selecting rows: the Selected property

- The Selected property is used in combination with the 'Rows' property to select a row, or to know whether the row is selected or not. This gives Boolean values.

```
For i = 0 To DataGridView1.Rows.Count - 1
 DataGridView1.Rows(i).Selected = True
Next
```

#### 2. Counting rows/columns: the Count property

- The 'Count' property is used in combination of 'Rows' and 'Columns' properties. It gives the number of the rows or columns.

```
Label1.Text = DataGridView1.Rows.Count
```

#### 3. Getting selected rows/columns: the SelectedRows/SelectedColumns property

- SelectedRows/SelectedColumns is a ReadOnly property that is used to get the rows/columns selected by the user.

#### 4. Changing the background color: the BackgroundColor property

- You can change the background color of the DataGridView control using the BackgroundColor property.

```
DataGridView1.BackgroundColor = Color.Aqua
```

#### 5. Row at a specified index: the Item property

- Item is a ReadOnly property that is used to get an item such as a row at a specified index. The index value 0 indicates the first item.

```
TextBoxName.Text = DataGridView1.SelectedRows.Item(0).Cells(0).Value
```

**Methods of DataGridView control:****1. Selecting cells: the SelectAll method**

- Use the SelectAll method to select all the cells in the DataGridView control.

```
DataGridView1.Rows.SelectAll()
```

**2. Adding rows/columns: the Add method**

- The Add method is used in combination with the Rows or Columns properties to add rows and columns. Consider the following example.

```
DataGridView1.Rows.Add("Car", "bike", "train", "taxi")
```

**• GetFirstRow**

The GetFirstRow method returns the index of the first row. This method is used in combination with the Rows property.

**• GetLastRow**

The GetLastRow method returns the index of the last row. This method is used in combination with the Rows property.

**• GetNextRow**

The GetNextRow method returns the index of the next row. This method is used in combination with the Rows property.

**• GetPreviousRow**

The GetPreviousRow method returns the index of the previous row. This method is used in combination with the Rows property.

**• GetRowCount**

The GetRowCount method returns the number of rows. This method is used in combination with the Rows property.

**2.3.2 Menus and PopUp Menu**

[S-23, W-22]

The main menu is implemented through the MainMenu class.

User can create the menu similar to windows menu items.

To create menu on the form, we have to drag MainMenu strip from toolbox and drop it on form.

The first top row is used to define Main menu items, below that you can define sub menu items, sub menu items.

We need to double click on menu item to write the code under specific menu item.

User can also define Menu Access Key, Shortcut Key to the respective menu items.

**Creating Menus and Submenus**

[W-22]

Simplest way to create menus in VB.NET is at design time because the MainMenu control is available in toolbox.

To create a new menu, double\_click text to open a text box where in that text box we can enter caption for menus and menu item.

To create submenu, select it in form designer, it opens a "Type Here" box to the right of that menu item, here we can enter the caption for submenu item.

- Selecting the first item in the submenu opens a "Type Here" box for the next item under it as well as another "Type Here" box for a new submenu to the right of it.
- 1. Creating Menu Access Key:** The access key makes it possible to select menu items from the keyboard using Alt Key. We precede the access key in its caption with an '&'.
  - 2. Creating Menu Shortcuts:** We can create shortcuts for menu items, which are key combination that when pressed, will select that item, making it click event happen. We select the Shortcut property of any menu item in the properties window.
  - 3. Creating Menu Separator:** It is a horizontal line that used to separate the menu item for one another, to create a menu separator we can assign a single hyphen (-) to a menu items Text Property.
  - 4. Showing and Hiding Menu Items:** To show and hide menu items, we can use their Visible property and set its value either True or False.

#### Event of the MenuItem:

- The default event of the MenuItem is the Click event.

#### Properties of MenuItem Class:

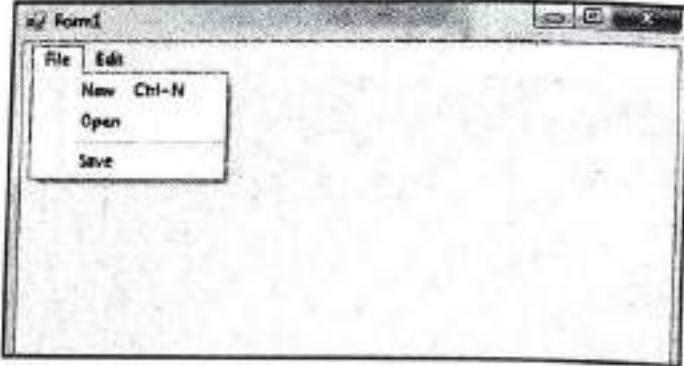
- Checked:** Default value is set to False, changing it to True makes a checkmark appear towards the left of the menu.
- DefaultItem:** Default value is set to false, changing it to True makes this menu item default menu item.
- RadioCheck:** Changing it to true makes a menu item display a radio button instead of a checkmark.
- Shortcut:** Enables to set a short cut key from a list of available shortcuts for the menu item.

**Example 17:** For Menu and submenu creation along with shortcut key, Access key and Menu separator:

```

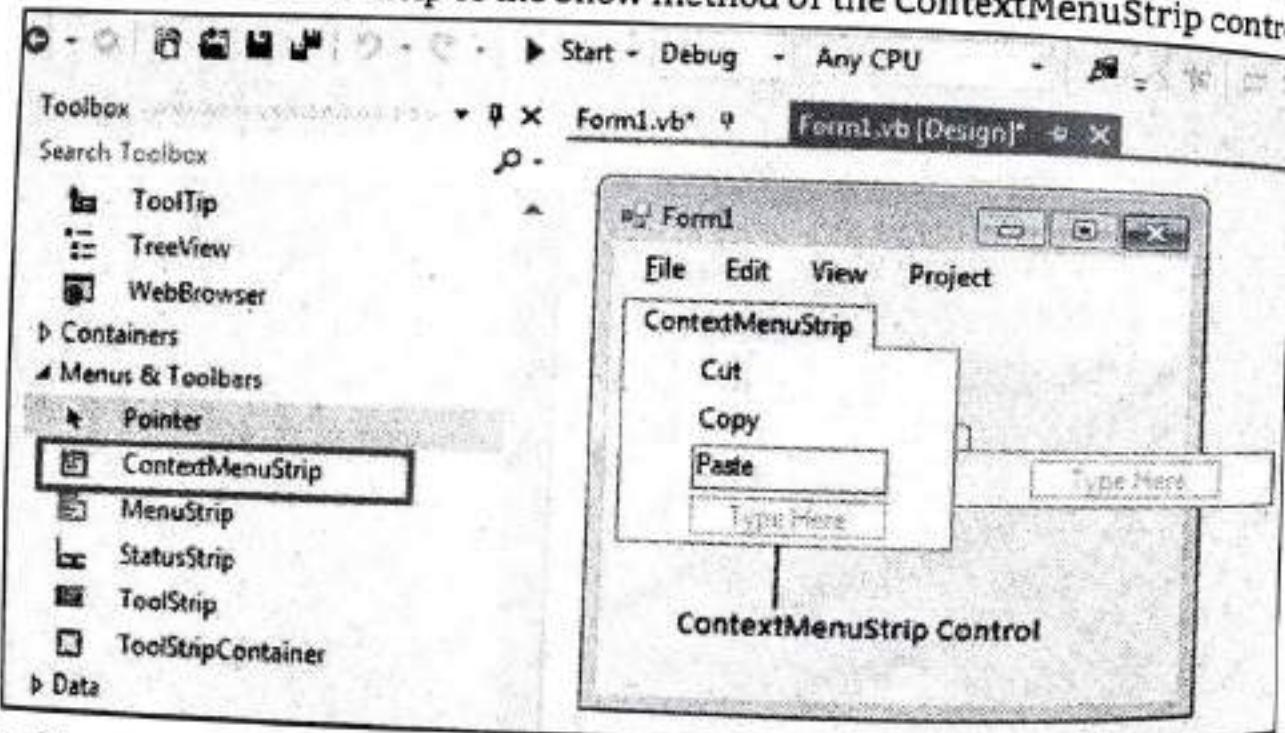
public Class Form1
 private Sub MenuStrip1_ItemClicked(ByVal sender As System.Object, ByVal
 e As System.Windows.Forms.ToolStripItemClickedEventArgs)
 Handles MenuStrip1.ItemClicked
 End Sub
 private Sub NewToolStripMenuItem_Click(ByVal sender As System.Object,
 ByVal e As System.EventArgs) Handles NewToolStripMenuItem.Click
 MessageBox.Show("You click on New Submenu item")
 End Sub
 private Sub OpenToolStripMenuItem_Click(ByVal sender As System.Object,
 ByVal e As System.EventArgs) Handles OpenToolStripMenuItem.Click
 MessageBox.Show("You click on Open Submenu item")
 End Sub
 private Sub SaveToolStripMenuItem_Click(ByVal sender As System.Object,
 ByVal e As System.EventArgs) Handles SaveToolStripMenuItem.Click
 MessageBox.Show("You click on Save Submenu item")
 End Sub
End Class

```



## PopUp Menu in VB.NET

- The **ContextMenuStrip** control represents a shortcut menu that pops up controls, usually when you right click them. They appear in context of some specific controls, so are called context menus. For example, Cut, Copy or Paste options.
- This control associates the context menu with other menu items by setting menu item's ContextMenuStrip property to the ContextMenuStrip control designed.
- Context menu items can also be disabled, hidden or deleted. You can also show context menu with the help of the Show method of the ContextMenuStrip control.



- In this example, let us add a content menu with the menu items Cut, Copy and Paste. Take the following steps:
  - Drag and drop or double click on a ContextMenuStrip control to add it to the form.
  - Add the menu items, Cut, Copy and Paste to it.
  - Add a TextBox control on the form.
  - Set the ContextMenuStrip property of the text box to ContextMenuStrip1 using the properties window.

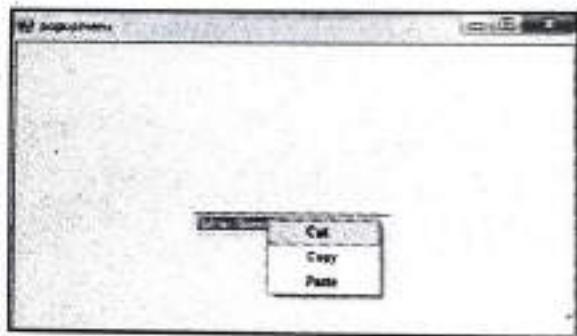
- o Double the menu items and add following codes in the Click event of these menus

```

public Class popupmenu
 private Sub CutToolStripMenuItem_Click(ByVal sender As System.Object,
 ByVal e As System.EventArgs) Handles CutToolStripMenuItem.Click
 TextBox1.Cut()
 End Sub
 private Sub CopyToolStripMenuItem_Click(ByVal sender As System.Object,
 ByVal e As System.EventArgs) Handles CopyToolStripMenuItem.Click
 TextBox1.Copy()
 End Sub
 private Sub PasteToolStripMenuItem_Click(ByVal sender As
 System.Object, ByVal e As System.EventArgs) Handles PasteToolStripMenuItem.Click
 TextBox1.Paste()
 End Sub
End Class

```

#### **Output:**



### **2.3.3 Predefined Dialog Controls**

#### **Color Dialog Control:**

- Color dialog provides facilities to select a color from the available color palette.
- The principal we use of these dialogs is the color property, which returns a color object ready for use.
- If we set the SolidColorOnly property to True, the user can select only solid colors.

#### **Open File Dialog:**

- The Open File let the user select a file to open, open file dialogs are supported with the OpenFileDialog class.
- Open File Dialog box lets the user select a file to open, open file dialogs are supported with the OpenFileDialog class.
- Open File Dialogs are supported with the OpenFileDialog class, we can let users select multiple files with the Multiselect property.
- User can use the ShowReadOnly property to determine if a read-only checkbox appears in the dialog box.
- The name and path of the user selected file is stored in the FileName property of the OpenFileDialog object.

**Save File Dialog:**

- Save File Dialog are supported by the SaveFileDialog class.
- Save File Dialog let the user specify the name of a file to save data.
- Save File Dialog is same as the standard Save File dialog box used by Windows.

**Font Dialog:**

- Font dialogs lets the user select a font size, color, face etc.
- To display the font dialog box, call the ShowDialog method.
- Font dialog shows list boxes for Font, Size and Style as well as checkboxes for effects like Strikeout and Underline, a drop-down list for script and a sample of how the font will appear.

**Print Dialog Box:**

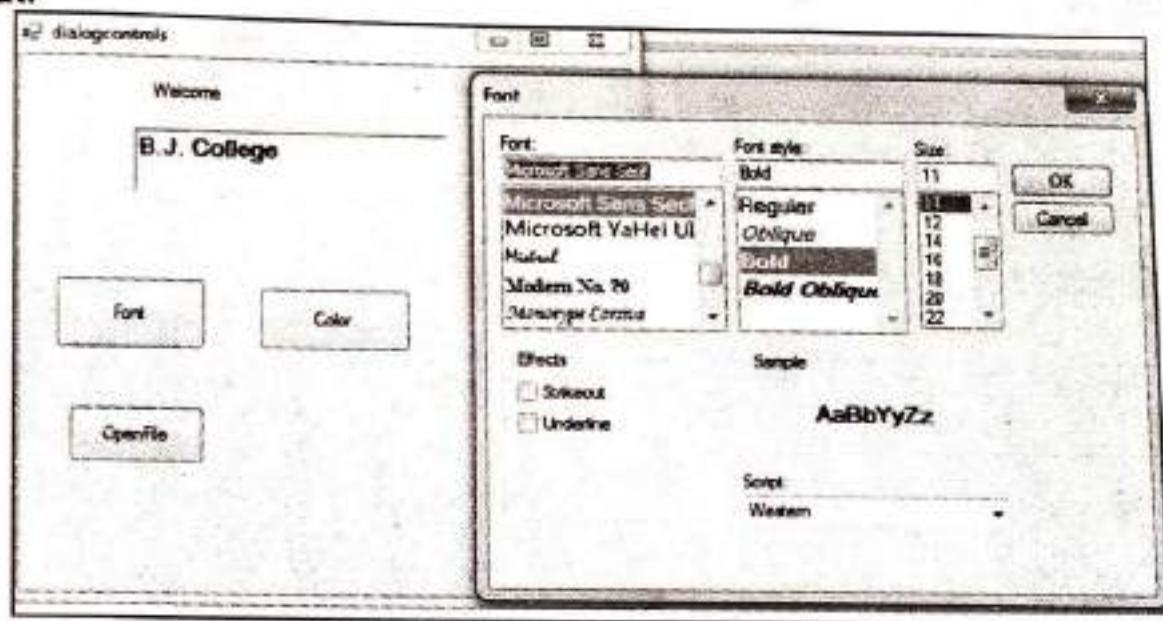
- Print dialogs allows user to print documents for which print dialogs are supplied with the PrintDialog class.
- We have to set the Document property of a PrintDialog object to a PrintDocument object and the PrinterSettings property to PrinterSettings objects of the kind of page setup dialog.

**Example 18: To demonstrate dialog boxes in your application:**

```

public Class dialogcontrols
 private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
 If FontDialog1.ShowDialog() <> Windows.Forms.DialogResult.Cancel Then
 RichTextBox1.Font = FontDialog1.Font
 End If
 End Sub
 private Sub Button2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button2.Click
 If ColorDialog1.ShowDialog() <> Windows.Forms.DialogResult.Cancel Then
 Label1.BackColor = ColorDialog1.Color
 End If
 End Sub
 private Sub Button3_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button3.Click
 If OpenFileDialog1.ShowDialog() <> Windows.Forms.DialogResult.Cancel Then
 PictureBox1.Image = Image.FromFile(OpenFileDialog1.FileName)
 End If
 End Sub

```

**Output:****2.3.4 DialogBox**

[S-22]

**InputBox()**

- InputBox is used to prompt the user to enter the values.
- A text can be return in the text box by using the InputBox function if the user clicks on the OK or Enter button. And if the user clicks on the Cancel button, then the InputBox function will return an empty string ("").

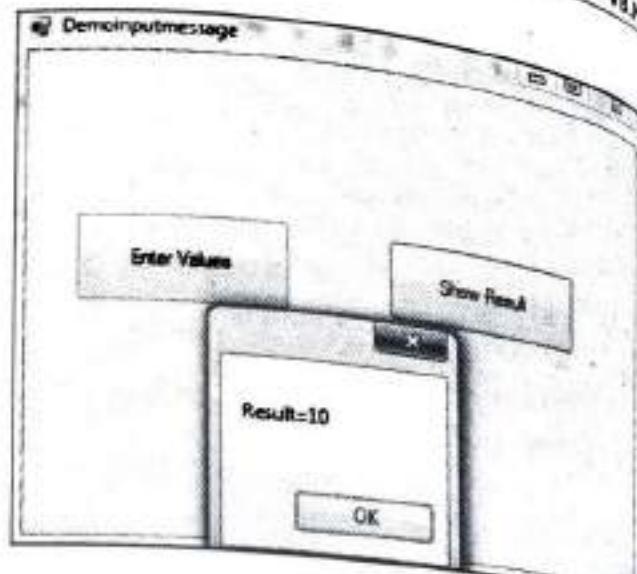
**MessageBox()**

- A Class called as MessageBox because it is introduced by VB.NET which encapsulates all the features of MsgBox.
- The difference between MessageBox and MsgBox is that MsgBox is a function while MessageBox is a Class.
- The MessageBox class has overloaded method show () to display message on prompt.

**Example 19: To demonstrate the use of InputBox and MessageBox in VB.Net application**

```
public Class Demoinputmessage
 Dim a, b, c As Integer
 private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
 System.EventArgs) Handles Button1.Click
 a = InputBox("Enter value for a")
 b = InputBox("Enter value for b")
 c = a + b
 End Sub
 private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
 System.EventArgs) Handles Button2.Click
 c = a + b
 MessageBox.Show("Result=& c")
 End Sub
End Class
```

## Output:



### MsgBox()

- The **MsgBox** function displays a message in a DialogBox, waits for the user to click a button and returns an integer indicating which button the user clicked.
- The **MsgBox function** displays a message box and waits for the user to click a button and then an action is performed based on the button clicked by the user.

**Syntax:** `MsgBox(prompt[,buttons][,title][,helpfile, context])`

### Parameter Description

- Prompt:** A Required Parameter. A String that is displayed as a message in the dialog box. The maximum length of prompt is approximately 1024 characters. If the message extends to more than a line, then the lines can be separated using a carriage return character (`Chr(13)`) or a linefeed character (`Chr(10)`) between each line.
- Buttons:** An Optional Parameter. A Numeric expression that specifies the type of buttons to display, the icon style to use, the identity of the default button, and the modality of the message box. If left blank, the default value for buttons is 0.
- Title:** An Optional Parameter. A String expression displayed in the title bar of the dialog box. If the title is left blank, the application name is placed in the title bar.
- Helpfile:** An Optional Parameter. A String expression that identifies the Help file to use for providing context-sensitive help for the dialog box.
- Context:** An Optional Parameter. A Numeric expression that identifies the Help context number assigned by the Help author to the appropriate Help topic. If context is provided, helpfile must also be provided.

### summary

Visual Basic .NET (VB.NET) is an object-oriented computer language that can be viewed as an evolution of Microsoft's Visual Basic (VB) implemented on the Microsoft .NET Framework.

VB.NET provides many built-in operators that allow us to manipulate data.

- Available operators are as bellow:
  1. Arithmetic Operators
  2. Comparison Operators
  3. Logical/Bitwise Operators
  4. Concatenation Operators
- A Process of converting a value of one datatype to another is called as conversion or casting.
- To enables decision making ability .NET supports various decision-making statements.
- In VB.NET working environment, user can develop forms visually, adding controls and other items from the toolbox.
- Docking and anchoring is used to make a control covers an entire client area of the form.
- Anchoring is used to resize controls dynamically with the form.
- The main menu is implemented through the `MainMenu` class.
- User can also define Menu Access Key, Shortcut Key to the respective menu items.
- Color dialog provides facilities to select a color from the available color palette.
- The principal we use of these dialogs is the `color` property, which returns a `color` object ready for use.

## Practice Questions

1. Enlist various operators in VB.NET.
2. Describes following loops in VB.NET
  - (i) For Loop
  - (ii) While Loop
3. What are the different data types used in VB.NET?
4. How to add control to the form?
5. List ant two Properties of `ListBox` Control.
6. Which property is used to find out whether the checkbox is on or off?
7. Name the different Dialog boxes which are used in VB.Net Application.
8. Explain Docking and Anchoring Controls.
9. Explain following functions with example:
  - (i) `MessageBox()`
  - (ii) `InputBox()`
10. Describe the term Windows form.
11. How to create Menus in VB.Net?
12. Explain various Built-in dialog boxes.
13. With suitable example describe tree view control.
14. Explain following controls with example:
  - (i) `Textbox`
  - (ii) `RadioButton`
  - (iii) `CheckBox`
  - (iv) `PictureBox`
15. State and explain various statements used in VB.NET.



# 3...

# Introduction to C#

## Learning Objectives ...

Students will be able to:

- Read, write, execute, and debug C# applications.
- Understand variables and data types.
- Code decision and control structures (if, if/else, switch, while, do/while, for) and use primitive data types.
- Write user-defined methods.
- Write and manipulate arrays.
- Write programs using object-oriented programming techniques including classes, objects, inheritance, and polymorphism.

### 3.1 LANGUAGE FUNDAMENTALS

- C# is an object-oriented programming language, which combines the power and efficiency of C++, simplicity and object-oriented approach of Java and creativeness of Visual Basic.
- Similar to Java, C# does not use multiple inheritances, moreover contrary to java, C# includes the useful operations, such as operator overloading, enumerations, pre-processor directives and function pointers.

#### Need of C#:

- C# is one of the intermediate programming languages used to create executable program. It is an object-oriented programming language having many similarities to java, C++, Visual Basic.
- C# has various other features, such as Language-Integrated Query (LINQ), delegates and lambda expressions.

#### C# becomes the integral part of .NET:

1. **Flexible:** C# program has the capability of getting executed on the local system or any remote system on web.
2. **Powerful:** An ability of C# to create any type of applications, such as word processors, spreadsheets and even compilers for some languages.

3. **Easy to use:** Refers to the use of simpler code statements instead of using complex data types.
4. **Visually oriented:** Allows user to use the .NET code libraries for developing attractive User Interface.
5. **Secure:** Enables user to protect code and data from unwanted access and damage by using the different techniques and class libraries of .NET Framework.

### C# Variables and Constants:

- A variable is an identifier that denotes a storage location in memory. It stores numeric or string values that might change during program execution.
- A variable may take different values at different times during the execution of a program but the name of the variable always remains same.
- Every variable has a data type that determines the type of values you can store in a variable.

#### **Example:**

```
int i;
i = 15;
```

- Variables can be of array types sometimes.  
i.e. int [ ] arr = new int[10];
- Variable can also be defined as user-defined types i.e., enumeration
- Similar to a variable, a constant is also used to store a value. However, unlike variable, the value of a constant does not change during the execution of a program.
- A variable is declared as constant using the const keyword.

#### **Example:**

[S-23, W-22]

```
const int x=100;
```

- The constants are mostly used while performing the mathematical calculations where the value of pi, square roots is pre-calculated.

#### **Example:**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace Constant
{
 class program
 {
 static void Main (string [] args)
 {
 const double pi=3.1472; //Declaring constant variable
```

```

int r;
Console.WriteLine("Enter the value of r");
r = int.Parse(Console.ReadLine());
double circleArea = pi * r * r;
Console.WriteLine("The area of circle is:" + circleArea);
Console.ReadLine();
}
}
}

```

**Expressions and Operators:**

- An Expression is a set of language elements arranged together to perform a specific task or computations.
- Operators play a vital role in performing some computation or other operations such as arithmetical or logical operations on operands.

**Different categories of operators are as bellowed:**

| Category | Operators | Description                                                                                        |
|----------|-----------|----------------------------------------------------------------------------------------------------|
| Primary  | x.y       | Accesses the member variable and methods of a class.                                               |
|          | x++       | Increases the value of its operand x by 1.                                                         |
|          | x--       | Decreases the value of its operands x by 1.                                                        |
|          | new       | Creates objects and call constructors.                                                             |
|          | typeof    | Discovers information about reference and value types.                                             |
| Unary    | +         | Gets the logical complement of the negative value.                                                 |
|          | -         | Gets the negation of the operand value.                                                            |
|          | !         | Inverts the result of a Boolean expression.                                                        |
|          | ~         | Inverts the binary representation of an expression.                                                |
|          | true      | Returns the Boolean value true, indicating an expression satisfies the specified condition.        |
|          | false     | Returns the Boolean value true, indicating an expression does not satisfy the specified condition. |
|          | &         | Returns the address of its operand.                                                                |
|          | sizeof    | Returns the size of value types in bytes.                                                          |

contd..

|                             |    |                                                                                                                                     |
|-----------------------------|----|-------------------------------------------------------------------------------------------------------------------------------------|
| Multiplicative and Additive | *  | Computes the product of its operands.                                                                                               |
|                             | /  | Computes the division of its operands by dividing the first operand by second operand.                                              |
|                             | %  | Computes the remainder of its operands after dividing the first operand by the second.                                              |
|                             | +  | Computes the sum of its operands.                                                                                                   |
|                             | -  | Computes the difference of its operands.                                                                                            |
| Shift                       | << | Shifts its first operand left by a number of bits specified by the second operand.                                                  |
|                             | >> | Shifts its first operand right by a number of bits specified by the second operand.                                                 |
| Relational and type testing | <  | Finds out whether the first operand is less than the second operand.                                                                |
|                             | >  | Finds out whether the first operand is larger than the second operand.                                                              |
|                             | <= | Finds out whether the first operand is less than or equal to the second operand.                                                    |
|                             | >= | Finds out whether the first operand is larger than or equal to the second operand.                                                  |
|                             | is | Checks the compatibility of a variable with a given type.                                                                           |
|                             | as | Converts a value to a given reference type.                                                                                         |
| Equality                    | == | Checks whether the two expressions are same or not. It returns true if the expressions are equal; otherwise, it returns false.      |
|                             | != | Checks whether the two expressions are same or not. It returns true, if the expressions are not equal; otherwise, it returns false. |

- Besides this there are Logical AND, Logical XOR, Logical OR, conditional AND, conditional OR, Assignment and Scope resolution operators are also there in C#.

### 3.1.1 Data Types and Control Constructs

[W-22]

- C# supports a rich and varied selection of data types, from built-in types, such as integers or strings, to user-defined types, such as enumerations, structures and classes.

- When a variable is declared, a data type is assigned to the variable, compiler must know what kind of data a specific variable is expected to store.
- C# mainly categorized data types in two types:
  - Value Types:** Value types include simple types such as int, float, bool and char also include Enum types, struct types and Nullable value.
  - Reference Types:** Reference types include class types, interface types, delegate types and array types.

**Control Construct in C#:****C# offers three types of control statements:****[S-23]**

- Selection Statements:** This consists of if, else, switch.
- Iteration Statements:** This consists of do, for, while looping.
- Jump Statements:** This consists of break, continue, return, and goto statements.

**1. if statement:**

- An if statement consists of a boolean expression followed by one or more statements.
- The following is the syntax:

```
if(boolean_expression)
{
 /* statement(s) will execute if the boolean expression is true */
}
```

**2. if-else statement:**

- An if statement can be followed by an optional else statement, which executes when the boolean expression is false.
- The following is the syntax:

```
if(boolean_expression)
{
 /* statement(s) will execute if the boolean expression is true */
}
else
{
 /* statement(s) will execute if the boolean expression is false */
}
```

**Example:**

```
using System;
namespace Studytonight
{
 public class Program
 {
```

```
public static void Main(string[] args)
{
 int i=100;
 if (i>0)
 {
 Console.WriteLine("Given number is positive!");
 }
 else
 {
 Console.WriteLine("Given number is negative!");
 }
 Console.ReadKey();
}
```

**Output:**

Given number is positive!

**3. Switch statement:**

[S-22, W-22]

- The switch statement executes only one statement from multiple given statements associated with conditions. If any condition returns true, then the code statements below it gets executed. Following diagram shows the basic flow of a switch statement.

```
using System;
namespace Studytonight
{
 public class Program
 {
 public static void Main(string[] args)
 {
 Console.WriteLine("1.Addition\n2.Subtraction\n3.Division\nEnter
your choice:");
 int c = Convert.ToInt32(Console.ReadLine());
 switch(c)
 {
 case 1: Console.WriteLine("\nAddition selected!");break;
 case 2: Console.WriteLine("\nSubtraction selected!");break;
 case 3: Console.WriteLine("\nDivision selected!");break;
 }
 }
 }
}
```

```
case2: Console.WriteLine("\nSubtraction selected!");break;
case3: Console.WriteLine("\nDivision selected!");break;
default: Console.WriteLine("\nYour choice not found!");break;
}
Console.ReadKey();
}
}
```

**Output:**

1. Addition
2. Subtraction
3. Division

```
Enter your choice: 4
Your choice not found!
```

**for loop:**

It executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.

The following is the syntax:

```
for (init; condition; increment)
{
 statement(s);
}
```

**Example:**

```
using System;
namespace Loop
{
 class ForLoop
 {
 public static void Main(string[] args)
 {
 for (int i=1; i<=5; i++)
 {
 Console.WriteLine("C# For Loop: Iteration {0}", i);
 }
 }
 }
}
```

**Output:**

When we run the program, the output will be:

C# For Loop: Iteration 1  
 C# For Loop: Iteration 2  
 C# For Loop: Iteration 3  
 C# For Loop: Iteration 4  
 C# For Loop: Iteration 5

**5. while loop:**

[S-22]

- It repeats a statement or a group of statements while a given condition is true. It tests the condition before executing the loop body.
- The following is the syntax:

```
while(condition)
{
 statement(s);
}
```

**Example:**

```
using System;
namespace Loop
{
 class WhileLoop
 {
 public static void Main(string[] args)
 {
 int i=1;
 while (i<=5)
 {
 Console.WriteLine("C# For Loop: Iteration {0}", i);
 i++;
 }
 }
 }
}
```

**Output:**

When we run the program, the output will be:

C# For Loop: Iteration 1  
 C# For Loop: Iteration 2  
 C# For Loop: Iteration 3  
 C# For Loop: Iteration 4  
 C# For Loop: Iteration 5

## **Do...while loop:**

It is similar to a while statement, except that it tests the condition at the end of the loop body.

The following is the syntax:

```
do
{
 statement(s);
}while(condition);
```

### **Example:**

```
using System;
namespace Loop
{
 class DoWhileLoop
 {
 public static void Main(string[] args)
 {
 int i = 1, n = 5, product;
 do
 {
 product = n * i;
 Console.WriteLine("{0} * {1} = {2}", n, i, product);
 i++;
 } while (i<= 10);
 }
 }
}
```

### **Output:**

When we run the program, the output will be:

```
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
5 * 10 = 50
```

### 7. Break Statement:

- The break statement is used to terminate the loop or statement in which it present. After that, the control will pass to the statements that present after the break statement, if available. If the break statement present in the nested loop, then it terminates only those loops which contains break statement.

C# program to illustrate the

```
// use of break statement
using System;
class Geeks
{
 // Main Method
 static public void Main()
 {
 // GeeksforGeeks is printed only 2 times
 // because of break statement
 for (int i = 1; i < 4; i++)
 {
 if (i == 3)
 break;
 Console.WriteLine("GeeksforGeeks");
 }
 }
}
```

#### Output:

GeeksforGeeks

GeeksforGeeks

### 8. Continue statement:

- This statement is used to skip over the execution part of the loop on a certain condition. After that, it transfers the control to the beginning of the loop. Basically, it skips its following statements and continues with the next iteration of the loop.

C# program to illustrate the

```
// use of continue statement
using System;
class Geeks
{
 // Main Method
```

```
public static void Main()
{
 // This will skip 4 to print
 for (int i = 1; i<= 10; i++)
 {
 // if the value of i becomes 4 then
 // it will skip 4 and send the
 // transfer to the for loop and
 // continue with 5
 if (i == 4)
 continue;
 Console.WriteLine(i);
 }
}
```

#### **Output:**

```
1
2
3
5
6
7
8
9
10
```

#### **9. goto statement:**

- The goto statement is used to transfer the control from a loop or switch case to a specified label. It is also known as a **jump statement**. It is generally recommended to avoid using it as it makes the program more complex. In the following example, if the user is logged in then we have directly jumped our code to **success** label.

```
using System;
namespace Studytonight
{
 public class Program
 {
```

```

public static void Main(string[] args)
{
 string user = "login";
 if (user == "login")
 {
 goto success;
 }
 success:
 Console.WriteLine("Welcome to Studytonight!");
 Console.ReadKey();
}
}
}

```

- This statement is used to transfer control to the labelled statement in the program. The label is the valid identifier and placed just before the statement from where the control is transferred.

```

C# program to illustrate the
// use of goto statement
using System;
class Geeks
{
 // Main Method
 static public void Main()
 {
 int number = 20;
 switch(number)
 {
 case5:
 Console.WriteLine("case5");
 break;
 case10:
 Console.WriteLine("case10");
 break;
 case20:
 Console.WriteLine("case20");
 // goto statement transfer
 // the control to case 5
 goto case5;
 }
 }
}

```

```

 default:
 Console.WriteLine("No match found");
 break;
 }
}
}

```

**Output:**

```

case 20
case 5

```

**10. return statement:**

- This statement terminates the execution of the method and returns the control to the calling method. It returns an optional value. If the type of method is void, then the return statement can be excluded.

C# program to illustrate the

```

// use of return statement
using System;
class Geeks
{
 // creating simple addition function
 static int Addition(int a)
 {
 // add two value and
 // return the result of addition
 int add = a + a;
 // using return statement
 return add;
 }
 // Main Method
 static public void Main()
 {
 int number = 2;
 // calling addition function
 int result = Addition(number);
 Console.WriteLine("The addition is{0}", result);
 }
}

```

**Output:**

```
The addition is 4
```

### 3.1.2 Value Type and Reference Types, Boxing

[W-22]

#### (A) Value Types:

- Value types allow us to store the data directly into the variable.
- They are derived from System.ValueType and Common Language Runtime (CLR).
- The Default values of value types are stored on stack.
- Available value types in C#.

| Type     | Description                                 | Range                                                   |
|----------|---------------------------------------------|---------------------------------------------------------|
| byte     | 8-bit unsigned integer                      | 0 to 255                                                |
| sbyte    | 8-bit signed integer                        | -128 to 127                                             |
| short    | 16-bit signed integer                       | -32,768 to 32,767                                       |
| ushort   | 16-bit unsigned integer                     | 0 to 65,535                                             |
| char     | 16-bit single Unicode character             | Any                                                     |
| int      | 32-bit signed integer                       | -2,147,483,648 to 2,147,483,647                         |
| float    | 32-bit Single-precision floating point type | -3.402823e38 to 3.402823e38                             |
| long     | 64-bit signed integer                       | -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 |
| double   | 64-bit double-precision floating point type | -1.79769313486232e308 to 1.797693134862232e308          |
| string   | A sequence of Unicode characters            |                                                         |
| object   | Base type of all other types                |                                                         |
| bool     | 8-bit logical true/false value              | True or False                                           |
| DateTime | Represents date and time                    | 00:00:00 am 1/1/01 to 11:59:59 pm 12/31/9999            |

- As we can see in the above table that each data type (except string and object) includes value range.
- The compiler will give an error if the value goes out of datatype's permitted range.
- The Value of unsigned integers, long, float, doubles and decimal type must be suffixed by u, l, f, d and m respectively.
- The value Type are divided into the following categories:
  - Struct type
  - Enumeration type

**1. Struct Type:**

- Struct types are a special form of classes having the properties of value types. As you know that the value types are stored on the stack; inherently the struct type are also stored on the stack.
- The struct types can be copied and created efficiently, as stacks are the efficient and convenient means of storing and accessing the types.
- The struct types encapsulate small group of related variables, they can contain constructors, methods, properties, operators, events, nested types, indexes, constants etc.
- While creating the struct type, you should remember that struct members cannot be declared as protected as they fail to support inheritance.
- User can create struct type by using the struct keyword, when user create a struct object and assign it to a variable, the variable holds the value of the struct object.
- We can also use struct type as a nullable type by assigning a null value to it.

**For example,**

```
Public struct student
{
 string name;
 int rollno;
 int classname; //fields
 int section;
 //methods
 //properties
}
```

**The struct types are divided into the following five categories:**

1. Integral type
2. Floating-Point type
3. Decimal type
4. Boolean type
5. Nullable type

**Enumeration Type:**

Enumerations are user-defined integer data types that are declared using the enum keyword.

Using enumeration, you can define a set of named integral constants that can be assigned to a variable.

Enumerations are strongly typed constants, which allow you to assign symbolic names to integral values.

**For example:** Showing the Code for the Enumeration Application.

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace Enumeration
{
 public enum color {Red=1, Green, Yellow}
 class Program
 {
 static void Main(string[] args)
 {
 Console.WriteLine("Select 1 for Red, 2 for Green, 3 for Yellow:");
 string str=Console.ReadLine();
 int colInt=Int32.Parse(str);
 color col=(color)colInt;
 switch(col)
 {
 case color.Red:
 Console.WriteLine("The selected color is Red");
 break;
 case color.Green:
 Console.WriteLine("The selected color is Green");
 break;
 case color.Yellow:
 Console.WriteLine("The selected color is Yellow");
 break;
 default:
 Console.WriteLine("Invalid Selection");
 break;
 }
 Console.ReadLine();
 }
 }
}

```

#### (B) Reference Type:

- C# provides types that are passed by references to calling functions called as reference types.
- Variables that refer to the objects store the reference of the actual data.
- The reference types use heap to store the references instead of actual data.

In C#, heap is called managed heap, as it manages and maintains a pointer to the address where the next object is to be allocated.

Reference type can be categorized into two parts: pre-defined and user-defined types.

Pre-defined Reference types:

C# provides some built-in reference types, which are as follows:

1. Dynamics Type
2. Object Type
3. String Type

**1. Dynamic Type:** The Dynamic Type performs the type checking of the dynamic type variable at run time instead of compile time.

Using the dynamic keyword, one can declare a dynamic type variable.

**Example:** dynamic dyn=10;

**2. Object Type:** The object type enables user to assign value of any type to the variable of object type.

Object is an alias for the predefined System.Object class, type checking of the variable of object type is performed at compile time.

The conversion of value type to the object type and vice versa takes place known as Boxing and UnBoxing.

**Example:** Object x;

x=15; //boxing

**3. String Type:** String type enables you to design string values to the variable of string type. The string type is an alias for the System.String class.

It is an unchangeable sequence of Unicode characters contained within double quotes.

Then string type is a sealed class type that is directly derived from the object type.

**Example:** Showing the code to declare and initialize String.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text
namespace Strings
{
 Class Program
 {
 Static void Main(string[] args)
```

```

 {
 String str;
 Str="Welcome to C#";
 console.WriteLine("The message is" + str);
 console.ReadLine();
 }
}
}

```

**User-defined Reference Types:**

- User-defined reference types are defined by using pre-defined types.
- User-defined reference types can be divided into four parts as mentioned below:
  1. Classes
  2. Interfaces
  3. Delegates
  4. Arrays

**[W-22]****Boxing and Unboxing:**

- Boxing and Unboxing are important concepts used in the C# type systems.
- They are used to create a link between the two major data types in C# i.e., value types and reference type.
- All value types are stored in stack, but in some situations, they need to be referenced as heap.
- As reference types are stored in heap; therefore, you can convert the value type into reference type. This conversion is called as Boxing.
- Boxing is required in situations when a value type is converted into a base object or an interface. The Common Language Runtime (CLR) converts the value type to reference type.
- CLR allocates memory on heap and then copies the value type instance to it.
- Unboxing is the process of converting an instance of object type or interface back to value type. This is done explicitly by using unboxing.

**For Example:**

```

int y=123; //value type
object obj=y; //boxing is implicit
int z;
z=(int)obj; //Unboxing is explicit (use of type cast to unbox)

```

**3.1.3 Arrays****[S-22]**

- An array is used to store the number of elements or variables of the same data type at contiguous memory location in an ordered manner.
- In C#, the array is an actual type, much like integers. Arrays must be allocated on the heap.

- For example, an array is declared as:  
`int[ ] x = new int [10];`
- C# also supports multi-dimensional arrays, which must be allocated on the heap as well.
- Unlike C++ or Java, C# uses the old-style FORTRAN syntax for declaring multiple dimensions (by having comma-delimited dimensions in an array definition):  
`int[ , ] X = new int[5, 5];`
- An array is much more than a simple modifier, as it would be in C++ or Java. Because arrays are first class objects, just as all other types in are in C#, you can also inspect them for certain attributes.

**[S-23]****Properties of an Array:**

- Length Property:** User can access the length of an array which specifies the number of elements that can be stored in the array.  
**Example:** `int len = X.Length;`
- GetLowerBound():** User can find out lower boundary of an Array, returns an inter bound value.  
**Example:** `X.GetLowerBound();`
- GetUpperBound():** User can find out upper boundary of an Array, returns an integer bound value.  
**Example:** `X.GetUpperBound ( );`

**Example:** Showing below code for SingleDimensionArray Application

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace SingleDimensionArray
{
 class Program
 {
 static void Main (string [] args)string [2];
 {
 String[] name = new string
 int[] sub1 = new int[2];
 int[] sub2 = new int[2];
 int[] total = new int[2];
 for (int i=0;i<=1;i++)
 {
 Console.WriteLine("Enter the "+(i+1) +" student name");
 name[i]= Console.ReadLine();
 Console.WriteLine("Enter the marks of first subject");
 sub1[i]=int.Parse(Console.ReadLine());
 Console.WriteLine("Enter the marks of second subject");
 }
 }
 }
}
```

```
Sub2[i]=int.Parse(Console.ReadLine());
total[i]= sub1[i]+sub2[i];
}
for (int i=0;i<=1;i++)
{
 Console.WriteLine("The student name is" + name[i] + "and
 total marks =" +total[i]);
 Console.ReadLine();
}
}
```

**Example:** Showing the code for MultiDimensionArray Application

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace MultiDimensionArray
{
 class Program
 {
 static void Main (string [] args)string [2];
 {
 //Declaring multidimensional array
 int [,] X = new int [3,3];
 for (int j=0;j<=2;j++)
 {
 for (int k=0;k<=2;k++)
 {
 Console.WriteLine("Enter the Element");
 X[j,k] = int.Parse(Console.ReadLine());
 }
 }
 for (int j=0;j<=2;j++)
 {
 Console.WriteLine();
 for(int k=0;k<=2;k++)
 {
 Console.Write(X[j,k] + "\t");
 }
 }
 Console.ReadLine();
 }
 }
}
```

### 3.1.4 String class and its various operations-

- In C# programming, string is another kind of data type that represents Unicode Characters. It is the alias of System.String, however, you can also write System.String instead of a string. It is the sequence of character in which each character is a Unicode character.
- There is no difference between string and String because a string is the alias of System.String. Most of the developers get confused what to use between sting and String. Technically there is no difference between them and they can use any of them. However, you will have to use "using System" to use the String in C#. Another difference is String is a class name whereas a string is a reserved keyword. You should always use string instead of String.

#### C# string function:

[S-23]

| String Functions | Definitions                                                                                               |
|------------------|-----------------------------------------------------------------------------------------------------------|
| Clone()          | Make clone of string.                                                                                     |
| CompareTo()      | Compare two strings and returns integer value as output. It returns 0 for true and 1 for false.           |
| Contains()       | The C# Contains method checks whether specified character or string is exists or not in the string value. |
| EndsWith()       | This EndsWith Method checks whether specified character is the last character of string or not.           |
| Equals()         | The Equals Method in C# compares two string and returns Boolean value as output.                          |
| GetHashCode()    | This method returns HashValue of specified string.                                                        |
| GetType()        | It returns the System.Type of current instance.                                                           |
| GetTypeCode()    | It returns the System.TypeCode for class System.String.                                                   |
| IndexOf()        | Returns the index position of first occurrence of specified character.                                    |
| ToLower()        | Converts String into lower case based on rules of the current culture.                                    |
| ToUpper()        | Converts String into Upper case based on rules of the current culture.                                    |
| Insert()         | Insert the string or character in the string at the specified position.                                   |

contd..

|                |                                                                                    |
|----------------|------------------------------------------------------------------------------------|
| IsNormalized() | This method checks whether this string is in Unicode normalization form C.         |
| LastIndexOf()  | Returns the index position of last occurrence of specified character.              |
| Length         | It is a string property that returns length of string.                             |
| Remove()       | This method deletes all the characters from beginning to specified index position. |
| Replace()      | This method replaces the character.                                                |
| Split()        | This method splits the string based on specified value.                            |
| StartsWith()   | It checks whether the first character of string is same as specified character.    |
| Substring()    | This method returns substring.                                                     |
| ToCharArray()  | Converts string into char array.                                                   |
| Trim()         | It removes extra white spaces from beginning and ending of string.                 |

**Program 1: C# String Function:**

[S-23]

```

using System.Text;
namespace string_function
{
 class Program
 {
 static void Main(string[] args)
 {
 string firstname;
 string lastname;
 firstname = "Steven Clark";
 lastname = "Clark";
 Console.WriteLine(firstname.Clone());
 Console.WriteLine(firstname.CompareTo(lastname));
 Console.WriteLine(firstname.Contains("ven"));
 //Check whether specified value exists or not in string
 Console.WriteLine(firstname.EndsWith("n"));
 //Check whether specified value is the last character of string
 Console.WriteLine(firstname.Equals(lastname));
 Console.WriteLine(firstname.GetHashCode());
 Console.WriteLine(firstname.GetType());
 Console.WriteLine(firstname.GetTypeCode());
 }
 }
}

```

```

 Console.WriteLine(firstname.IndexOf("e"));
 //Returns the first index position of specified value
 //the first index position of specified value
 Console.WriteLine(firstname.ToLower());
 Console.WriteLine(firstname.ToUpper());
 Console.WriteLine(firstname.Insert(0, "Hello"));
 //Insert substring into string
 Console.WriteLine(firstname.IsNormalized());
 Console.WriteLine(firstname.LastIndexOf("e"));
 //Returns the last index position of specified value
 Console.WriteLine(firstname.Length);
 Console.WriteLine(firstname.Remove(5));
 Console.WriteLine(firstname.Replace('e', 'i'));
 // Replace the character
 string[] split = firstname.Split(new char[] { 'e' });
 //Split the string based on specified value
 Console.WriteLine(split[0]);
 Console.WriteLine(split[1]);
 Console.WriteLine(split[2]);
 Console.WriteLine(firstname.StartsWith("S"));
 //Check whether first character of string is same as specified value
 Console.WriteLine(firstname.Substring(2,5));
 Console.WriteLine(firstname.ToCharArray());
 Console.WriteLine(firstname.Trim());
 }
}
}

```

- **String.Format:** Converts the value according the Format specified by you. In this tutorial, you will learn all techniques to convert every type of value in the desired format.

```

using System;
namespace StringFormat_Example
{
 public class Program
 {
 public static void Main(string[] args)
 {
 decimal coin=17.36m;
 Console.WriteLine(String.Format("{0:C}",coin));
 }
 }
}

```

**Output:**

17.36

**String.Format for Currency Conversion**

- Most of the time you need to display integer, double or number value into currency. You don't need to write dozens line of code for it. Just let String.Format in action and customize value for it.

| Format | Description                                                        |
|--------|--------------------------------------------------------------------|
| {0:C}  | It displays the actual value with currency                         |
| {0:C4} | It displays actual value along with 4 digit after point precision. |
| {0:C6} | It displays actual value along with 6 digit after point precision. |

**Program 2:**

```
using System;
namespace StringFormat_Currency
{
 public class Program
 {
 public static void Main(string[] args)
 {
 decimal dcoin=2398.87m;
 int icoine=2309;
 float fcoin=9283.65f;
 Console.WriteLine(String.Format("Convert Decimal {0:C}",dcoin));
 Console.WriteLine(String.Format("Convert Decimal {0:C4}", icoine));
 Console.WriteLine(String.Format("Convert Decimal {0:C6}", fcoin));
 }
 }
}
```

**Output:**

Convert Decimal 2,398.87  
 Convert Decimal 2,309.0000  
 Convert Decimal 9,283.650000

**String.Format for DateTime Conversion**

- This is the area where String.Format is widely used. A simple DateTime program doesn't give formatted output. String.Format in C# helps you to display DateTime according to your need. You can apply following formation on DateTime.

| Format         | Description                                                            |
|----------------|------------------------------------------------------------------------|
| {0:yy}         | Displays Year in 2 Digits.                                             |
| {0:yyyy}       | Displays Year in 4 Digits.                                             |
| {0:MM}         | Displays Month. Use Capital 'M' for month because small m for minutes. |
| {0:dd}         | Displays Day Date.                                                     |
| {0:dd-MM-yyyy} | Displays date in dd-MM-YYYY format.                                    |
| {0:hh:mm:ss}   | Displays Time in hh:mm:ss format.                                      |
| {0:zz}         | Displays Current TimeZone.                                             |

**Program 3:**

```

using System;
namespace StringFormat_DateTime
{
 public class Program
 {
 public static void Main(string[] args)
 {
 DateTime dt=DateTime.Now;
 //Start Formation
 Console.WriteLine(String.Format("Original Value :- {0}",dt));
 Console.WriteLine(String.Format("Display Year in 2 Digits :- {0:yy}", dt));
 Console.WriteLine(String.Format("Display Year in 4 Digits :- {0:yyyy}", dt));
 //Use Capital M for month. Small m displays minutes
 Console.WriteLine(String.Format("Display Year and Months :- {0:yyyy MM}",dt));
 Console.WriteLine(String.Format("Display date, month and Year :- {0:dd MM yyyy}",dt));
 Console.WriteLine(String.Format("Display Time :- {0:hh:mm:ss}", dt));
 Console.WriteLine(String.Format("My Time Zone is :- {0:zz}", dt));
 }
 }
}

```

**Output:**

Original Value :- 17/02/17 9:56:00 AM  
 Display Year in 2 Digits :- 17  
 Display Year in 4 Digits :- 2017  
 Display Year and Months :- 2017 02  
 Display date, month and Year :- 17 02 2017  
 Display Time :- 09:56:00  
 My Time Zone is :- +05

**DateTimeFormatInfo**

- There are some defined standards in `DateTimeFormatInfo` that will help you to customize Date Time in the easiest way. Here is the list of Defined Standard for en-us culture.

| Specifier | Property                         | Pattern (en-us culture)                    |
|-----------|----------------------------------|--------------------------------------------|
| t         | ShortTimePattern                 | h:mm tt                                    |
| d         | ShortDatePattern                 | M/d/yyyy                                   |
| T         | LongTimePattern                  | h:mm:ss tt                                 |
| D         | LongDatePattern                  | ddd, MMMM dd, yyyy                         |
| f         | (combination of D and t)         | ddd, MMMM dd, yyyy h:mm tt                 |
| F         | FullDateTimePattern              | ddd, MMMM dd, yyyy h:mm:ss tt              |
| g         | (combination of d and t)         | M/d/yyyy h:mm tt                           |
| G         | (combination of d and T)         | M/d/yyyy h:mm:ss tt                        |
| m,M       | MonthDayPattern                  | MMMM dd                                    |
| y,Y       | YearMonthPattern                 | MMMM, yyyy                                 |
| r,R       | RFC1123Pattern                   | ddd, dd MMM yyyy HH':'mm':'ss<br>'GMT' (*) |
| s         | SortableDateTimePattern          | yyyy'-'MM'-'dd'T'HH':'mm':'ss (*)          |
| u         | UniversalSortableDateTimePattern | yyyy'-'MM'-'dd HH':'mm':'ss'Z' (*)         |

**Program 4:**

```
using System;
namespace StringFormat_DateTimeFormatInfo
{
 public class Program
 {
```

```

 public static void Main(string[] args)
 {
 DateTime dt = DateTime.Now;
 //Start Formation
 Console.WriteLine(String.Format("Short Time Pattern [t] : {0:t}", dt));
 Console.WriteLine(String.Format("Short Date Pattern [d] : {0:d}", dt));
 Console.WriteLine(String.Format("Long Time Pattern [T] : {0:T}", dt));
 Console.WriteLine(String.Format("Long Date Pattern [D] : {0:D}", dt));
 Console.WriteLine(String.Format("Combination of D and T [f] : {0:f}", dt));
 Console.WriteLine(String.Format("Full Date Time Pattern [F] : {0:F}", dt));
 Console.WriteLine(String.Format("Combination of d and t [g] : {0:g}", dt));
 Console.WriteLine(String.Format("Combination of d and T [G] : {0:G}", dt));
 Console.WriteLine(String.Format("Month Day Pattern [m or M] : {0:M}", dt));
 Console.WriteLine(String.Format("Year Month Pattern [y or Y] : {0:Y}", dt));
 Console.WriteLine(String.Format("RFC1123Pattern [r or R] : {0:R}", dt));
 Console.WriteLine(String.Format("SortableDateTimePattern [s] : {0:s}", dt));
 Console.WriteLine(String.Format("UniversalSortableDateTimePattern [u] : {0:u}", dt));
 }
}

```

**Output:**

```

Short Time Pattern [t] : 9:53 AM
Short Date Pattern [d] : 17/02/17
Long Time Pattern [T] : 9:53:09 AM
Long Date Pattern [D] : Friday 17 February 2017
Combination of D and T [f] : Friday 17 February 2017 9:53 AM

```

Full Date Time Pattern [F] : Friday 17 February 2017 9:53:09 AM  
 Combination of d and t [g] : 17/02/17 9:53 AM  
 Combination of d and T [G] : 17/02/17 9:53:09 AM  
 Month Day Pattern [m or M] : 17 February  
 Year Month Pattern [y or Y] : February 2017  
 RFC1123Pattern [r or R] : Fri, 17 Feb 2017 09:53:09 GMT  
 SortableDateTimePattern [s] : 2017-02-17T09:53:09  
 UniversalSortableDateTimePattern [u] : 2017-02-17 09:53:09Z

### **String.Format for Double, Decimal, Float or Int**

- Several times you need to get customize output for Double, Decimal or Float value. Sometimes output in this datatype is too long after point precision and you just need to cut rest of the value after point precision. String.Format helps you lot in customizing Double, Decimal or Float value.

#### **Formatting Table:**

| Format     | Explain                                | Value     | Output        |
|------------|----------------------------------------|-----------|---------------|
| {0}        | Original Value                         | 83745.892 | 83745.892     |
| [0:0.00]   | Two decimal places after point.        | 83745.892 | 83745.89      |
| [0:0.##]   | Maximum Two decimal places after point | 83745.892 | 83745.89      |
| [0:0.00]   | Two digits before decimal point        | 5.892     | 05.892        |
| [0:0,0.00] | Thousands Separator                    | 83745.892 | 83,745.89     |
| {0:0.0}    | Point number conversion with zero      | 0.5       | 0.5           |
| {0:#.0}    | Point number conversion without zero   | 0.5       | .5            |
| [0,10:0.0] | Align Number with Spaces               | 120.5     | " 120.5"      |
| [0,10:0.0] | Align Number with Spaces               | 120.5     | "120.5 Hello" |

#### **Program 5:**

```

using System;
namespace StringFormat_Double
{
 public class Program
 {
 public static void Main(string[] args)
 {
 //Start Formation
 }
 }
}

```

```

 Console.WriteLine("Current Output : {0}", 83745.892);
 Console.WriteLine(string.Format("Two decimal places : {0:0.00}",
 83745.892));
 Console.WriteLine(string.Format("Three decimal places
 {0:0.000}", 83745.82));
 Console.WriteLine(string.Format("Maximum Two decimal places
 {0:0.##}", 83745.892));
 Console.WriteLine(string.Format("Two digits before decimal point
 : {0:00.00}", 5.892));
 Console.WriteLine(string.Format("Thousands Separator
 {0:0,0.00}", 83745.892));
 Console.WriteLine(string.Format("Point number conversion with
 zero : {0:0.0}", 0.5));
 Console.WriteLine(string.Format("Point number conversion without
 zero : {0:#.0}", 0.5));
 Console.WriteLine(string.Format("Align Number with Spaces :
 {0,10:0.0}", 120.5));
 Console.WriteLine(string.Format("Align Number with Spaces : {0,
 10:0.0} Hello", 120.5));
 Console.WriteLine(string.Format("Phone Number format {0:###-##-
 ####}", 123456789));
 }
}
}

```

**Output:**

Current Output : 83745.892  
 Two decimal places : 83745.89  
 Three decimal places : 83745.820  
 Maximum Two decimal places : 83745.89  
 Two digits before decimal point : 05.89  
 Thousands Separator : 83,745.89  
 Point number conversion with zero : 0.5  
 Point number conversion without zero : .5  
 Align Number with Spaces : 120.5  
 Align Number with Spaces : 120.5 Hello  
 Phone Number format 123-45-6789

## 3.2 OBJECT ORIENTED CONCEPTS

### 3.2.1 Defining Classes and Objects

- **Classes** allow you to control all the functions that can be applied to a given set of data as well as how to access the data.
- **Classes** also give you the ability to extend a runtime environment and allow the developer to reuse the existing functionalities of the runtime environment.

#### Creating a Class:

- User can **create** a class using the class keyword, followed by the class name.
- A **class** is similar to a container that may have data members (variables, constants or fields) and member functions (methods, properties, events, operators, instance constructors, destructors) and other classes.
- A **class** also supports inheritance, which is a mechanism in which a derived class extends a base class.
- The basic syntax of the class structure is as below-

```
class <classname>
{
 <access-modifier> [static] <return-type>MethodName (<signature>);
 <access-modifier> [static] <variable-type>variablename;
}
```

**Example:** Showing the code to create a class.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace MyClass
{
 class program
 {
 static void Main (string [] args)
 {
 }
 public class Amount
 {
 doubleTotalAmount, deductions, balanceamount;
 public void Balance()
 {
 //method body;
 }
 }
 }
}
```

```
public void Debit()
{
 //method body;
}
public void credit()
{
 //method body;
}
public void GetData()
{
 //method body;
}
```

- In above code we create a class Amount and in this class we specify the data members TotalAmount, Deductions and BalanceAmount as well as methods Balance(), Debit(), Credit() and GetData().

#### Creating an Object:

- In C#, objects helps you to access the members of a class (fields, methods and properties) by using dot(.) operator.
- The basic syntax for declaring an object is

```
<ClassName><ObjectName> = new <ClassName>();
```

- Following Example Illustrate how to access class member and member functions through objects.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace MyClass
{
 Class program
 {
 static void Main (string [] args)
 {
 Amount Amt = new Amount();
 Amt.TotalAmount=12000;
 Amt.deductions = 2000;
 Amt.Balance();
 Amt.GetData();
 }
 }
}
```

```
public class Amount
{
 double TotalAmount, deductions, balanceamount;
 public void Balance()
 {
 //method body;
 }
 public void Debit()
 {
 //method body;
 }
 public void credit()
 {
 //method body;
 }
 public void GetData()
 {
 //method body;
 }
}
```

- In above code, the Amt object of the Amount class accesses the data members i.e., TotalAmount, deductions and the methods (Balance and GetData) through a dot(.) operator.

**Use of 'this' Keyword:****[S-22, W-22]**

- The this keyword refers to the current instance of a class. With the this keyword, you can access an instance of a class and use it with instance methods, instance constructors and instance properties.
- Note that the this keyword cannot be used with static members because static members are accessed by a class and not by the instance of the class.

**Example to illustrate the use of 'this' keyword:**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace Mythiskeyword
{
 class program
 {
```

```
static void Main (string [] args)
{
 Amount Amt = new Amount();
 Amt.Balance();
 Amt.GetData();
 Console.WriteLine("\n Press Enter to Quit..");
 Console.ReadLine();
}

public class Amount
{
 doubleTotalAmount, deductions, balanceamount;
 public void Balance()
 {
 this.TotalAmount = 12000;
 this.deductions = 2000;
 balanceamount = TotalAmount - deductions;
 Console.WriteLine("Your Current Amount is Rs: " +BalanceAmount);
 }
 public void Debit()
 {
 intDebitAmount = 0;
 Console.write("Enter Amount");
 DebitAmount = int.Parse(Console.ReadLine());
 BalanceAmount = BalanceAmount - DebitAmount;
 Console.WriteLine("Your Balance Amount is Rs:" + BalanceAmount);
 }
 public void Credit()
 {
 intCreditAmount = 0;
 Console.Write("Enter Amount");
 CreditAmount = int.Parse(Console.ReadLine());
 BalanceAmount = BalanceAmount+CreditAmount;
 Console.WriteLine("Your Balanec Amount id Rs." +BalanceAmount);
 }
 public void GetData()
 {
 Char Data;
 Console.WriteLine("-----");
 Console.write("Press (d) for debit and (c) for credit");
 Data = char.Parse(console.ReadLine());
 }
}
```

```
 if (Data == 'd')
 {
 Debit();
 }
 if(Data == 'c')
 {
 Credit();
 }
}
}
```

### 3.2.2 Access Modifiers

#### Why Access Modifiers?

- To control the visibility of class members (the security level of each individual class and class member).
- To achieve "**Encapsulation**", this is the process of making sure that "sensitive" data is hidden from users.
- Access modifiers help to avoid jumbling of data and methods with the existing code as well as protect an object of a class from outside interference.
- These modifiers do this by defining a certain scope to access data and methods in a restricted manner.
- You can declare a class and each of its methods with an access modifier.
- Each method, however, can contain only one modifier. If you are declaring a class that is not inside any other class, then that class can make use of the following two modifiers:
  - Public modifier
  - Internal modifier
- On the other hand, nested classes can make use of following five modifiers:
  1. **public modifier:** Allows public access to members both inside and outside a class without any restrictions.
  2. **internal modifier:** Allows internal access to members. Only the current assembly can access these members. If a member of the internal accessibility level is accessed outside the assembly in which it has been defined, an error is generated.
  3. **protected modifier:** Allows protected access to members. You can access protected members from either the class in which they are declared or a class derived from the class in which they are declared.
  4. **private modifier:** Allows private access to members. Private members have the least access permission level; you can either access them within the body of the class or in the structure in which they are declared.
  5. **protected internal modifier:** Allows access to the members of the current assembly, the containing class, and the class derived from the containing class.

**1. Public Modifier:**

- If you declare a field with a public access modifier, it is accessible for all classes.

**Example:**

```
class Car
{
 public string model = "Mustang";
}

class Program
{
 static void Main(string[] args)
 {
 CarmyObj=newCar();
 Console.WriteLine(myObj.model);
 }
}
```

**The output will be:****Mustang**

**Note:** By default, all members of a class are private if you don't specify an access modifier.

**2. Internal Modifier:**

- Access is limited to only the current Assembly, that is any class or type declared as internal is accessible anywhere inside the same namespace. It is the default access modifier in C#.

**Syntax:** internal TypeName

**Example:** In the code given below, The class Complex is a part of internal Access Modifier namespace and is accessible throughout it.

```
// C# Program to show use of
// internal access modifier
// Inside the file Program.cs
using System;
namespace internal AccessModifier
{
 // Declare class Complex as internal
 internal class Complex
 {
 int real;
 int img;
```

```

 public void setData(int r, int i)
 {
 real = r;
 img = i;
 }
 public void displayData()
 {
 Console.WriteLine("Real = {0}", real);
 Console.WriteLine("Imaginary = {0}", img);
 }
 }
 // Driver Class
 class Program
 {
 // Main Method
 static void Main(string[] args)
 {
 // Instantiate the class Complex
 // in separate class but within
 // the same assembly
 Complex c = newComplex();
 // Accessible in class Program
 c.setData(2, 1);
 c.displayData();
 }
 }
}

```

**Output:**

Real = 2

Imaginary = 1

**Note:** In the same code if you add another file, the class *Complex* will not be accessible in that namespace and compiler gives an error.

```

// C# program inside file xyz.cs
// separate namespace named xyz
using System;
namespace xyz
{
 class text
 {
 // Will give an error during compilation
 Complex c1 = newComplex();
 c1.setData(2, 3);
 }
}

```

**Output:**

error CS1519

**3. Protected Modifier:**

- Access is limited to the class that contains the member and derived types of this class. It means a class which is the subclass of the containing class anywhere in the program can access the protected members.

**Syntax:** protected TypeName

**Example:** In the code given below, the class Y inherits from X, therefore, any protected members of X can be accessed from Y but the values cannot be modified.

C# Program to show the use of protected Access Modifier

```
using System;
namespace protected AccessModifier
{
 Class X
 {
 // Member x declared
 // as protected
 protected int x;
 public X()
 {
 x = 10;
 }
 }
 // class Y inherits the
 // class X
 class Y : X
 {
 // Members of Y can access 'x'
 public int getX()
 {
 return x;
 }
 }
 class Program
 {
 static void Main(string[] args)
 {
 X obj1 = newX();
 Y obj2 = newY();
 // Displaying the value of x
 Console.WriteLine("Value of x is : {0}", obj2.getX());
 }
 }
}
```

**Output:**

Value of x is : 10

**4. private Modifier:**

- If you declare a field with a private access modifier, it can only be accessed within the same class.

**Example:**

```
class Car
{
 private string model = "Mustang";
 static void Main(string[] args)
 {
 Car myObj=newCar();
 Console.WriteLine(myObj.model);
 }
}
```

**The output will be:**

Mustang

- If you try to access it outside the class, an error will occur.

**Example:**

```
class Car
{
 private string model = "Mustang";
}
class Program
{
 static void Main(string[] args)
 {
 Car myObj=newCar();
 Console.WriteLine(myObj.model);
 }
}
```

**The output will be:**

'Car.model' is inaccessible due to its protection level  
The field 'Car.model' is assigned but its value is never used

**5. Protected Internal Modifier:**

- Access is limited to the current assembly or the derived types of the containing class. It means access is granted to any class which is derived from the containing class within or outside the current Assembly.

**Syntax:** protected internal TypeName

**Example:** In the code given below, the member 'value' is declared as protected internal therefore it is accessible throughout the class Parent and also in any other class in the same assembly like ABC. It is also accessible inside another class derived from Parent, namely Child which is inside another assembly.

```

 / Inside file parent.cs
 using System;
 public class Parent
 {
 // Declaring member as protected internal
 protected internal int value;
 }
 class ABC
 {
 // Trying to access
 // value in another class
 public void testAccess()
 {
 // Member value is Accessible
 Parent obj1 = new Parent();
 obj1.value = 12;
 }
 }
 // Inside file GFG.cs
 using System;
 namespace GFG
 {
 class Child : Parent
 {
 // Main Method
 public static void Main(String[] args)
 {
 // Accessing value in another assembly
 Child obj3 = new Child();
 // Member value is Accessible
 obj3.value = 9;
 Console.WriteLine("Value = " + obj3.value);
 }
 }
 }

```

**Output:**

Value = 9

### 3.2.3 Constructors and Destructors

- A constructor is called when an object is first created. A destructor (or finalized) is called when the object is finally destroyed and the garbage collected.

- The constructor is called with the same name as the class; whereas the name of the destructor is made up of the class name prefixed with a tilde (~) sign. You can use a constructor to initialize objects and set any parameters. In addition, you can write constructors to accept argument.

**For Example:**

```
class Student
{
 string a, b;
 int c;
 Student(string s)
 {
 a=s;
 }
 Student(int I, string s)
 {
 b=s;
 c=I;
 }
 ~Student()
 {
 }
}
```

- C# has a garbage collection mechanism that runs to destroy an object when its reference count drops to 0.
- User can also destroy the object that is no longer used by calling the destructors in the program.

**Parameterized Constructor:**

- C# programming language allows you to create parameterized constructors in a class. A constructor of a class that accepts some arguments is called a parameterized constructor. To see how to create a parameterized constructor in a class let us see below example:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace MyParamConstructordemo
{
 class program
 {
```

```

 static void Main(string[] args)
 {
 Paramconstructor obj1 = new Paramconstructor(); //no parameter
 Paramconstructor obj2 = new Paramconstructor("Martin"); //one parameter
 Paramconstructor obj3 = new Paramconstructor("Martin", 26);
 //Two parameter
 Console.write("\nPress Enter to Quit....");
 Console.ReadLine();
 }
 }
 public class Paramconstructor
 {
 public Paramconstructor()
 {
 }
 public Paramconstructor(String Name)
 {
 Console.WriteLine("Name is:" + Name);
 Console.WriteLine("-----");
 }
 public Paramconstructor(String Name, int Age)
 {
 Console.WriteLine("Name is:" + Name);
 Console.WriteLine("Age is:" + Age);
 }
 }
}

```

### **Creating a Copy Constructor-**

- A copy constructor is used to create a copy of an existing object as new object. This constructor takes the reference of the object to be copied as an argument. C# does not provide the copy constructor by default; however, you can have an identical copy of an existing object. To create identical copies of an object having the same value for properties and members as the source object, you need to write a separate method or copy a constructor, as shown in following code:

#### **Class Example:**

```

{
 //create a new CarDetails object
 Static CarDetails car1 = new CarDetails("Zen Estilo",15.7)
 //create another new object, copying car1
 CarDetails car2 = new CarDetails(Example.car1);
}

```

```

class CarDetails
{
 //Data Members
 public CarDetails(CarDetails MyCar) //copy constructor
 {
 //constructor body
 }
 public CarDetails(string ModelName, double Mileage)
 //Instance constructor
 {
 //constructor body
 }
}

```

**Calling a Destructor of a Class:**

- Destructor is a method that is called to destroy the object that is no longer in use.
- A class can have only one destructor. A destructor is declared using the tilde (~) sign, followed by the name of destructor.

To see how to create a destructor of a class refer following code-

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace MyDestructor
{
 class Program
 {
 Static void Main(string [] args)
 {
 Destruct obj = new Destruct();
 }
 }
 class Destruct
 {
 ~Destruct()
 {
 console.WriteLine("Destructor is called");
 console.write("\nPress Enter to Quite");
 console.ReadLine();
 }
 }
}

```

### 3.2.4 Inheritance

- The most important reason to use OOP is to promote reusability of code and eliminate the redundant code.
- To reduce redundancy, the object-oriented languages support inheritance. Inheritance is the property through which a class derives properties from another class.
- A class that inherits the properties of another class is called a child or derived class, whereas, the class from which the child class inherits properties is known as a parent or base class.
- A parent class is at a higher level in the class hierarchy. A class hierarchy defines the logical structuring of the classes, such that a general class is on the top of the class hierarchy and more specialized classes are at the lower levels.
- For example, the Animal class can be a base class for a number of derived classes, such as Dog, Cow and Tiger.
- C# supports single inheritance, hierarchical inheritance and multilevel inheritance because there is only a single base class.
- It does not support multiple inheritances directly. To implement multiple inheritances, you need to use interfaces.
- The class which allows inheritance of its common properties is known as a base or parent class and the class that inherits the properties of base class is known as a derived or child class.

#### Types of inheritance:

- Single Inheritance:** Refers to inheritance in which there is only one base class and one derived class. This means that a derived class inherits properties from single base class.
- Hierarchical Inheritance:** Refers to inheritance in which multiple derived classes are inherited from the same base class.
- Multilevel Inheritance:** Refers to inheritance in which a child class is derived from a class, which in turn is derived from another class.

#### Program 6: An application showing the code of Inheritance Application.

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace MyInheritance
{
 public class BaseClass
 {
 public int intDataMember;
 }
}

```

```
public void BaseClassMethod()
{
 console.WriteLine("In a BaseClassMethods");
}

public class DerivedClass : BaseClass
{
 public void DerivedClassMethod()
 {
 console.WriteLine("In a derived class method");
 }
}

public class Program
{
 public static void Main(string [] args)
 {
 //create a base class object
 console.WriteLine("accessing base class object");
 BaseClassbc=new BaseClass();
 bc.DataMember = 1;
 bc.BaseClassMethod();
 //create a Derived class object
 console.WriteLine("accessing Derived class object");
 DerivedClass dc = new DerivedClass();
 dc.DataMember = 2;
 dc.BaseClassMethod();
 dc.DerivedClassMethod();
 console.write("\n Press enter to quit..");
 console.ReadLine();
 }
}
```

}

Output:

```
accessing base class object
In a BaseClassMethods
accessing Derived class object
In a BaseClassMethods
In a derived class method
Press enter to quit..
```

## Inheritance and Constructors:

A constructor is a special method of a class, which is used to initialize the member of the same class. A constructor is called by default whenever an object of a class is created. It is important to note that a derived class cannot inherit the constructor of its base class. All the derived classes have their default constructor. When user instantiate a derived class, the constructor of the base class is not made available to it, however, the default constructor of the base class is called automatically and then the derived class constructor is called. In this way, the constructor of the base class initializes the members of the base class, before the derived class constructor is executed.

## Sealed Classes and Sealed Methods:

Sealed classes are classes that cannot be inherited. You can use the sealed keyword to define a class as a sealed class.

```
sealed class <Class Name>
{
 //Data Members and Member Functions
}
```

The following code shows how to define a sealed class-

```
sealed class MyClass
{
 //Data member
 public void GetDetail ()
 {
 }
 public void ShowDetail ()
 {
 }
}
```

```
class MainClass
```

```
{
 //Instantiation of MyClass class
 //Method Calling
}
```

- However, if user attempt to derived a class from the MyClass class as shown in the following code:

```
Class DerivedClass :MyClass
```

User will get an error message.

### 3.2.5 Interfaces

- An interface is introduced to provide the feature of multiple inheritances to classes.
- Multiple inheritances is the feature of OOP which allows a class to inherit from multiple classes.
- The methods defined in an interface do not have implementation, they only specify the parameters that they will take and the types of values they will return.
- In C#, interfaces are more flexible in their implementation, than in any other language. An interface in C# is the equivalent of an abstract base class. This implies that user cannot instantiate an object through an interface, but user can offer a set of functionalities to classes that implement the interface.
- An interface can have the same access modifier as a class, such as public and private.

#### Syntax of Interfaces:

- An interface is declared just as a class, but the only difference is that a class is declared using the **class** keyword; whereas, an interface is declared using the **interface** keyword. Interface behaves as templates that have method declarations in them. These methods are abstract in nature, as they do not have code in their body.
- As discussed, a class that implements an interface must implement all of its methods; otherwise, an error occurs.

#### User can declare an interface using following code:

```
interface <interface name>
{
 //abstract method declarations in Interface body
}
```

- The interface definition itself contains no code, only the signatures of the actual method. The implementation of the methods is left to the class that inherits the interface.

#### Creating an interface:

```
public interface Channel
{
 void Next();
 void Previous();
}

public interface Book
{
 void Next();
 void Chapter();
}
```

- In above example, the Channel interface has two abstract methods named **Next()** and **Previous()**. The Book interface has two abstract methods named **Next()** and **Chapter()**.

- All of these methods are void and do not return any value. In this case, any inheriting class can implement the methods of the interfaces Channel and Book.

### Implementation of Interfaces:

- To implement multiple inheritances in a program, user needs to use interfaces. Implementing an interface is similar to inheriting a class.
- Example:** Let's create an application named Accessing Interface that shows how to implement an interface in a class.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace AccessingInterface
{
 public interface Channel
 {
 void Next();
 void Previous();
 }
 public interface Book
 {
 void Next();
 void Chapter();
 }
 class Program : Channel, Book //Implementing Interface in a class
 {
 void Channel.Next()
 {
 console.WriteLine("Channel next");
 }
 void Book.Next()
 {
 console.WriteLine("Book next");
 }
 public void Previous()
 {
 console.WriteLine("Previous");
 }
 public void Chapter()
 {
 console.WriteLine("Chapter");
 }
 }
}
```

```

 static void Main(string [] args)
 {
 Program app = new Program();
 ((Book)app).Next();
 app.Previous();
 app.Chapter();
 Console.write("\n press Enter to quit...");
 Console.ReadLine();
 }
}

```

### 3.2.6 Abstract Class

- Abstraction is the process of hiding the details of a particular object from a user and exposing only the essential features. The concepts of abstraction and encapsulation are complementary to each other.
- In terms of OOP, abstraction refers to the act of representing essential features without including the background details or explanations. Abstraction is the development of classes, objects, and types in terms of their functionality, instead of their implementation details.
- Abstraction simply denotes a model, a view, or a representation for an actual item (class, object or type).
- The main characteristics of abstraction are as follows:
  - Managing the complexity of the code.
  - Decomposing complex system into smaller components.

#### Abstract Class:

- In C#, user can have a single base class and multiple derived classes. If we want to restrict a class from creation of an object of the base class, user can make the base class as abstract.
- The abstract keyword in a class indicates that the class cannot be instantiated. User can create the instance of its derived class (only if it is not marked as abstract).

#### Creating an Abstract Class:

```

abstract class BaseClass
{
 //Data members and member functions
}

class DerivedClass:BaseClass
{
 //Data members and member functions
}

```

```

class MainClass()
{
 BaseClass bc; //can't be instantiated
 DerivedClass dc; // can't be instantiated
}

```

- In above code we cannot instantiate BaseClass because we have declared it as abstract.

#### **Some characteristics of an abstract class are as follows:**

- Restricts instantiation, implying that user cannot create an object of an abstract class.
- Allows user to define abstract as well as non-abstract members in it.
- Requires at least one abstract method in it.
- Restricts the use of sealed keyword in it.
- Possesses public access specifier therefore, it can be used anywhere in a program.

#### **Abstract Methods:**

- An abstract method is similar to a virtual method which does not provide any implementation; therefore, an abstract method does not have method body.

#### **Syntax to create abstract method:**

```
public abstract void Area (int Length, int Breadth);
```

#### **Some characteristics of abstract methods are as follows:**

1. Restricts its implementation in an abstract class.
2. Allows implementation in a non-abstract derived class.
3. Requires declaration in an abstract class only.
4. Restricts declaration with static and virtual keywords.
5. Allows user to override a virtual method.

#### **Example: Implementing an Abstract Method.**

```

public abstract class Shape
{
 public abstract void Area(int Length, int Breadth)
 {
 //Method Area() can't be implemented at this place.
 }
}
public abstract class BasicShape: Shape
{
 //Method Area() can't be implemented at this place also
}

```

```

public class Rectangle: BasicShape
{
 public override void Area(int Length, int Breadth)
 {
 //Method Area() can't be implemented at this place too
 }
}

```

**Polymorphism:**

- Polymorphism is one of the important features of OOP that is used to exhibit different forms of any particular procedure. With the help of polymorphism, you can use one procedure in many ways as per your requirements.
- The advantages of polymorphism are as follows:
  - Allows user to invoke methods of a derived class through base class reference during runtime.
  - Provides different implementations of methods in a class that are called through the same name.

**Polymorphism is categorized into:**

1. Static polymorphism/compile time polymorphism/Overloading
2. Dynamic polymorphism/run time polymorphism/overriding

**3.2.7 Method Overloading and Overriding**

[S-23]

**Method Overloading:**

[S-22]

- Method overloading is a concept in which a method behaves according to the number and types of parameters passes to it.
- In method overloading, user can define many methods with the same name but different signatures.
- A method signature is the combination of the method's name along with the number, type and order of the parameters.
- When user call overloaded methods, a compiler automatically determines which method should be used according to the signature specified in the method call.
- Method overloading is used when methods are required to perform similar task but with different input parameters.
- Note that only the return type does not play any important role in the method overloading, it must take consideration of the number or type of arguments passed to the method.

**Example: Showing the code of the MethodOverload Application**

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

```

```

namespace Methodoverload
{
 public class Shape
 {
 public void Area(int side)
 {
 int SquareArea= side* side;
 console.WriteLine("The area of square is:"+SquareArea);

 }
 public void Area(int Length, int Breadth)
 {
 int RectangleArea = Length * Breadth;
 console.WriteLine("The area of rectangle is:"+RectangleArea);

 }
 public void Area(double Radius)
 {
 double CircleArea=3.14*Radius*Radius;
 console.WriteLine("The area of circle is:"+CircleArea);

 }
 }
 class MainClass
 {
 static void Main(string [] args)
 {
 Shape sh = new Shape();
 sh.Area(15);
 sh.Area(10,20);
 sh.Area(10.5);
 console.write("\n Press Enter to quit....");
 console.ReadLine();
 }
 }
}

```

**Output:**

The area of square is:225

The area of rectangle is:200

The area of circle is:346.185

Press Enter to quit...

- In above example `Area()` method of `Shape` class is overloaded for calculating areas of Square, Rectangle and Circle.
- In the `Main()` method, the `Area()` method is called multiple times by passing different arguments.

- Note that C# supports operator overloading which enables user such that user can change the functionality of an operator by overloading them.
- Operator overloading is a mechanism of assigning a special meaning to an operator according to user defined data type, such as classes or structs.
- It is not possible to overload all the operators, only the specific operators can be overloaded.

| Operators                                   | Type              | Overloading Status                                                      |
|---------------------------------------------|-------------------|-------------------------------------------------------------------------|
| +, -, !, ~, ++, --,<br>true, false          | Unary             | Can be Overloaded.                                                      |
| +, -, *, /, %, &,  ,<br>&, <<, >>           | Binary            | Can be Overloaded.                                                      |
| ==, !=, <, >, <=, >=                        | Comparison        | Can be Overloaded.                                                      |
| &&,                                         | Conditional Logic | Cannot be overloaded but can be evaluated using & and  .                |
| [ ]                                         | Array Indexing    | Cannot be overloaded but can help in defining indexes.                  |
| ( )                                         | Cast              | Cannot be overloaded but can help in defining new conversion operators. |
| =, ., ?, :, ->, new<br>, is, sizeof, typeof | other             | Cannot be overloaded.                                                   |

### Method Overriding:

- Method Overriding in C# is similar to the virtual function in C++. Method Overriding is a technique that allows the invoking of functions from another class (base class) in the derived class. Creating a method in the derived class with the same signature as a method in the base class is called as method overriding.
- In simple words, Overriding is a feature that allows a subclass or child class to provide a specific implementation of a method that is already provided by one of its super-classes or parent classes. When a method in a subclass has the same name, same parameters or signature and same return type (or sub-type) as a method in its super-class, then the method in the subclass is said to override the method in the super-class. Method overriding is one of the ways by which C# achieve Run Time Polymorphism (Dynamic Polymorphism).
- The method that is overridden by an override declaration is called the overridden base method. An override method is a new implementation of a member that is inherited from a base class. The overridden base method must be virtual, abstract, or override.

**Note:**

- Method overriding is possible only in derived classes. Because a method is overridden in the derived class from the base class.
- A non-virtual or a static method can't be overridden.
- Both the override method and the virtual method must have the same access level modifier.

**Example:**

```

class base_class
{
 public void gfg();
}

class derived_class : base_class
{
 public void gfg();
}

class Main_Method
{
 static void Main()
 {
 derived_class d = new derived_class();
 d.gfg();
 }
}

```

- Here the base class is inherited in the derived class and the method gfg() which has the same signature in both the classes, is overridden.

In C# we can use 3 types of keywords for Method Overriding:

[S-23]

- virtual keyword:** This modifier or keyword use within base class method. It is used to modify a method in base class for overridden that particular method in the derived class.
- override:** This modifier or keyword use with derived class method. It is used to modify a virtual or abstract method into derived class which presents in base class.

```

class base_class
{
 public virtual void gfg();
}

class derived_class : base_class
{
 public override void gfg();
}

```

```
class Main_Method
{
 static void Main()
 {
 derivedd_class = new derived_class();
 d.gfg();
 base_class b = new derived_class();
 b.gfg();
 }
}
```

Here first, d refers to the object of the class derived\_class and it invokes gfg() of the class derived\_class then, b refers to the reference of the class base and it hold the object of class derived and it invokes gfg() of the class derived. Here gfg() method takes permission from base class to overriding the method in derived class.

#### Method overriding using virtual and override modifiers:

```
// C# program to illustrate the use of
// 'virtual' and 'override' modifiers
using System;
class baseClass
{
 // show() is 'virtual' here
 public virtual void show()
 {
 Console.WriteLine("Base class");
 }
}
// class 'baseClass' inherit
// class 'derived'
class derived : baseClass
{
 // 'show()' is 'override' here
 public override void show()
 {
 Console.WriteLine("Derived class");
 }
}
class GFG
{
 // Main Method
```

```

public static void Main()
{
 baseClassobj;
 // 'obj' is the object
 // of class 'baseClass'
 obj = new baseClass();
 // it invokes 'show()'
 // of class 'baseClass'
 obj.show();

 // the same object 'obj' is now
 // the object of class 'derived'
 obj = new derived();

 // it invokes 'show()' of class 'derived'
 // 'show()' of class 'derived' is overridden
 // for 'override' modifier
 obj.show();
}
}

```

**Output:**

Base class

Derived class

3. **base Keyword:** This is used to access members of the base class from derived class. It basically used to access constructors and methods or functions of the base class. The base keyword cannot use within a static method. Base keyword specifies which constructor of the base class should be invoked while creating the instances of the derived class.

**Use of Base keyword:**

- Call methods or functions of base class from derived class.
- Call constructor internally of base class at the time of inheritance.

```

// C# program to show the use of 'base'
// keyword in method overriding
using System;

// base class
public class web
{
 string name = "GeeksForGeeks";
 // 'showdata()' is member method,
 // declare as virtual

```

```
public virtual void showdata()
{
 Console.WriteLine("Website Name: " + name);
}

// derived class
// class 'web' is inherits
// class 'stream'
class stream : web
{
 string s = "Computer Science";
 // 'showdata()' is overridden
 // in derived class
 public override void showdata()
 {
 // Calling 'showdata()' of base
 // class using 'base' keyword
 base.showdata();
 Console.WriteLine("About: " + s);
 }
}
class GFG
{
 // Main Method
 static void Main()
 {
 // 'E' is object of class stream
 // also works as object of
 // class 'web'
 stream E = new stream();
 // it first invokes 'showdata()'
 // of class 'web' then it invokes
 // 'showdata()' of class 'stream'
 E.showdata();
 }
}
```

**Output:**

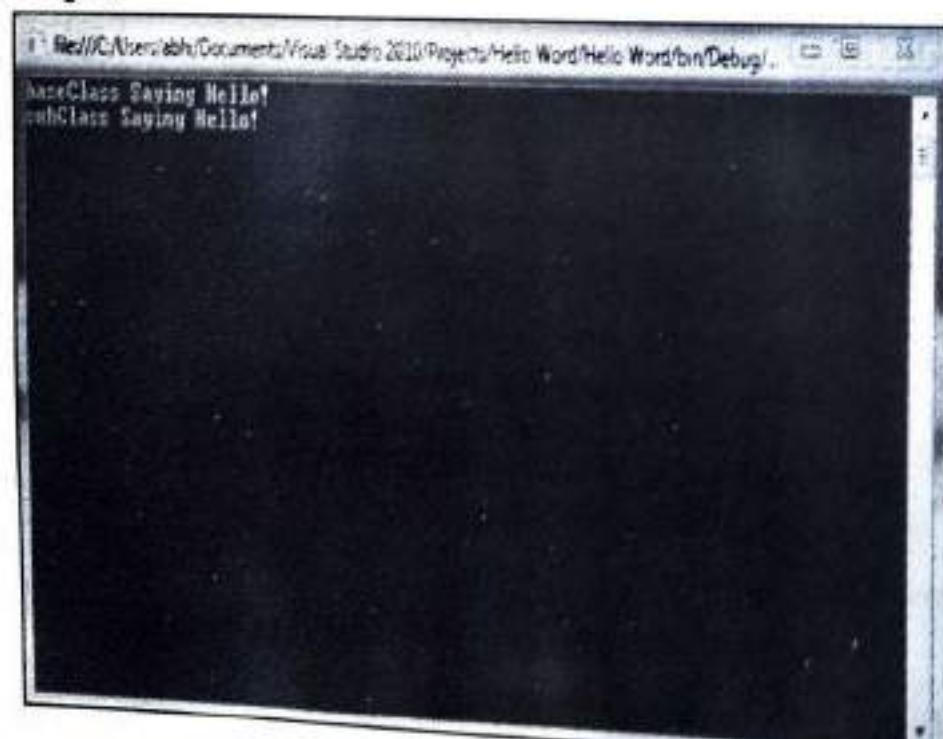
```
Website Name: GeeksForGeeks
About: Computer Science
```

**Method Overriding:**

- Overriding can be defined as: being able to change or augment the behavior of methods in classes, known as overriding their logic; it is one of the most powerful aspects of Object Oriented Programming.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
```

```
namespace Hello_World
{
 class baseClass
 {
 public virtual void Greetings()
 {
 Console.WriteLine("baseClass Saying Hello!");
 }
 }
 class subClass : baseClass
 {
 public override void Greetings()
 {
 base.Greetings();
 Console.WriteLine("subClass Saying Hello!");
 }
 }
 class Program
 {
 static void Main(string[] args)
 {
 baseClass obj1 = new subClass();
 obj1.Greetings();
 Console.ReadLine();
 }
 }
}
```

**Output:**

### 3.2.3 Delegates

- A delegate is a type that represents references to methods with a particular parameter list and return type. When you instantiate a delegate, you can associate its instance with any method with a compatible signature and return type. You can invoke (or call) the method through the delegate instance.
- C# delegates are similar to pointers to functions, in C or C++. A **delegate** is a reference type variable that holds the reference to a method. The reference can be changed at runtime.
- What if we want to pass a function as a parameter? How does C# handles the callback functions or event handler? The answer is - delegate.
- The delegate is a reference type data type that defines the method signature. You can define variables of delegate, just like other data type, that can refer to any method with the same signature as the delegate.
- Delegate is also used to declare an Event and an Anonymous Method
- There are three steps involved while working with delegates:
  1. Declare a delegate
  2. Set a target method
  3. Invoke a delegate
- Delegates are especially used for implementing events and the call-back methods. All delegates are implicitly derived from the `System.Delegate` class.

#### Declaring Delegates:

- Delegate declaration determines the methods that can be referenced by the delegate. A delegate can refer to a method, which has the same signature as that of the delegate.
- For example, consider a delegate:
 

```
public delegate int MyDelegate (string s);
```
- The preceding delegate can be used to reference any method that has a single `string` parameter and returns an `int` type variable.
- Syntax for delegate declaration is:  
A delegate can be declared using the `delegate` keyword followed by a function signature, as shown below.

```
[modifier] delegate [return type] [delegate name]([parameters])
```

#### Instantiating Delegates:

- Once a delegate type is declared, a delegate object must be created with the `new` keyword and be associated with a particular method. When creating a delegate, the argument passed to the `new` expression is written similar to a method call, but without the arguments to the method.
- For example:

```
Public delegate void printString(string s);
```

```
..
PrintString ps1 = new printString(WriteToScreen);
PrintString ps2 = new printString(WriteToFile);
```

- Following example demonstrates declaration, instantiation, and use of a delegate that can be used to reference methods that take an integer parameter and returns an integer value.

```

using System;
delegate int NumberChanger(int n);
namespace DelegateApp1{
 class TestDelegate{
 static int num=10;
 public static int AddNum(int p)
 {
 num+= p;
 return num;
 }
 public static int MultNum(int q)
 {
 num*= q;
 return num;
 }
 public static int getNum()
 {
 return num;
 }
 static void Main(string[]args)
 {
 //create delegate instances
 NumberChanger nc1 =newNumberChanger(AddNum);
 NumberChanger nc2 =newNumberChanger(MultNum);
 //calling the methods using the delegate objects
 nc1(25);
 Console.WriteLine("Value of Num: {0}",getNum());
 nc2(5);
 Console.WriteLine("Value of Num: {0}",getNum());
 Console.ReadKey();
 }
 }
}

```

- When the above code is compiled and executed, it produces the following result:

Value of Num: 35

Value of Num: 175

#### Multicasting of a Delegate:

- Delegate objects can be composed using the "+" operator. A composed delegate contains the two delegates it was composed from. Only delegates of the same type can be composed. The "-" operator can be used to remove a component delegate from a composed delegate.

- Using this property of delegates you can create an invocation list of methods that will be called when a delegate is invoked. This is called multicasting of a delegate. The following program demonstrates multicasting of a delegate -

```

using System;
delegate int NumberChanger(int n);
namespace DelegateAppl{
 class TestDelegate{
 static int num=10;
 public static int AddNum(int p)
 {
 num+= p;
 return num;
 }
 public static int MultNum(int q)
 {
 num*= q;
 return num;
 }
 public static int getNum()
 {
 return num;
 }
 }
 static void Main(string[] args)
 {
 //create delegate instances
 NumberChanger nc;
 NumberChanger nc1 =newNumberChanger(AddNum);
 NumberChanger nc2 =newNumberChanger(MultNum);
 nc= nc1;
 nc+= nc2;
 //calling multicast
 nc(5);
 Console.WriteLine("Value of Num: {0}",getNum());
 Console.ReadKey();
 }
}
}

```

When the above code is compiled and executed, it produces the following result:

Value of Num: 75

#### Generic Delegate:

- A generic delegate can be defined the same way as a delegate but using generic type parameters or return type. The generic type must be specified when you set a target method.

- For example, consider the following generic delegate that is used for int and string parameters.

```

Example: Generic Delegate
public delegate T add<T>(T param1, T param2); // generic delegate
class Program
{
 static void Main(string[] args)
 {
 add<int> sum = Sum;
 Console.WriteLine(sum(10, 20));
 add<string> con = Concat;
 Console.WriteLine(concat("Hello ", "World!!"));
 }
 public static int Sum(int val1, int val2)
 {
 return val1 + val2;
 }
 public static string Concat(string str1, string str2)
 {
 return str1 + str2;
 }
}

```

## Summary

- C# is an object-oriented programming language, which combines the power and efficiency of C++, simplicity and object-oriented approach of Java and creativeness of Visual Basic.
- A variable is an identifier that denotes a storage location in memory. It stores numeric or string values that might change during program execution.
- C# supports a rich and varied selection of data types, from built-in types, such as integers or strings, to user-defined types, such as enumerations, structures and classes.
- An array is used to store the number of elements or variables of the same data type at contiguous memory location in an ordered manner.
- An array is much more than a simple modifier, as it would be in C++ or Java. Because arrays are first class objects, just as all other types in are in C#, you can also inspect them for certain attributes.
- In C# programming, string is another kind of data type that represents Unicode Characters. It is the alias of System.String, however, you can also write System.String instead of a string. It is the sequence of character in which each character is a Unicode character.
- Classes allow you to control all the functions that can be applied to a given set of data as well as how to access the data.
- Access modifiers help to avoid jumbling of data and methods with the existing code as well as protect an object of a class from outside interference.

- A constructor is called when an object is first created. A destructor (or finalizer) is called when the object is finally destroyed and the garbage collected.
- To reduce redundancy, the object-oriented languages support inheritance. Inheritance is the property through which a class derives properties from another class.
- An interface is introduced to provide the feature of multiple inheritances to classes.
- Abstraction is the process of hiding the details of a particular object from a user and exposing only the essential features. The concepts of abstraction and encapsulation are complementary to each other.
- Method overloading is a concept in which a method behaves according to the number and types of parameters passes to it.
- Method Overriding is a technique that allows the invoking of functions from another class (base class) in the derived class.
- Delegate is the reference type data type that defines the signature.
- Delegate type variable can refer to any method with the same signature as the delegate.
- Syntax: [access modifier] delegate [return type] [delegate name]([parameters])
- A target method's signature must match with delegate signature.
- Delegates can be invoke like a normal function or Invoke () method.
- Multiple methods can be assigned to the delegate using "+" or "+=" operator and removed using "-" or "-=" operator. It is called multicast delegate.
- If a multicast delegate returns a value then it returns the value from the last assigned target method.
- Delegate is used to declare an event and anonymous methods in C#.

## Practice Questions

---

1. List different data types in C#.
2. What do you mean by value type and reference type?
3. Explain Enumeration type with an Example.
4. Which are built-in reference types available in C#?
5. What is boxing?
6. List properties of an Array.
7. Define Delegates.
8. List and explain string functions of Date Time conversion.
9. What is the use of 'this' keyword?
10. List and explain different access modifiers available in C#.
11. What do you mean by constructor?
12. What is copy constructor?
13. Explained sealed classes and sealed methods.
14. Define Abstract Class.
15. Explain in details the concept of interface with an example.
16. Write a short note on "Method overloading".
17. What is use of virtual keyword?



# Introduction to ASP.NET

## Learning Objectives ...

Students will be able to:

- Explain the three pillars of object oriented programming.
- Develop working knowledge of C# programming constructs and the .NET Framework.
- Write an object oriented program using custom classes, control structures, HTML Events etc.
- Build and debug well-formed Web Forms with ASP.NET Controls.
- Perform form validation with validation controls.
- Create custom controls with user controls.

### 4.1 WHAT IS ASP.NET?

- ASP.NET is a server-side technology used for developing dynamic websites and web applications. ASP.NET aids developers to create web applications by using HTML, CSS, and JavaScript.
- ASP.NET is the latest version of Active Server Pages, which Microsoft developed to build websites. It is a web application framework released in 2002 and had an extension of .aspx.

#### Why ASP.NET?

- ASP.NET reduces all the issues that come up while building a web application like speed, cost, and language.
- ASP.NET provides multiple development modes, which help to develop applications in an easy and better way.
- ASP.NET works on an HTTP protocol and uses HTTP commands.
- ASP.NET provides a platform that allows writing a code in a text editor program and Visual Studio .NET.

- If you are building an application, ASP.NET could be the best choice as it is faster and more efficient than other technologies.
- Components of ASP.NET:**
- ASP.NET depends on three major components:
    - Language:** Language helps in the creation of web applications. Some languages that ASP.NET uses for development are VB.Net and C#.
    - Library:** There are different types of libraries with all the components to help developers and create applications.
    - Common Language Runtime:** It is a platform that helps to execute the .Net programs. It is used for running key activities like exception handling and garbage collection.

## 4.2 ASP.NET PAGE LIFE CYCLE

[W-22]

- ASP.Net page life cycle and how different events are fired during an ASP.NET page life cycle. When an ASP.NET page runs, the page goes through a life cycle in which it performs a series of processing steps. These include initialization, instantiating controls, restoring and maintaining state, running event handler code, and rendering. The following are the various stages or events of ASP.Net page life cycle.

Page life cycle

User Control life cycle

Custom Control life cycle

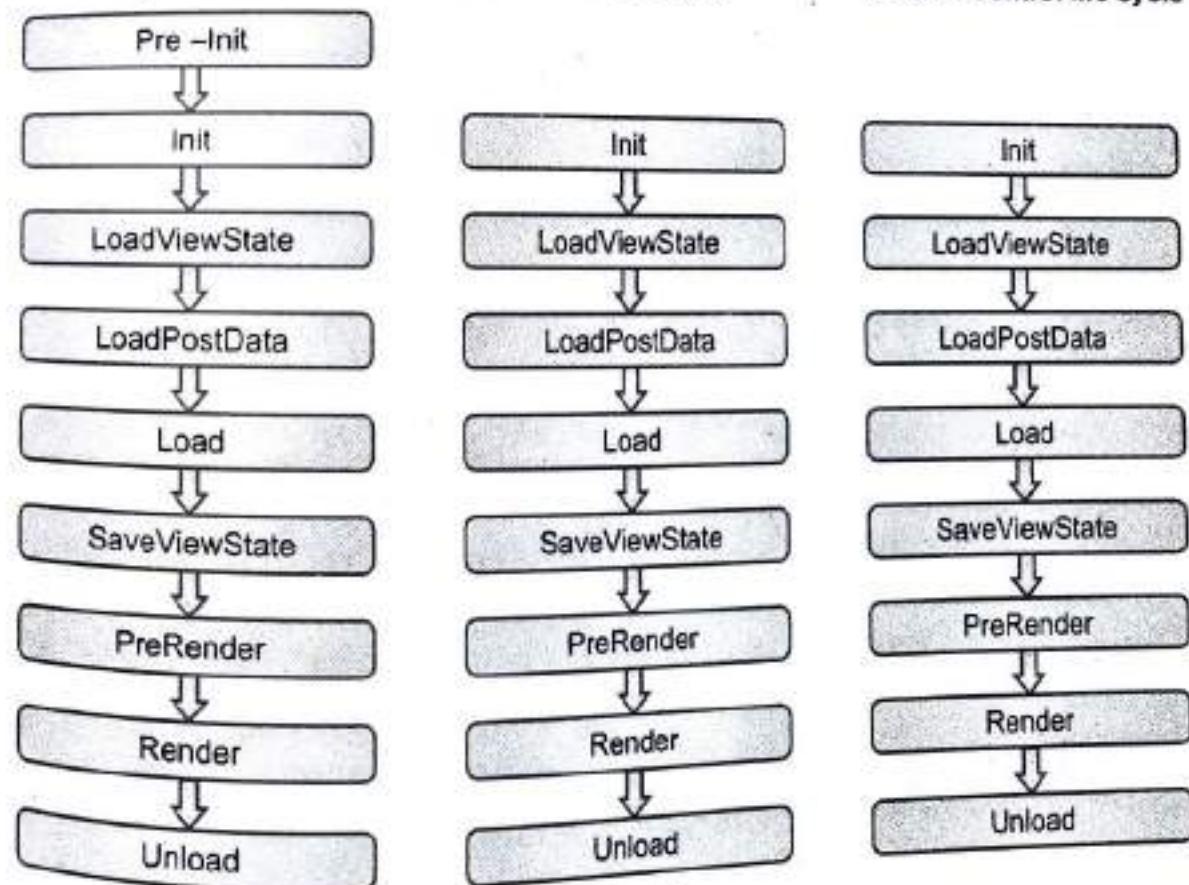


Fig. 4.1

- Pre-Init**
- Check the IsPostBack property to determine whether this is the first time the page is being processed.

- Create or re-create dynamic controls.
- Set a master page dynamically.
- Set the Theme property dynamically.

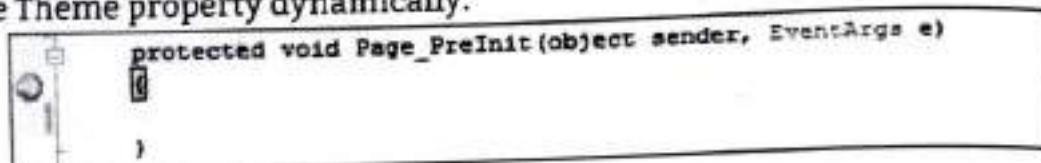


Fig. 4.2

**Note:** If the request is a postback then the values of the controls have not yet been restored from the view state. If you set a control property at this stage, its value might be overwritten in the next event.

#### Init

- This event fires after each control has been initialized.
- Each control's UniqueID is set and any skin settings have been applied.
- Use this event to read or initialize control properties.
- The "Init" event is fired first for the bottom-most control in the hierarchy, and then fired up the hierarchy until it is fired for the page itself.

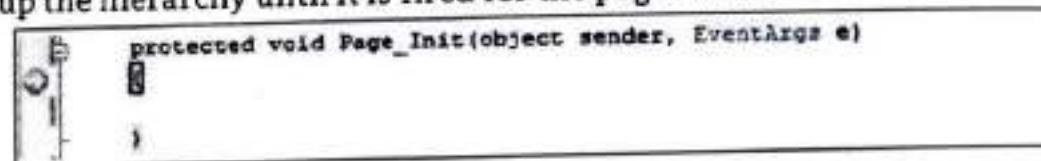


Fig. 4.3

#### InitComplete

- Until now the viewstate values are not yet loaded, hence you can use this event to make changes to the view state that you want to ensure are persisted after the next postback.
- Raised by the Page object.
- Use this event for processing tasks that require all initialization to be complete.

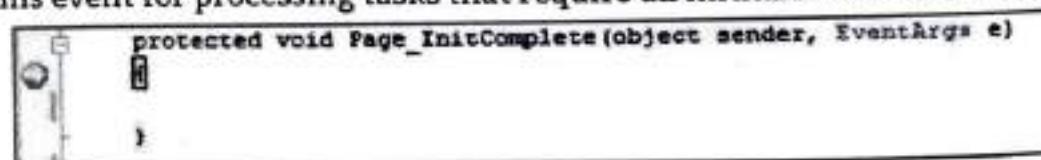


Fig. 4.4

#### OnPreLoad

- Raised after the page loads view state for itself and all controls, and after it processes postback data that is included with the Request instance.
- Before the Page instance raises this event, it loads view state for itself and all controls, and then processes any postback data included with the Request instance.
- **Loads ViewState:** ViewState data are loaded to controls.
- **Loads Postback data:** Postback data are now handed to the page controls

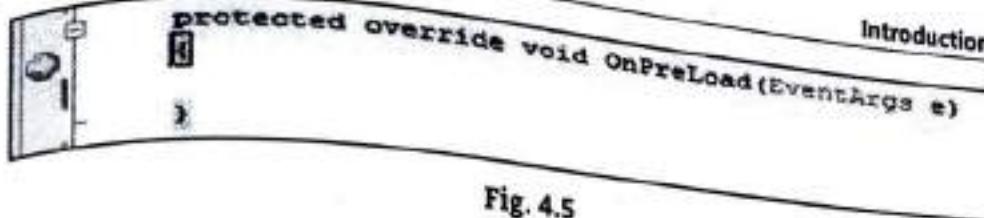


Fig. 4.5

**Load**

- The Page object calls the OnLoad method on the Page object, and then recursively does the same for each child control until the page and all controls are loaded. The Load event of individual controls occurs after the Load event of the page.
- This is the first place in the page life cycle that all values are restored.
- Most code checks the value of IsPostBack to avoid unnecessarily resetting state.
- You may also call Validate and check the value of IsValid in this method.
- You can also create dynamic controls in this method.
- Use the OnLoad event method to set properties in controls and establish database connections.

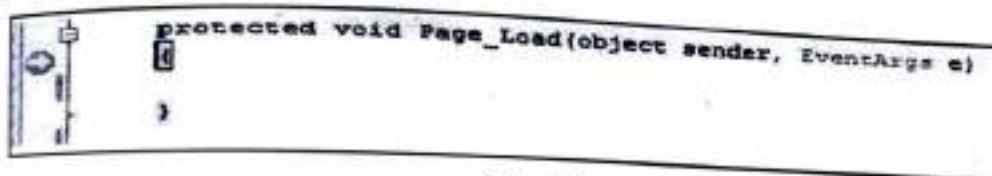


Fig. 4.6

**control PostBack Event(s)**

- ASP.NET now calls any events on the page or its controls that caused the PostBack to occur.
- Use these events to handle specific control events, such as a Button control's Click event or a TextBox control's TextChanged event.
- In a postback request, if the page contains validator controls, check the IsValid property of the Page and of individual validation controls before performing any processing.
- This is just an example of a control event. Here it is the button click event that caused the postback.

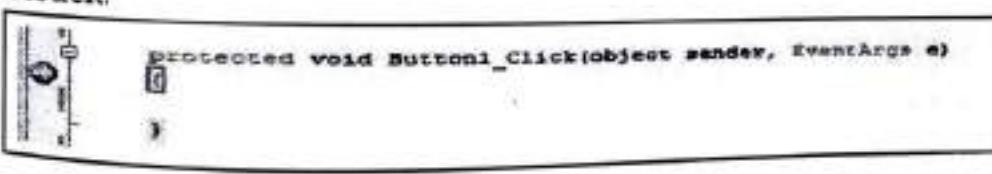


Fig. 4.7

**LoadComplete**

- Raised at the end of the event-handling stage.
- Use this event for tasks that require that all other controls on the page be loaded

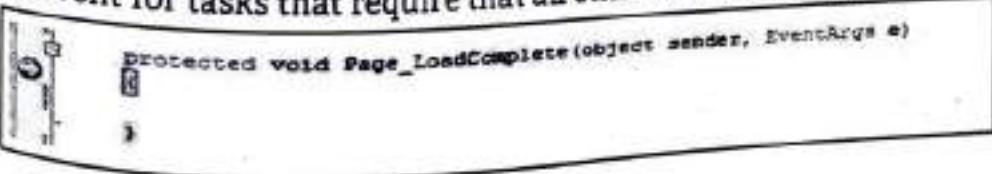


Fig. 4.8

**OnPreRender**

- It is raised after the Page object has created all controls that are required in order to render the page, including child controls of composite controls.
- The Page object raises the PreRender event on the Page object, and then recursively does the same for each child control. The PreRender event of individual controls occurs after the PreRender event of the page.
- The PreRender event of individual controls occurs after the PreRender event of the page.
- Allows final changes to the page or its control.
- This event takes place before saving ViewState, so any changes made here are saved.
- For example: After this event, you cannot change any property of a button or change any viewstate value.
- Each data bound control whose DataSourceID property is set calls its DataBind method.
- Use the event to make final changes to the contents of the page or its controls

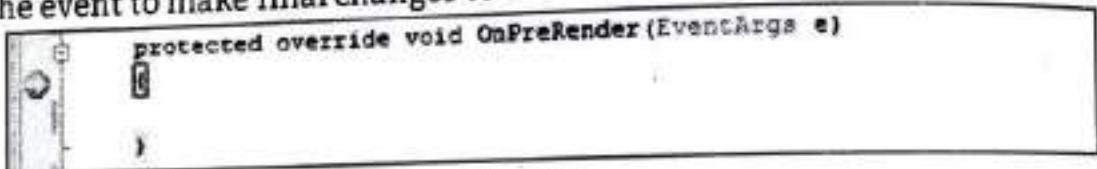


Fig. 4.9

**OnSaveStateComplete**

- It is raised after view state and control state have been saved for the page and for all controls.
- Before this event occurs, ViewState has been saved for the page and for all controls.
- Any changes to the page or controls at this point will be ignored.
- Use this event perform tasks that require the view state to be saved, but that do not make any changes to controls.

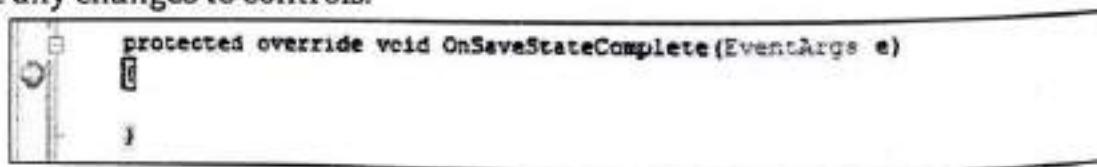


Fig. 4.10

**Render Method**

- This is a method of the page object and its controls (and not an event).
- The Render method generates the client-side HTML, Dynamic Hypertext Markup Language (DHTML), and script that are necessary to properly display a control at the browser.

**UnLoad**

- This event is used for cleanup code.
- At this point, all processing has occurred and it is safe to dispose of any remaining objects, including the Page object.

- Cleanup can be performed on:
  - Instances of classes, in other words objects
  - Closing opened files
  - Closing database connections.
- This event occurs for each control and then for the page.
- During the unload stage, the page and its controls have been rendered, so you cannot make further changes to the response stream.
- If you attempt to call a method such as the Response. Write method then the page will throw an exception.

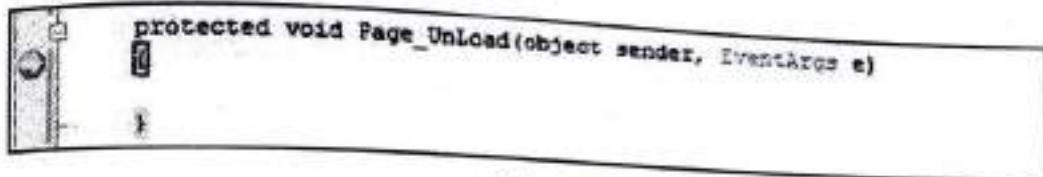


Fig. 4.11

### Examples:

#### Example 1: Control Values.

In the following code, I have assigned the values to the label control on each event. When you run the code, you will see that in the "Page\_UnLoad", the values are not assigned to the label. WHY? Because, during the unload stage, the page and its controls have been rendered, so you cannot change the values.

Please observe the code comments and output. It will help you to clearly understand the concepts.

```
public partial class PageLiftCycle: System.Web.UI.Page
{
 protected void Page_PreInit(object sender, EventArgs e)
 {
 //Work and It will assign the values to label.
 lblName.Text = lblName.Text + "
" + "PreInit";
 }
 protected void Page_Init(object sender, EventArgs e)
 {
 //Work and It will assign the values to label.
 lblName.Text = lblName.Text + "
" + "Init";
 }
 protected void Page_InitComplete(object sender, EventArgs e)
 {
 //Work and It will assign the values to label.
 lblName.Text = lblName.Text + "
" + "InitComplete";
 }
}
```

```
protected override void OnPreLoad(EventArgs e)
{
 //Work and It will assign the values to label.
 //If the page is post back, then label control values
 will be loaded from view state.
 //E.g: If you string str = lblName.Text, then str
 will contain viewstate values.
 lblName.Text = lblName.Text + "
" + "PreLoad";
}

protected void Page_Load(object sender, EventArgs e)
{
 //Work and It will assign the values to label.
 lblName.Text = lblName.Text + "
" + "Load";
}

protected void btnSubmit_Click(object sender, EventArgs e)
{
 //Work and It will assign the values to label.
 lblName.Text = lblName.Text + "
" + "btnSubmit_Click";
}

protected void Page_LoadComplete(object sender, EventArgs e)
{
 //Work and It will assign the values to label.
 lblName.Text = lblName.Text + "
" + "LoadComplete";
}

protected override void OnPreRender(EventArgs e)
{
 //Work and It will assign the values to label.
 lblName.Text = lblName.Text + "
" + "PreRender";
}

protected override void OnSaveStateComplete(EventArgs e)
{
 //Work and It will assign the values to label.
 //But "SaveStateComplete" values will not be available
 during post back. i.e. View state.
 lblName.Text = lblName.Text + "
" + "SaveStateComplete";
}

protected void Page_UnLoad(object sender, EventArgs e)
{
 //Work and it will not effect label contrl, view state
 and post back data.
 lblName.Text = lblName.Text + "
" + "UnLoad";
}
```

PreInit

Init

InitComplete

PreLoad

Load

LoadComplete

PreRender

SaveStateComplete

When you click on the Submit Button output:

PreInit

Init

InitComplete

PreLoad

Load

LoadComplete

PreRender

Load

btnSubmit\_Click

LoadComplete

PreRender

SaveStateComplete

During the first time of the page load with EnableViewState="false":

PreInit

Init

InitComplete

PreLoad

Load

LoadComplete

PreRender

SaveStateComplete

When you click on the Submit Button output with EnableViewState="false"

PreInit

Init

InitComplete

PreLoad

Load

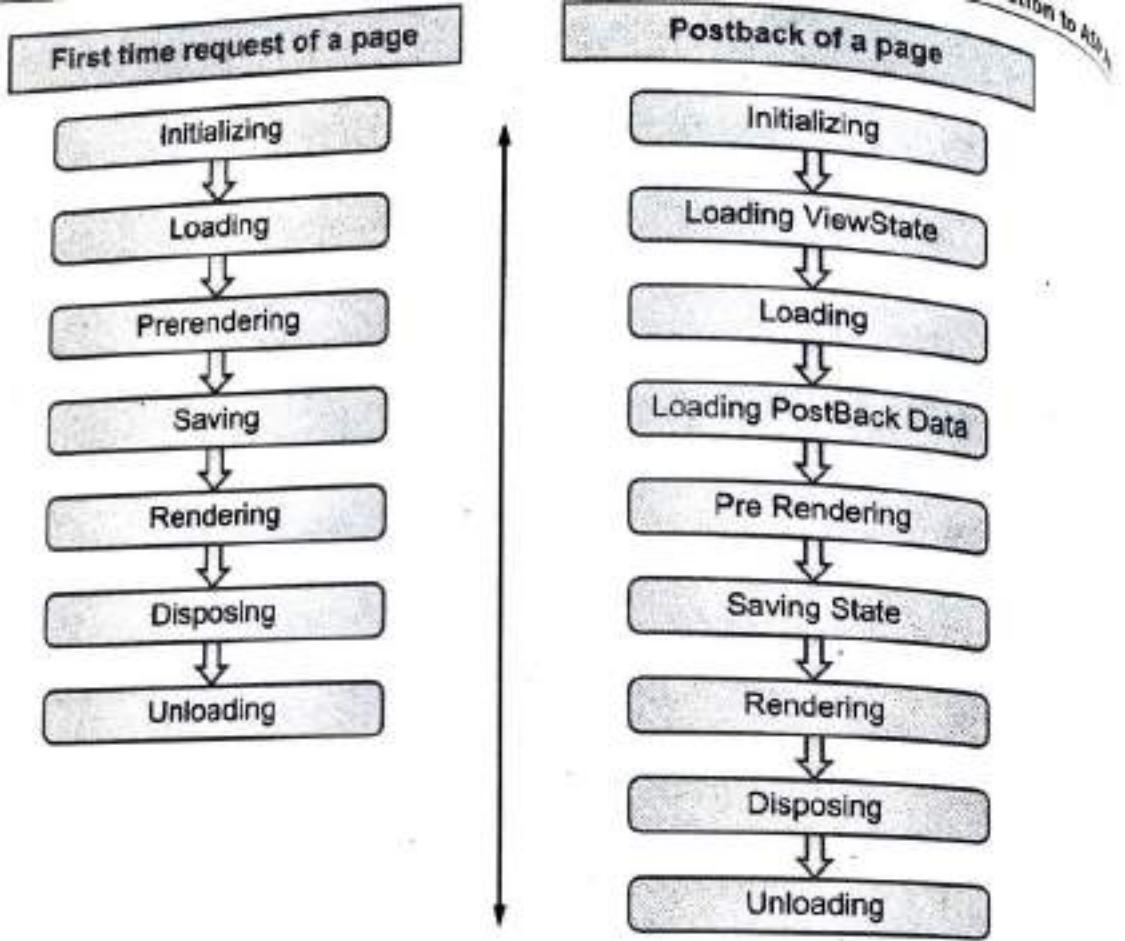
btnSubmit\_Click

LoadComplete

PreRender

SaveStateComplete

SaveStateComplete



**Fig. 4.12**

### Example 2: ViewState Values.

Please observe the code comments and output. It will help you to clearly understand the concepts.

```

public partial class PageLiftCycle : System.Web.UI.Page
{
 protected void Page_PreInit(object sender, EventArgs e)
 {
 //Work and It will assign the values to label.
 //Note : If page is post back or first time call and
 //you have not set any values to ViewState["value"], ""
 //Convert.ToString(ViewState["value"]) is always empty.
 ViewState["value"] = Convert.ToString(ViewState["value"])
 + "
" + "preInit";
 lblName.Text = Convert.ToString(ViewState["value"]);
 }

 protected void Page_Init(object sender, EventArgs e)
 {
 //Work and It will assign the values to label.
 //Note : If page is post back or first time call and you
 //have not set any values to ViewState["value"]
 //previous events, "
 }
}

```

```
//Convert.ToString(ViewState["value"]) is always empty.
ViewState["value"] = Convert.ToString(ViewState["value"])
 + "
" + "Init";
lblName.Text = Convert.ToString(ViewState["value"]);
}
protected void Page_InitComplete(object sender, EventArgs e)
{
 //Work and It will assign the values to label.
 //Note : If page is post back or first time call and you
 have not set any values to ViewState["value"]
 //in previous events, then
 //Convert.ToString(ViewState["value"]) is always empty.
 ViewState["value"] = Convert.ToString(ViewState["value"])
 + "
" + "InitComplete";
 lblName.Text = Convert.ToString(ViewState["value"]);
}
protected override void OnPreLoad(EventArgs e)
{
 //Work and It will assign the values to label.
 //Note : If page is post back and you have set or not
 set any values to ViewState["value"] in previous events, then
 //Convert.ToString(ViewState["value"])
 //will always have post back data.
 //E.g: If you string str = Convert.ToString(ViewState["value"]),
 then str will contain post back values.
 ViewState["value"] = Convert.ToString(ViewState["value"])
 + "
" + "PreLoad";
 lblName.Text = Convert.ToString(ViewState["value"]);
}
protected void Page_Load(object sender, EventArgs e)
{
 //Work and It will assign the values to label.
 ViewState["value"] = Convert.ToString(ViewState["value"])
 + "
" + "Load";
 lblName.Text = Convert.ToString(ViewState["value"]);
}
protected void btnSubmit_Click(object sender, EventArgs e)
{
 //Work and It will assign the values to label.
 ViewState["value"] = Convert.ToString(ViewState["value"])
 + "
" + "btnSubmit_Click";
 lblName.Text = Convert.ToString(ViewState["value"]);
}
```

```
protected void Page_LoadComplete(object sender, EventArgs e)
{
 //Work and It will assign the values to label.
 ViewState["value"] = Convert.ToString(ViewState["value"])
 + "
" + "LoadComplete";
 lblName.Text = Convert.ToString(ViewState["value"]);
}

protected override void OnPreRender(EventArgs e)
{
 //Work and It will assign the values to label.
 ViewState["value"] = Convert.ToString(ViewState["value"])
 + "
" + "PreRender";
 lblName.Text = Convert.ToString(ViewState["value"]);
}

protected override void OnSaveStateComplete(EventArgs e)
{
 //Work and It will assign the values to label.
 //But "SaveStateComplete" values will not be available
 //during post back. i.e. View state
 ViewState["value"] = Convert.ToString(ViewState["value"])
 + "
" + "SaveStateComplete";
 lblName.Text = Convert.ToString(ViewState["value"]);
}

protected void Page_UnLoad(object sender, EventArgs e)
{
 //Work and it will not effect label control values,
 //view state and post back data
 ViewState["value"] = Convert.ToString(ViewState["value"])
 + "
" + "UnLoad";
 lblName.Text = Convert.ToString(ViewState["value"]);
}

}
```

#### Output:

During the first the Time Page Load is output:

PreInit  
Init  
InitComplete  
PreLoad  
Load  
LoadComplete  
PreRender  
SaveStateComplete

When you click on the Submit Button output:

```
preInit
Init
InitComplete
preLoad
Load
LoadComplete
preRender
perLoad
Load
btnSubmit_Click
LoadComplete
preRender
SaveStateComplete
```

During the first time the page loads with EnableViewState="false":

```
preInit
```

```
Init
```

```
InitComplete
```

```
PreLoad
```

```
Load
```

```
LoadComplete
```

```
preRender
```

```
SaveStateComplete
```

When you click on the Submit Button the output with EnableViewState="false":

```
preInit
```

```
Init
```

```
InitComplete
```

```
PreLoad
```

```
Load
```

```
btnSubmit_Click
```

```
LoadComplete
```

```
preRender
```

```
SaveStateComplete
```

**Example 3: ViewState Values.**

Please observe the code comments and output. It will help you to clearly understand the concepts.

```
public partial class PageLifeCycle : System.Web.UI.Page
```

```
{
```

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

```

 {
 Response.Write("
" + "PreInit");
 }
protected void Page_Init(object sender, EventArgs e)
{
 Response.Write("
" + "Init");
}
protected void Page_InitComplete(object sender, EventArgs e)
{
 Response.Write("
" + "InitComplete");
}
protected override void OnPreLoad(EventArgs e)
{
 Response.Write("
" + "PreLoad");
}
protected void Page_Load(object sender, EventArgs e)
{
 Response.Write("
" + "Load");
}
protected void Page_LoadComplete(object sender, EventArgs e)
{
 Response.Write("
" + "LoadComplete");
}
protected override void OnPreRender(EventArgs e)
{
 Response.Write("
" + "PreRender");
}
protected override void OnSaveStateComplete(EventArgs e)
{
 Response.Write("
" + "SaveStateComplete");
}
protected void Page_UnLoad(object sender, EventArgs e)
{
 //Runtime Error : Response is not available in this context.
 //Response.Write("
" + "UnLoad"); //Error
}
}

```

**Output:**

PreInit  
 Init  
 InitComplete  
 PreLoad  
 Load  
 LoadComplete  
 PreRender  
 SaveStateComplete

**Note:** If you write Response.Write("<br/>" + "UnLoad"); in the Page\_UnLoad event then it will generate the Runtime Error "Response is not available in this context".

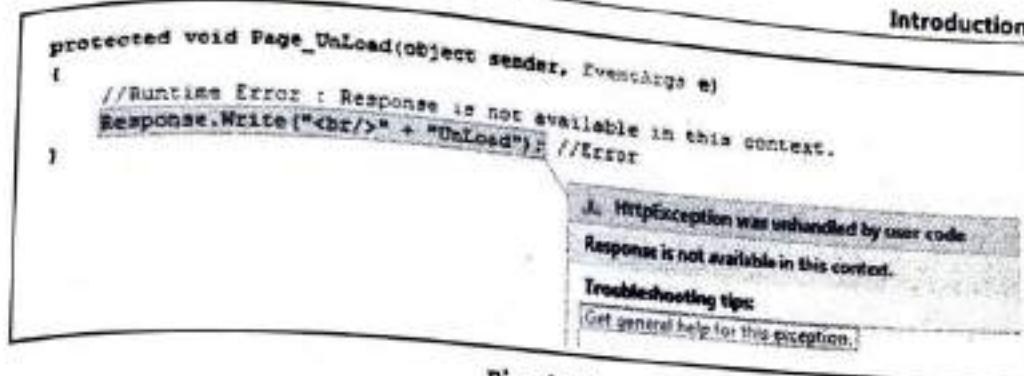


Fig. 4.13

- 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:

### 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.

### 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 things 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 IDs 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 relevant 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 Onload.
- 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.

### Page Life Cycle in ASP.NET

- Let's learn ASP.Net page life cycle and how different events are fired during an ASP.NET page life cycle. When an ASP.NET page runs, the page goes through a life cycle in which it performs a series of processing steps. These include initialization, instantiating controls, restoring and maintaining state, running event handler code, and rendering. The following are the various stages or events of ASP.Net page life cycle.

### 4.3 ASP.NET ARCHITECTURE

[S-23]

- ASP.Net is a web development platform provided by Microsoft. It is used for creating web-based applications. ASP.Net was first released in the year 2002.
- The first version of ASP.Net deployed was 1.0. The most recent version of ASP.Net is version 4.6. ASP.Net is designed to work with the HTTP protocol. This is the standard protocol used across all web applications.
- ASP.Net applications can also be written in a variety of .Net languages. These include C#, VB.Net, and J#. In this chapter, you will see some basic fundamental of the .Net framework.
- The full form of ASP is Active Server Pages, and .NET is Network Enabling Technologies.

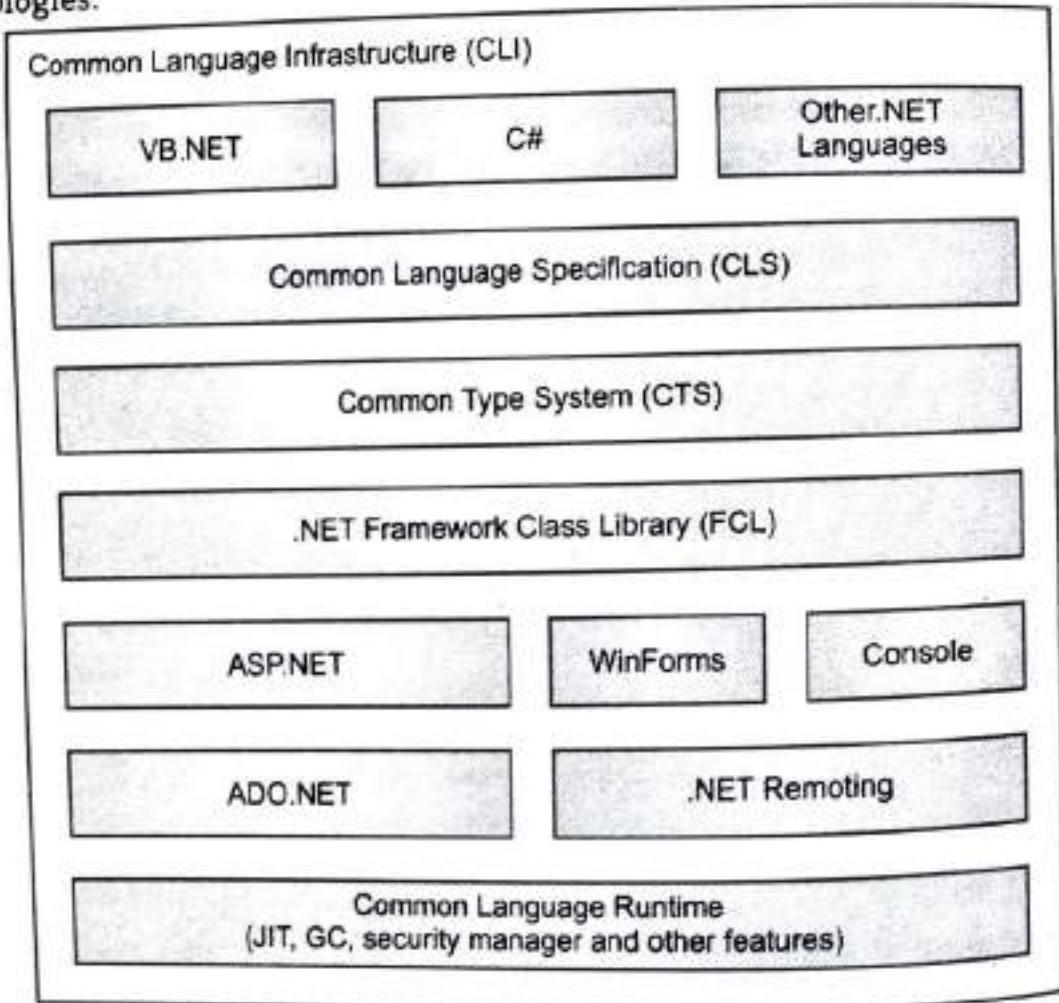
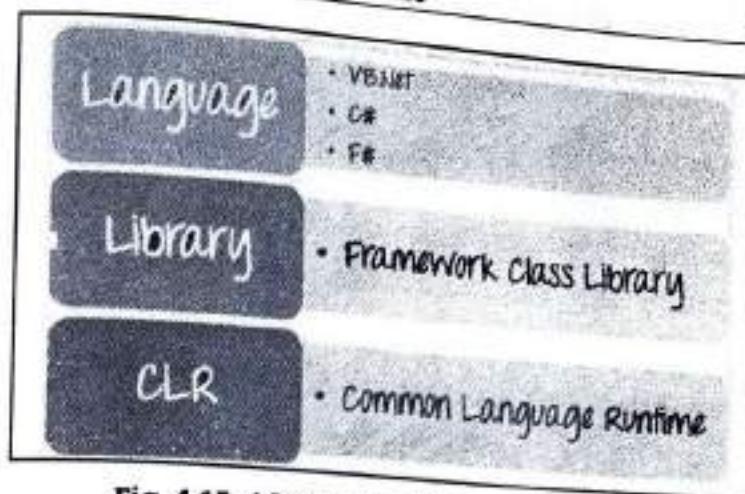


Fig. 4.14

#### ASP.NET Architecture and its Components

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



**Fig. 4.15: 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.
- Below are some of the key characteristics of the ASP.Net framework:
  1. **Code Behind Mode:** This is the concept of separation of design and code. By making this separation, it becomes easier to maintain the ASP.Net application. The general file type of an ASP.Net file is aspx. Assume we have a web page called MyPage.aspx. There will be another file called MyPage.aspx.cs which would denote the code part of the page. So Visual Studio creates separate files for each web page, one for the design part and the other for the code.
  2. **State Management:** ASP.Net has the facility to control state management. HTTP is known as a stateless protocol. Let's take an example of a shopping cart application. Now, when a user decides what he wants to buy from the site, he will press the submit button. The application needs to remember the items the user chooses for the purchase. This is known as remembering the state of an application at a current point in time. HTTP is a stateless protocol. When the user goes to the purchase page, HTTP will not store the information on the cart items. Additional coding needs to be done to ensure that the cart items can be carried forward to the purchase pages. Such an implementation can become complex at times. But ASP.Net can do state management on your behalf. So ASP.Net can remember the cart items and pass it over to the purchase page.

3. **Caching:** ASP.Net can implement the concept of Caching. This improves the performance of the application. By caching those pages which are often requested by the user can be stored in a temporary location. These pages can be retrieved faster and better responses can be sent to the user. So caching can significantly improve the performance of an application.
- ASP.Net is a development language used for constructing web-based applications. ASP.Net is designed to work with the standard HTTP protocol.

#### 4.4 ASP.NET WEB PAGES - HTML 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.

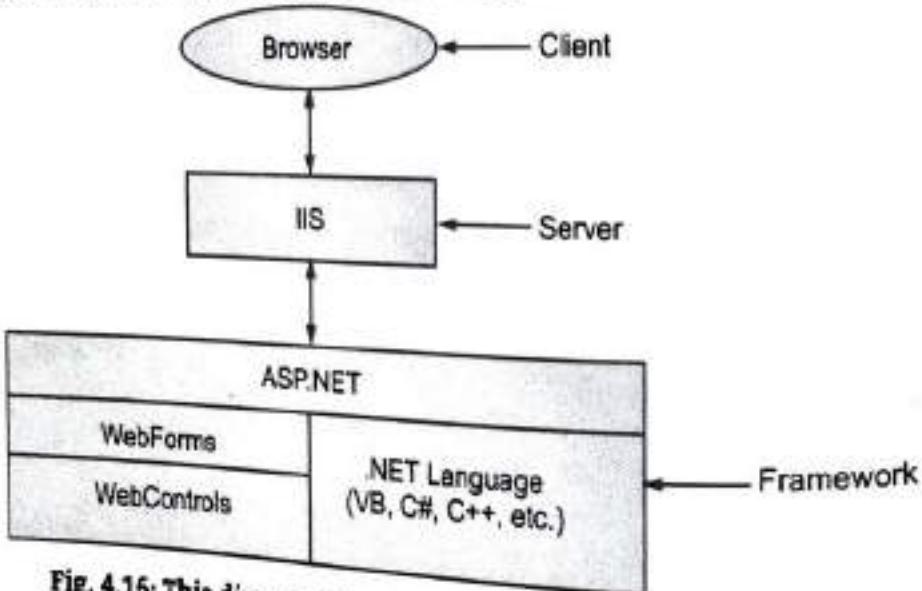


Fig. 4.16: This diagram shows the components of the ASP.NET

- 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:
- 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      | SelectedIndexChanged                                                                                                                                                        | It is used to create a ListBox control like the HTML control.                                                                                                                   |
| GridView     | CancelCommand,<br>EditCommand, DeleteCommand,<br>ItemCommand,<br>SelectedIndexChanged,<br>PageIndexChanged,<br>SortCommand,<br>UpdateCommand, ItemCreated,<br>ItemDataBound | It used to create a grid that is used to show data. We can also perform paging, sorting, and formatting very easily with this control.                                          |
| DataList     | CancelCommand,<br>EditCommand, DeleteCommand,<br>ItemCommand,<br>SelectedIndexChanged,<br>UpdateCommand, ItemCreated,<br>ItemDataBound                                      | It is used to create datalist that is non-tabular and used to show data.                                                                                                        |
| Repeater     | ItemCommand, ItemCreated,<br>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. |

contd...

| Dot Net Framework [BBA (C.A) : Sem. VI] |                                                        | 4.21 | Introduction to ASP.NET                                                                                                             |
|-----------------------------------------|--------------------------------------------------------|------|-------------------------------------------------------------------------------------------------------------------------------------|
| 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,<br>VisibleMonthChanged,<br>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. |

### HTML Controls

[S-22, W-22]

- 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 |

contd...

|                 |                                                                                                                                                                                                                                    |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 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                                                                                                                                                                                    |

## 4.5 REQUEST CLASS IN ASP.NET

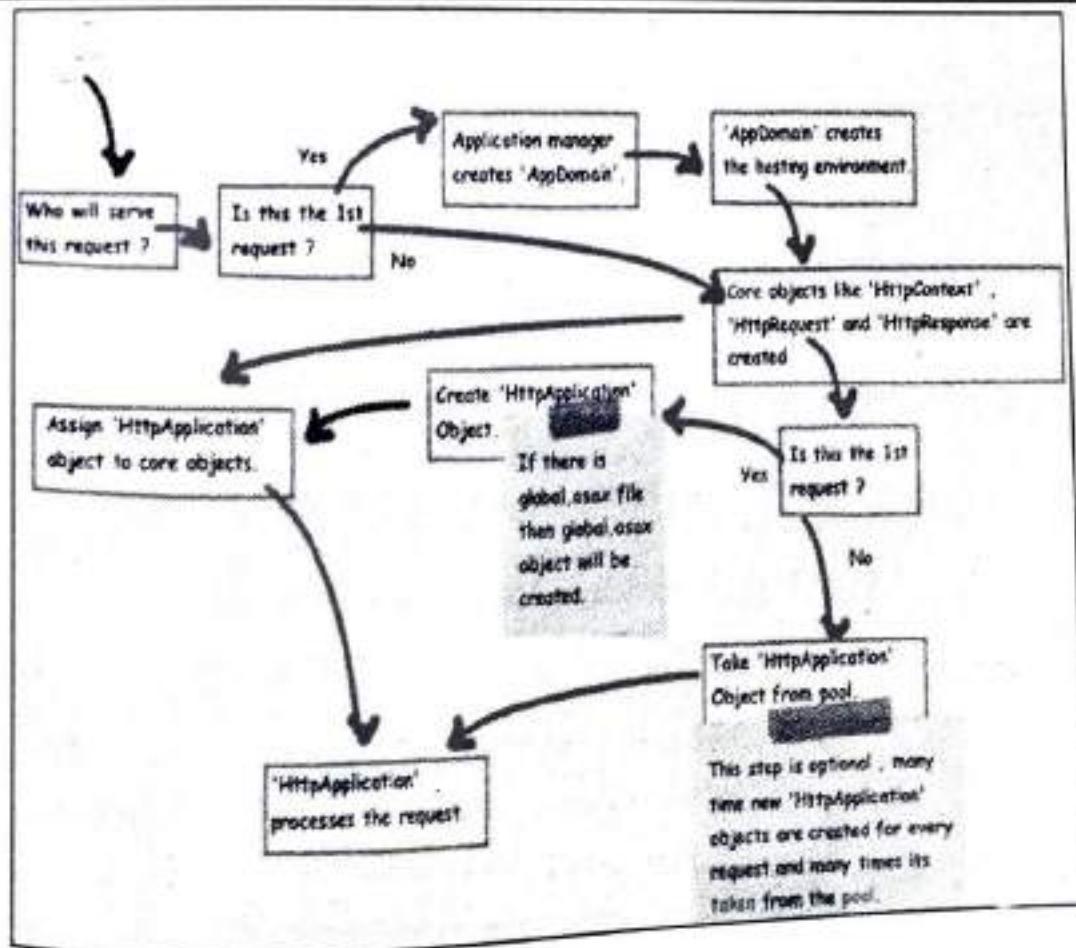


Fig. 4.17

- The `HttpRequest` class is present in the `System.Web` namespace. During a web request, the `HttpRequest` class enables ASP.NET to read the HTTP values sent by a client. The method of this class is used to get the path information and save requests on the disk.

### Properties and Methods of `HttpRequest` Class

- Following are some of the important properties of the `HttpRequest` class

#### Path:

- This property retrieves the virtual path of the current request.

```
public string Path { get; }
```

- Code retrieves the path of the requested URL.

```
Response.Write("Path is :" + Request.Path);
```

Here `Request.Path` retrieves the path of the current URL request. The method `Response.Write()` displays the retrieved path.

#### PathInfo:

- This property retrieves the additional path information of the current request. It retrieves the information only for resources that have a URL extension.

Syntax is,

```
public string PathInfo { get; }
```

#### Source code:

```
if (Request.PathInfo == String.Empty)
{
 Response.Write("PathInfo property has no info...");}
} else
{
 Response.Write("
 PathInfo contains:
 " + Request.PathInfo);
}
```

Here `String.Empty` represents an empty string. If condition check whether `PathInfo` is empty. If there is no path information of the current request, `Response.Write()` method displays a message stating that the `PathInfo` property contains no information. If path information is available, the `Request.PathInfo` retrieves the path and `Response.Write()` method displays it on the web page.

#### UserAgent:

- This property retrieves the user agent string of the client browser. Consider the following example of a user agent string returned by the `UserAgent` property.
- Mozilla/5.0(Windows;U;Windows NT 5.1; en-IN; rv:1.7.2) Gecko/20050915 Firefox/2.0.6
- In this example, the `UserAgent` string indicates which browser the user is using, its version number, and details about the system such as operating system and version.

The web server can use this information to provide content that is defined for a specific browser. Also identifies the user's browser and provides certain system details to servers.

The syntax for using the `UserAgent` property is,

```
public string UserAgent { get; }
```

For example,

```
Response.Write("User Agent is :" +Request.UserAgent);
```

The code `UserAgent` property retrieves the user agent string of the client browser for the current URL request. This string is displayed on the web page using the `Response.Write()` method.

#### **UserHostAddress:**

- This property retrieves the host Internet Protocol (IP) address of the remote client who makes the request.
- The IP address is a unique address used to identify a computer on a network. This address helps computers on the network to communicate with each other. 102.169.2.1 is an example of an IP address. The property `UserHostAddress` is used to retrieve the IP address of the client computer.
- The syntax for using the `UserHostAddress` property is,

```
public string UserHostAddress { get; }
```

For example,

```
Response.Write("User Host Address is :" +Request.UserHostAddress);
```

- The code `Request.UserHostAddress` retrieves the host IP address of the remote client making the current URL request. This IP host address is displayed on web page using `Response.Write()` method.

#### **TotalBytes:**

- This property retrieves the number of bytes in the current input stream. The syntax for using the `TotalBytes` property is,

```
public int TotalBytes {get; }
```

- Code below demonstrate the use of the `TotalBytes` property.

```
if (Request.TotalBytes > 1000)
{
 Response.Write("The request is 1 KB or greater");
}
```

- The if condition checks if the total number of bytes in the input stream is greater than 1000. If the condition evaluates to true, `Response.Write()` method is displays the message "The request is 1KB or greater" on the web page.

Following are some of the important methods of the `HttpRequest` class.

**BinaryRead():**

- This method is used to retrieve the data sent to the server from the client.
- The syntax is,

```
public byte[] BinaryRead(int count)
```

where count represents the number of bytes to be read. Code shows the use of the BinaryRead() method.

```
int intTotalBytes=Request.TotalBytes;
byte[] bCount=Request.BinaryRead(intTotalBytes);
```

- In this code, intTotalBytes is declared as an integer variable that stores the total number of bytes in the current input stream. This number is retrieved using Request.TotalBytes. This value is then used by the BinaryRead() method to perform a binary read of the input stream and store the data that is read into a byte array named bCount.

**GetType():**

- This method retrieves the type of the current instance.

**Syntax:**

```
public Type GetType()
```

Where, Type is the class that represents various type declarations such as class types, array types, and interface types.

For example,

```
Response.Write("
Type:" +Request.GetType());
```

- The GetType() method retrieves the Type of the instance and it is displayed using the Response.Write() method.

**Expectations of HttpRequest Class**

- Exceptions are runtime errors that occur due to certain problems encountered at the time of execution of code. When working with ASP.NET using C#, exception handling is achieved using the try-catch-finally construct. The try block encloses the statements that might throw an exception and the catch block handles the exception. The code is finally block is always executed, regardless of whether an exception occurred or not, and this code contains cleanup routines for exception situations.
- When calling the BinaryRead() method of the HttpRequest class, an exception of type ArgumentException can possibly occur. This exception occurs when calling a method, if at least one of the arguments passed to the method does not meet the parameter specification.
- How to get the complete information on HTTP Request, HTTP Response and Server using asp.net c#.

**Step 1 : Used Namespace**

```
using System.Web;
```

**Step 2 :** Helps to get Complete information about the server.

```
public static HttpServerUtility GetCurrentServerInformation()
{
 return HttpContext.Current.Server;
}
```

**Step 3 :** Helps to get Complete infomation about the HttpRequest.

```
public static HttpRequest GetCurrentHttpRequestInformation()
{
 return HttpContext.Current.Request;
}
```

**Step 4 :** Helps to get Complete infomation about the HttpResponse.

```
public static HttpResponse GetCurrentHttpResponseInformation()
{
 return HttpContext.Current.Response;
}
```

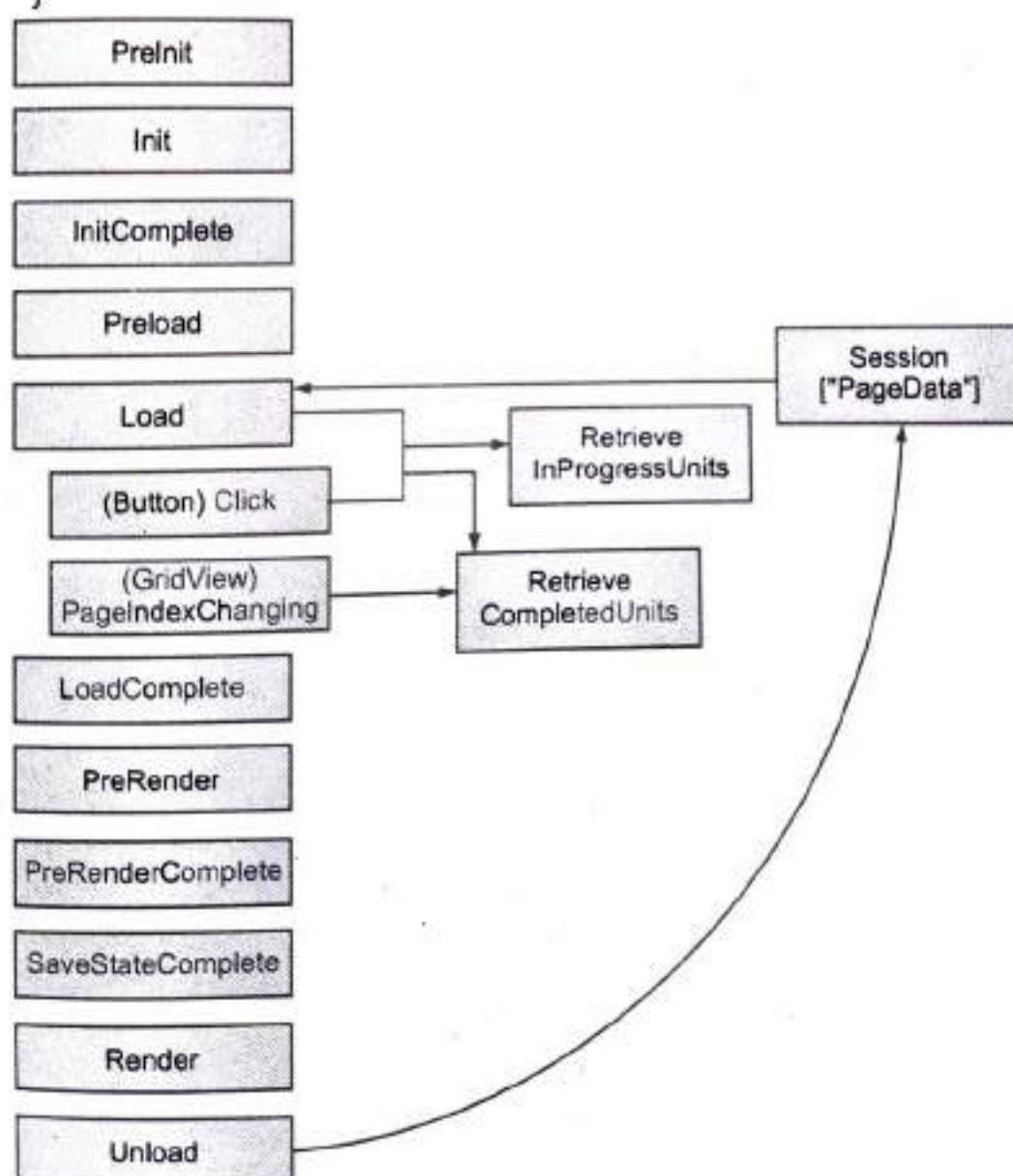


Fig. 4.18

## HTTP Response:

- **HTTPResponseMessage** feature is provided by WebAPI framework which is used to create HTTP services. It helps us to send responses in different ways. **HTTPResponseMessage** is used to return data as well as some user friendly messages like:

| Status Code | Status Message |
|-------------|----------------|
| 200         | OK             |
| 201         | Created        |
| 404         | Not Found      |

## HTTPResponseMessage in Web API

- Now let's see step by step implementation of **HTTPResponseMessage**:

  1. Open Visual Studio Editor.
  2. Select 'File' menu, expand 'New' and click on 'Project...'.

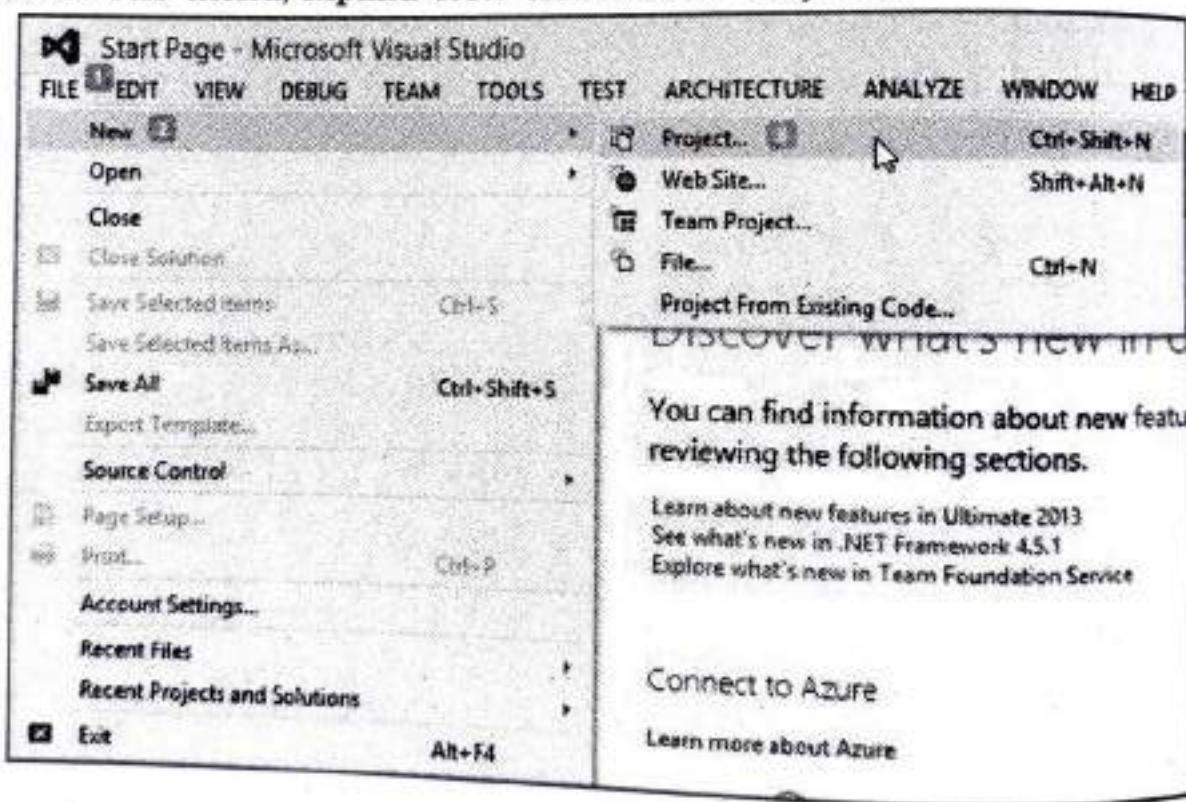


Fig. 4.19

- Expand 'Visual C#' and select 'Web' from the left panel.
- Select 'ASP.NET Web Application'.
- Provide appropriate name of the application and select the location.
- Click on 'OK' button.

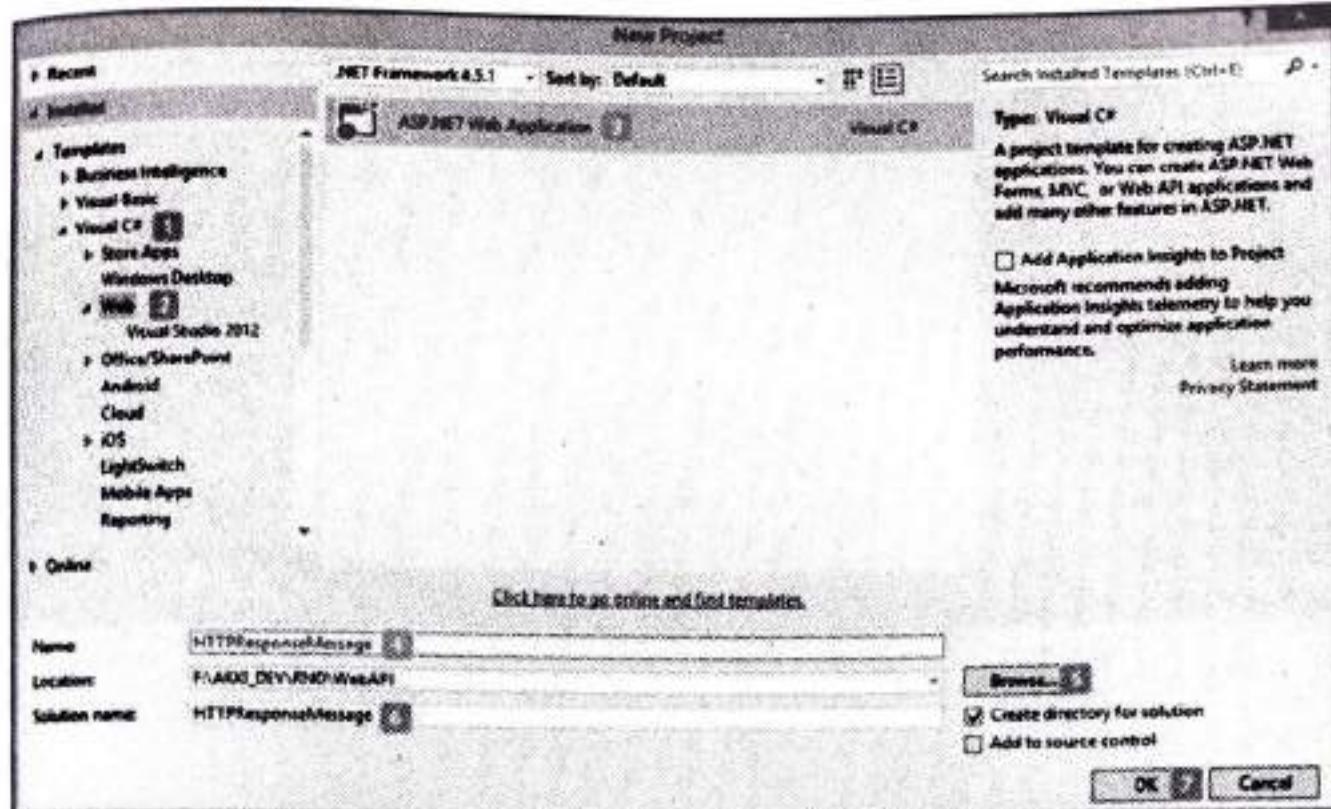


Fig. 4.20

- Select 'Empty' as a template and check 'Web API' check box.
- Press 'OK' button and it will create an empty Web API project.

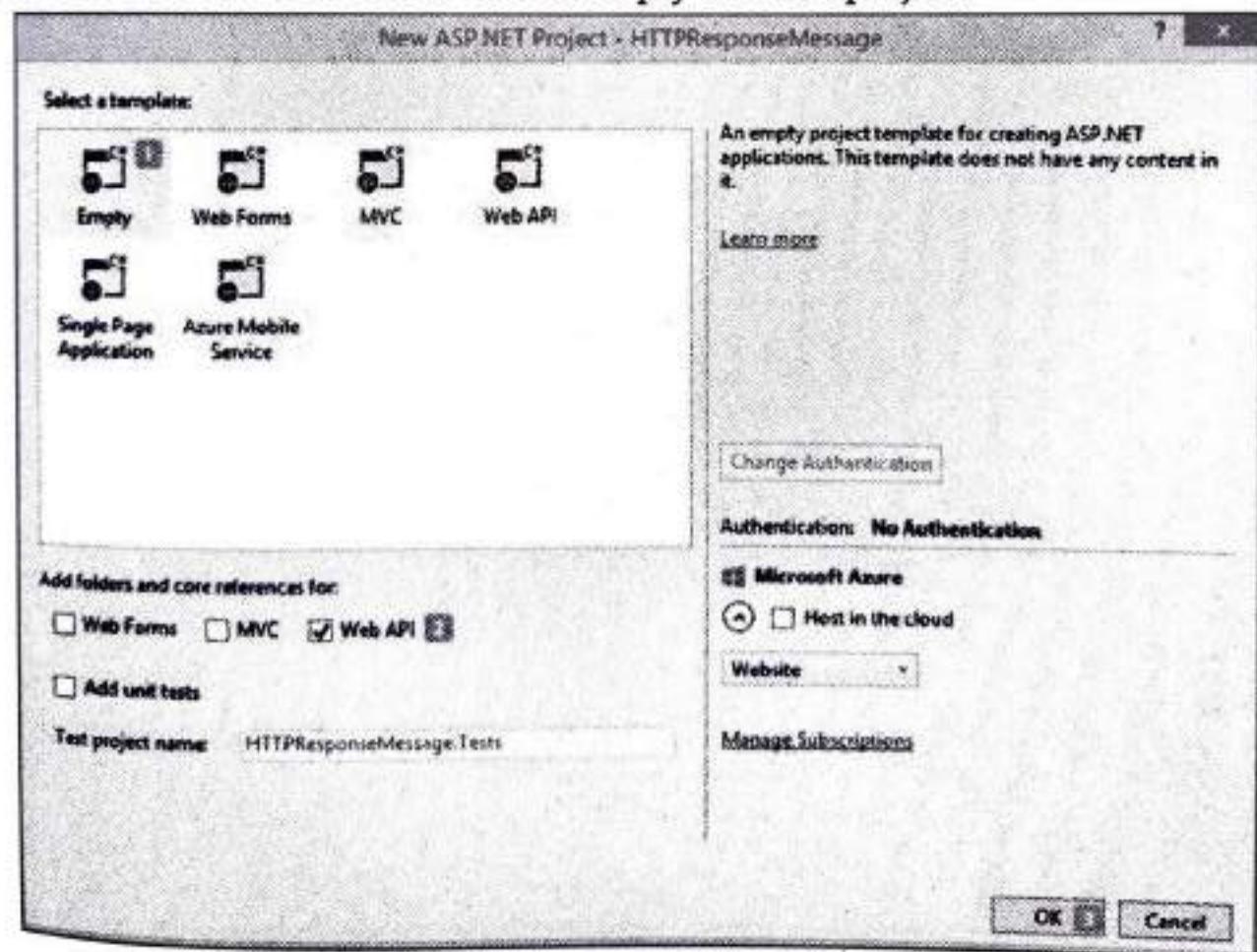


Fig. 4.21

Right click on 'Controllers', expand 'Add' and click on 'Controller...'.

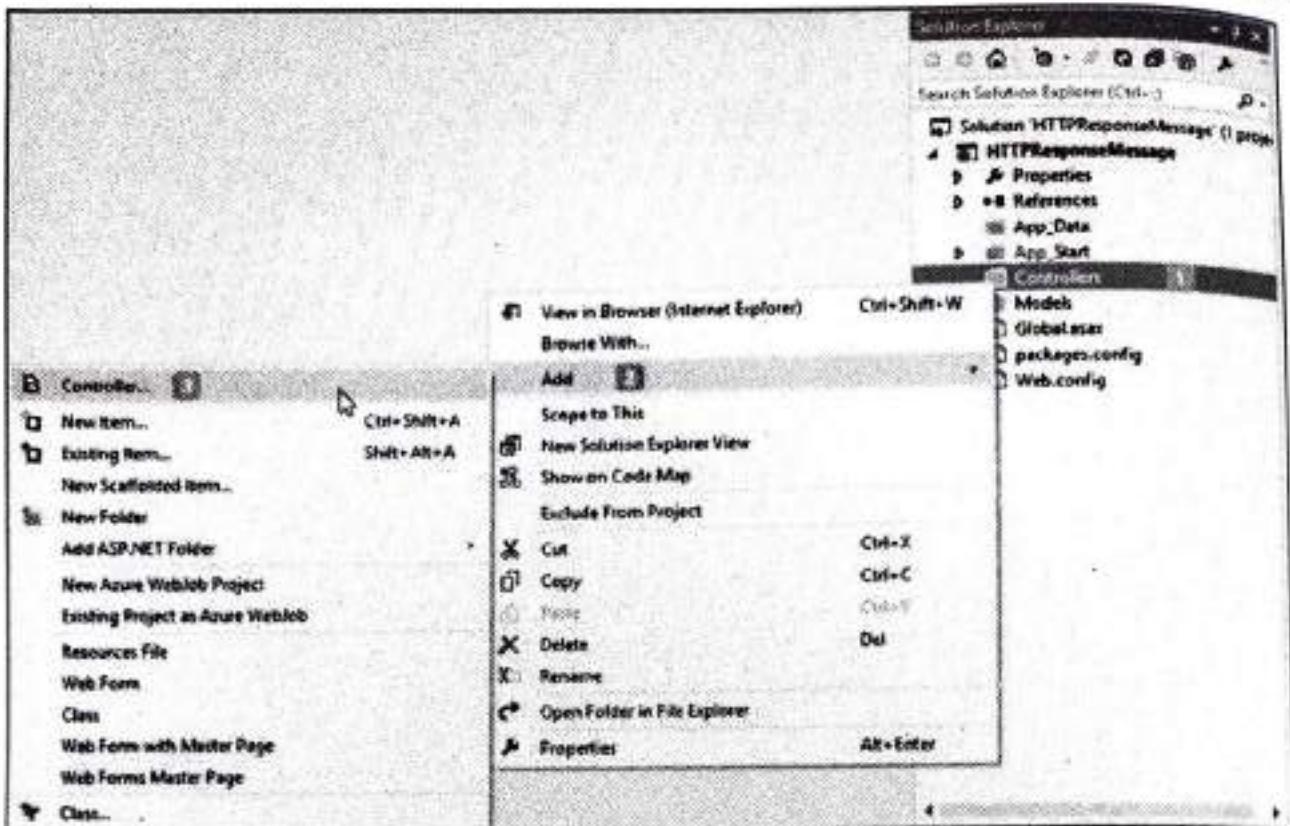


Fig. 4.22

- Select 'Web API 2 Controller with read/write actions' and press 'Add' button, which will add all HTTP methods.

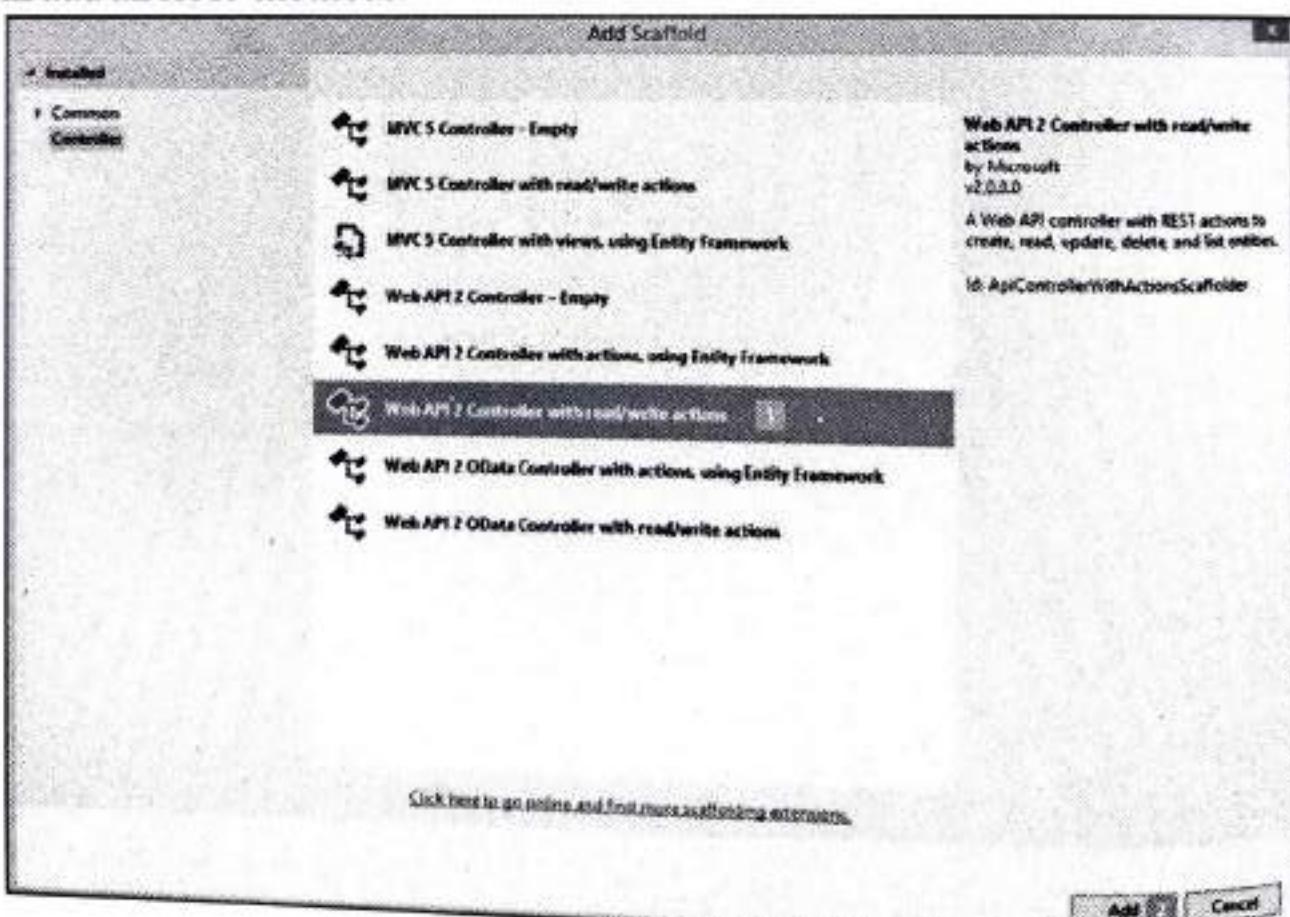


Fig. 4.23

- Provide appropriate name and press 'Add' button.

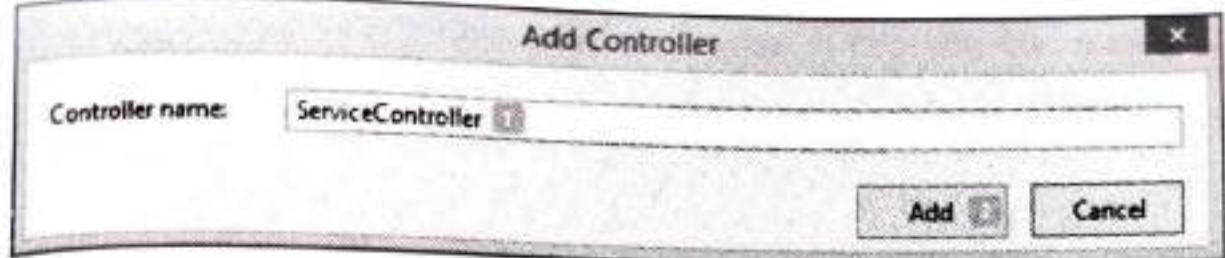


Fig. 4.24

Controller is created with read / write actions.

namespace HttpResponseMessage.Controllers

{

public class ServiceController: ApiController

{

// GET: api/Service

public IEnumerable < string > Get()

{

return new string[]

{

"value1",

"value2"

};

}

// GET: api/Service/5

public string Get(int id)

{

return "value";

}

// POST: api/Service

public void Post([FromBody] string value) {}

// PUT: api/Service/5

public void Put(int id, [FromBody] string value) {}

// DELETE: api/Service/5

public void Delete(int id) {}

}

}

Set HttpResponseMessage as return type for all API methods.

Here we are using Request.CreateErrorResponse to create an HttpResponseMessage.

namespace HttpResponseMessage.Controllers

{

public class ServiceController: ApiController

{

static List < string > serviceData = LoadService();

```
public static List < string > LoadService()
{
 return new List < string > ()
 {
 "Mobile Recharge",
 "Bill Payment"
 };
}

// GET: api/Service
public HttpResponseMessage Get()
{
 return Request.CreateResponse < IEnumerable < string >>
 (HttpStatusCode.OK, serviceData);
}

// GET: api/Service/5
public HttpResponseMessage Get(int id)
{
 if (serviceData.Count > id) return Request.
 CreateResponse < string > (HttpStatusCode.OK, serviceData[id]);
 else return Request.CreateErrorResponse
 (HttpStatusCode.NotFound, "Item Not Found");
}

// POST: api/Service
public HttpResponseMessage Post([FromBody] string value)
{
 serviceData.Add(value);
 return Request.CreateResponse(HttpStatusCode.Created,
 "Item Added Successfully");
}

// PUT: api/Service/5
public HttpResponseMessage Put(int id, [FromBody] string value)
{
 serviceData[id] = value;
 return Request.CreateResponse(HttpStatusCode.OK,
 "Item Updated Successfully");
}

// DELETE: api/Service/5
```

```
public HttpResponseMessage Delete(int id)
{
 serviceData.RemoveAt(id);
 return Request.CreateResponse(HttpStatusCode.OK,
 "Item Deleted Successfully");
}
```

- Run your application and follow below steps:

**GET:**

- Select 'GET' as Method.
- Copy URL and press 'SEND' button.
- In the response header, you can see the http status code and message '200 OK'.
- Observe the response we can see two string elements are added in the list.

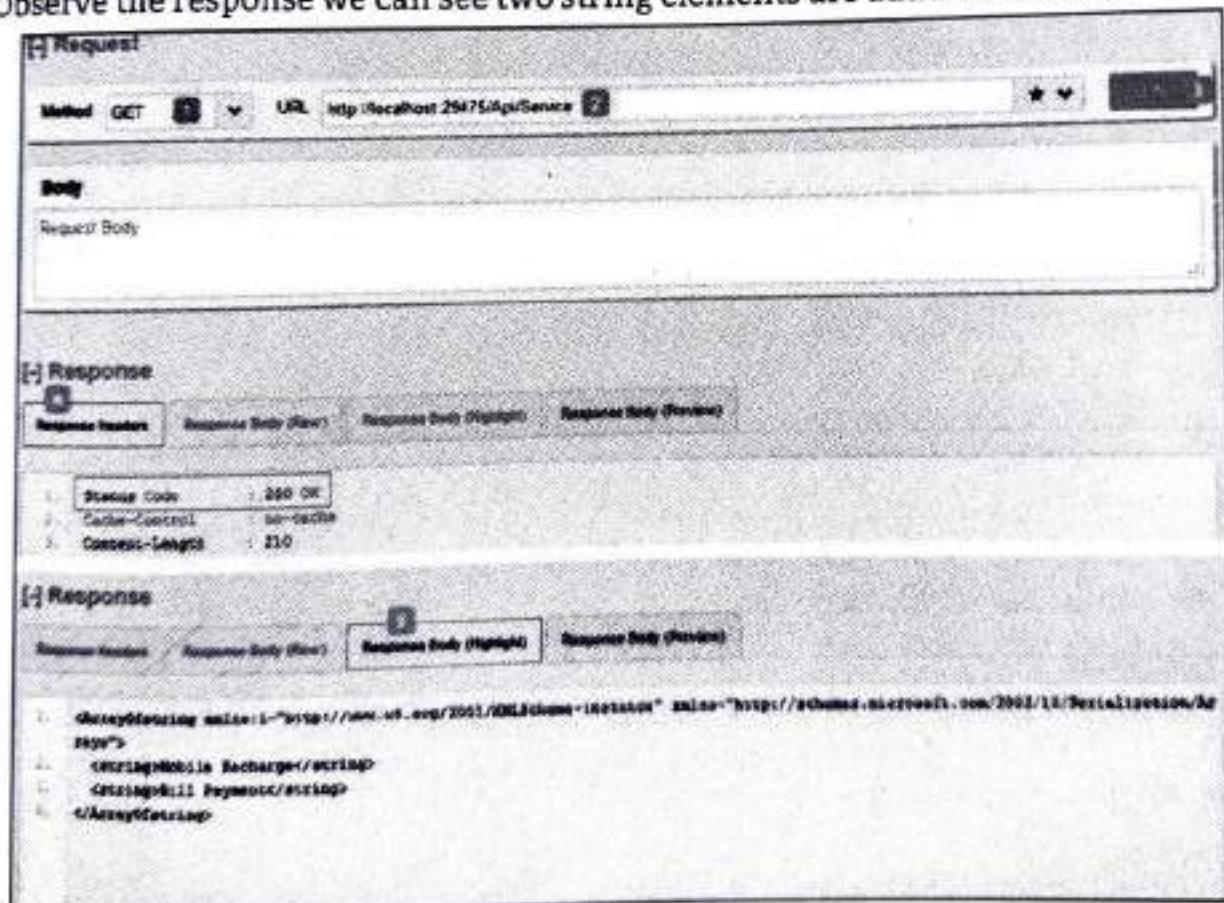


Fig. 4.25

**GET By ID**

- In order to get the individual string element from the list, pass index in the URL.
- In the response header, you can see the http status code and message '200 OK'.
- Check the response, you will have individual element associated with the index.

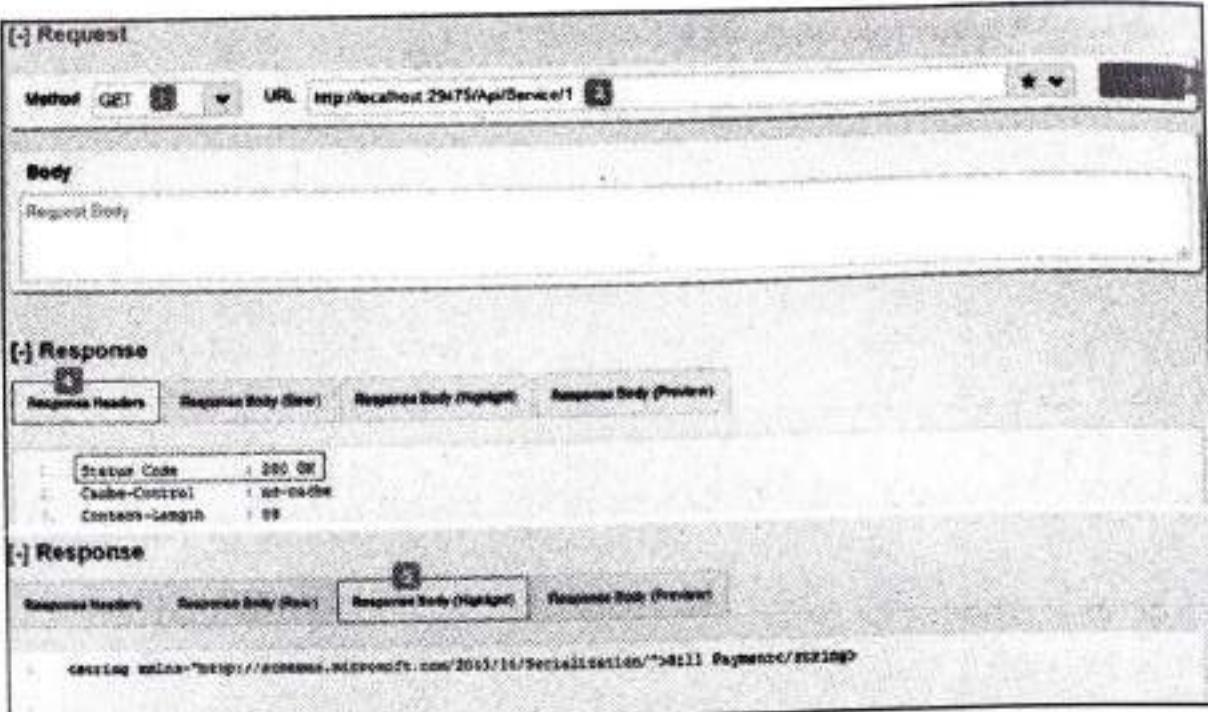


Fig. 4.26

### POST:

- Select 'POST' as method.
- Add Content-Type header with the value 'application/json'.
- Add the value which you want to add in the list in Body and press 'SEND' button.
- In the response header, you can see the http status code and message '201 Created'.

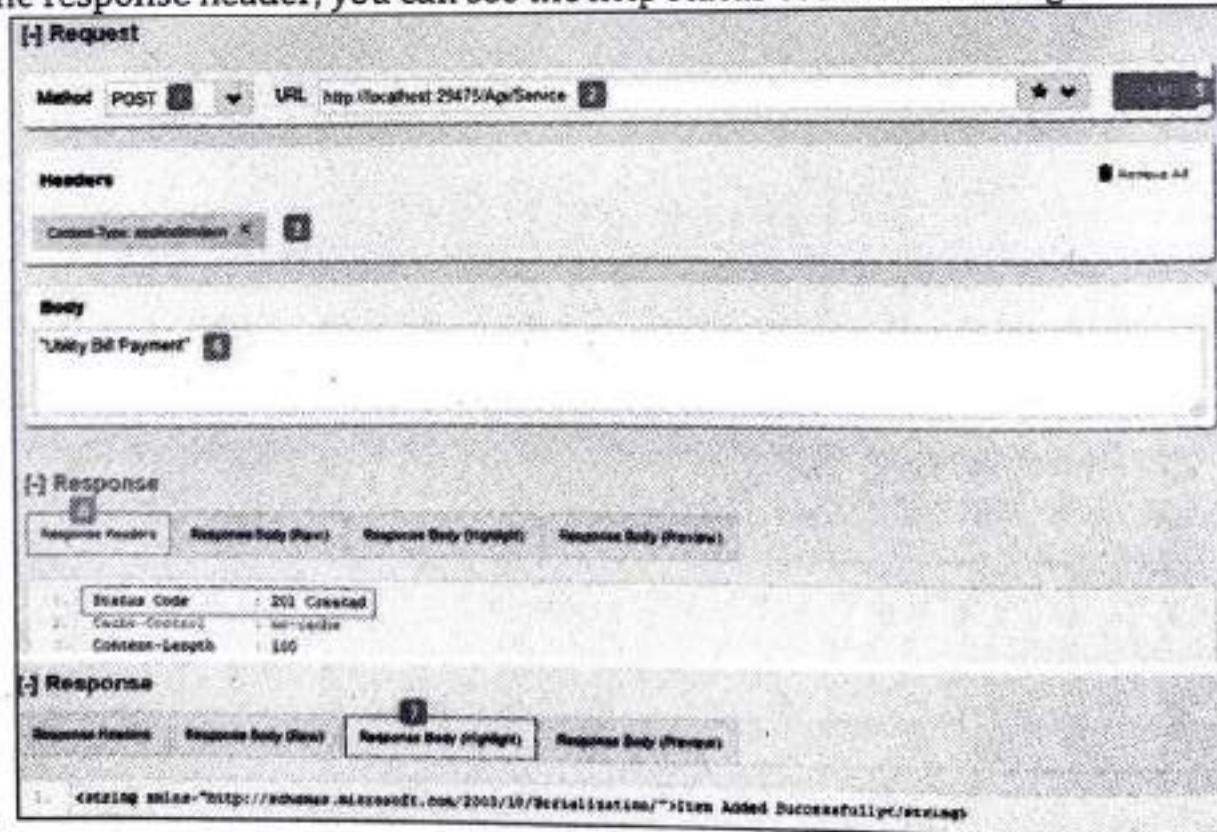


Fig. 4.27

### PUT:

- Select 'PUT' as method and add index for which you want to modify in the URL.
- Add Content-Type header with value 'application/json'.

- Add updated value in the body and press 'SEND' button.

- In the response header, you can see the http status code and message '200 OK'.

**[+] Request**

Method: PUT URL: http://localhost:29475/Api/Service/2

Headers:

```
Content-Type: application/json
```

Body:

```
"Utility Payment"
```

---

**[+] Response**

Response Headers:

|                |            |
|----------------|------------|
| Status Code    | : 200 OK   |
| Cache-Control  | : no-cache |
| Content-Length | : 162      |

Response Body (Raw):

```
<string value="http://schemas.microsoft.com/2003/10/Serialization/">Item Updated Successfully</string>
```

Response Body (Highlight):

Response Body (Preview):

Fig. 4.28

### DELETE:

- Select 'DELETE' as method and add index for which you want to delete from the list in URL.
- Press 'SEND' button.
- In the response header, you can see the http status code and message '200 OK'.

**[+] Request**

Method: DELETE URL: http://localhost:29475/Api/Service/2

Body:

```
Request Body:
```

Buttons:

---

**[+] Response**

Response Headers:

|                |            |
|----------------|------------|
| Status Code    | : 200 OK   |
| Cache-Control  | : no-cache |
| Content-Length | : 162      |

Response Body (Raw):

```
<string value="http://schemas.microsoft.com/2003/10/Serialization/">Item Deleted Successfully</string>
```

Response Body (Highlight):

Response Body (Preview):

Fig. 4.29

## 4.6 USING ASP.NET SERVER SIDE CONTROLS

[W-22]

- 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:
  - Session
  - Application
  - Cache
  - Request
  - Response
  - Server
  - User
  - Trace
- 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:

[S-22]

- 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:

[S-23]

- 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. |

contd...

|                          |                                                                                                                                                                |
|--------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 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. |

contd...

|                    |                                                                                                    |
|--------------------|----------------------------------------------------------------------------------------------------|
| 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.) |

contd...

|                     |                                                                                                                |
|---------------------|----------------------------------------------------------------------------------------------------------------|
| 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.                           |

contd...

|                   |                                                                              |
|-------------------|------------------------------------------------------------------------------|
| 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 4:

- 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:

```


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

<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) +
 ".
 The url is " + Server.UrlEncode(Request.Url.ToString())
 }
}
```

- Run the page to see the following result:

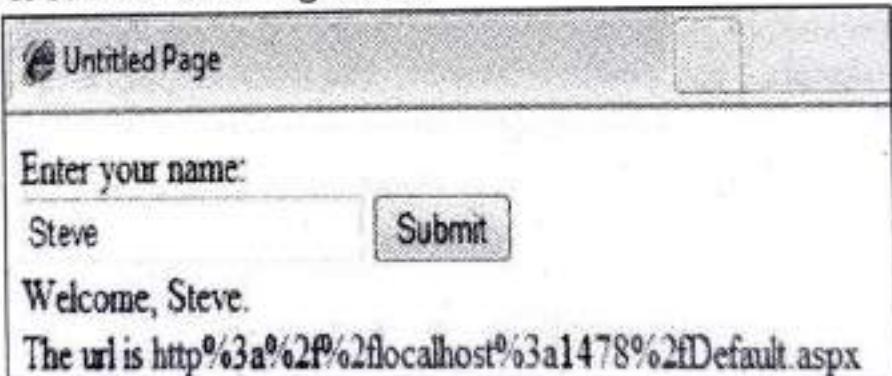


Fig. 4.30

## 4.7 ASP.NET - BASIC CONTROLS

[S-22]

- In this chapter, we will discuss the basic controls available in 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.

*contd. ...*

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.
FindByText(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"
 OnSelectedIndexChanged="RadioButtonList1_SelectedIndexChanged" AutoPostBack="True">
</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.

## 4.8 INTRODUCTION TO FUNCTIONS IN ASP.NET

- Like a sub procedure, a function is used to perform an assignment. The main difference between a sub procedure and a function is that, after carrying its assignment, a function gives back a result. We also say that a function "returns a value". To distinguish both, there is a different syntax you use for a function.

### Using a Script

- If you want to create a function in the code of a web page where you plan to use it, include it in the head section of the web page. The creation of a function starts with `<script>` and ends with `</script>`:

```
<%@ Page Language="VB" %>
<html>
<head>
<script>
```

```
</script>
<title>Exercise</title>
</head>
<body>
</body>
</html>
```

- The `<script>` tag uses various attributes we will review. To start, you must specify that the script will run on the server. To do this, at the attribute `runat` to the script and assign the server string to it. This can be done as follows:

```
<%@ Page Language="VB" %>
<html>
<head>
<script runat="server">
</script>
<title>Exercise</title>
</head>
<body>
</body>
</html>
```

- After doing this, you can create your function between the starting `<script>` and the end `</script>` tags.
- The `<script>` tag is equipped with an attribute named `language`. To specify the language, assign VB, VBScript, JavaScript, JScript, or ECMAScript (remember that the Visual Basic language is not case-sensitive). This can be done as follows:

```
<%@ Page Language="VB" %>
<html>
<head>
<script language="vbscript" runat="server">
</script>
<title>Exercise</title>
</head>
<body>
</body>
</html>
```

- If you don't specify the language, VB is assumed. In most cases, VB and VBScript follow the same rules (but they have some differences). JavaScript and JScript don't follow the same rules. For this reason, if you plan to follow rules other than those of VB, you should specify the language.

Besides the language, you should specify how the code of your script will be formatted or considered. To support this, the <script> tag provides the type attribute. To specify it, assign:

- text/VB to the type attribute if you had, or will, specify the language as VB.
- text/vbsscript to the type attribute if you had, or will, specify the language as VBScript.
- text/javascript to the type attribute if you had, or will, specify the language as JavaScript.
- text/jscript to the type attribute if you had, or will, specify the language as Jscript.
- text/ecmascript to the type attribute if you had, or will, specify the language as ECMA Script.

- Here is an example:

```
<%@ Page Language="VB" %>
<html>
<head>
<script language="vbscript" type="text/vbsscript" runat="server">
</script>
<title>Exercise</title>
</head>
<body>
</body>
</html>
```

- Remember that only the runat attribute is required. The others are optional. After specifying the values of the desired attributes, you can create your procedure.

### Creating a Function

- To create a function, you use the Function keyword followed by a name and parentheses. Unlike a sub procedure, because a function returns a value, you must specify the type of value the function will produce. To give this information, on the right side of the closing parentheses, you can type the As keyword, followed by a data type. To indicate where a function stops, type End Function. Based on this, the minimum syntax used to create a function is:

```
<%@ Page Language="VB" %>
<html>
<head>
<script language="vbscript" type="text/vbsscript" runat="server">
AccessModifier(s) Function FunctionName() As DataType
End Function
</script>
<title>Exercise</title>
</head>
<body>
</body>
</html>
```

- As seen for a sub procedure, a function can have an access modifier. The rules are the same as we described for a sub procedure. The Function keyword is required. The name of a function follows the same rules and suggestions we reviewed for sub procedures.
- The As keyword may be required (in the next sections, we will review the alternatives to the As DataType expression).
- The DataType factor indicates the type of value that the function will return. If the function will produce a word or a group of words, you can create it as String. The other data types are also valid in the contexts we reviewed them. Here is an example:

```
<%@ Page Language="VB" %>
<html>
<head>
<script language="vbscript" type="text/vbsscript" runat="server">
Function GetFullName() As String
End Function
</script>
<title>Exercise</title>
</head>
<body>
</body>
</html>
```

- As done with the variables, you can also use a type character as the return type of a function and omit the As DataType expression. The type character is typed on the right side of the function name and before the opening parenthesis. An example would be GetFullname\$(). As with the variables, you must use the appropriate type character for the function:

Character	The function must return
\$	a String type
%	a Byte, Short, Int16, or In32
&	an Int64 or a Long
!	a Single type
#	a Double
@	a Long integer

- Here is an example:

```
<%@ Page Language="VB" %>
<html>
<head>
<script language="vbscript" type="text/vbsscript" runat="server">
```

```

Function GetFullName$()
End Function
</script>
<title>Exercise</title>
</head>
<body>
</body>
</html>

```

- As mentioned already, the section between the Function and the End Function lines is the body of the function. It is used to describe what the function does. As done on a sub procedure, one of the actions you can perform in a function is to declare a (local) variable and use it as you see fit. Here is an example:

```

<script language="vbscript" type="text/vbsscript" runat="server">
Function GetFullName$()
 Dim Variable As String
End Function
</script>

```

### Returning a Value From a Function

- After performing an assignment in a function, to indicate the value it returns, somewhere after the assignment and before the End Function line, you can type the name of the function, followed by the = sign, followed by the value the function returns. Here is an example in which a function returns a name:

```

<script language="vbscript" type="text/vbsscript" runat="server">
Function GetFullName$()
 Dim FirstName As String, LastName As String
 FirstName = "Gertrude"
 LastName = "Monay"
 GetFullName = LastName & ", " & FirstName
End Function
</script>

```

- Alternatively, instead of using the name of the function to indicate the value it returns, you can type Return, followed by the value or the expression that the function returns. Based on this, the above function could also be created as follows:

```

<script language="vbscript" type="text/vbsscript" runat="server">
Function GetFullName$()
 Dim FirstName As String, LastName As String
 FirstName = "Gertrude"
 LastName = "Monay"
 Return LastName & ", " & FirstName
End Function
</script>

```

- You can also use some local variables in the function to perform an assignment and then assign their result to the name of the function.

## Calling a Function:

- As done for the sub procedure, in order to use a function in your program, you must call it. Like a sub procedure, to call a function, you can simply type its name in the desired section of the program. Since the primary purpose of a function is to return a value, to better take advantage of such a value, you can assign the name of a function to a variable in the section where you are calling the function. Here is an example:

```
<%@ Page Language="VB" %>
<html>
<head>
<script language="vbscript" type="text/vbsscript" runat="server">
Function GetFullName$()
 Dim FirstName As String, LastName As String
 FirstName = "Gertrude"
 LastName = "Monay"
 Return LastName & ", " & FirstName
End Function
</script>
<title>Exercise</title>
</head>
<body>
<%
 Dim FullName$
 FullName = GetFullName()
 Response.Write(FullName$)
%>
</body>
</html>
```

- Here is an example of running this program:

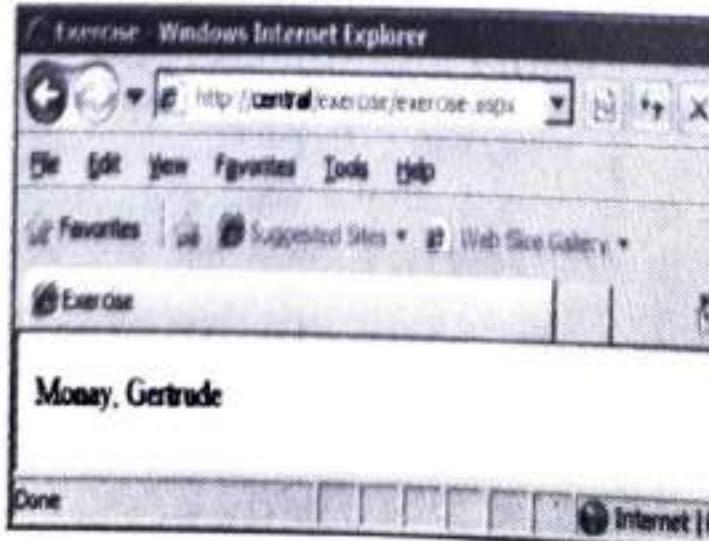


Fig. 4.31

**A Function and a Procedure:**

- Depending on an author, in the Visual Basic language, the word "procedure" includes either a sub-procedure created with the Sub keyword, or a function created with the Function keyword. In the same way, for the rest of our lessons, the word procedure will be used to represent both types. Only when we want to be precise will we use the expression "a sub-procedure" to explicitly mean the type of procedure that does not return a value. When the word "function" is used in our lessons, it explicitly refers to the type of procedure that returns a value.

**4.9 INTRODUCTION TO ASP.NET****4.9.1 ASP.NET - Event Handling**

[S-22]

- 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
DataGridView	SelectedIndexChanged
DownList	SelectedIndexChanged
DropDownList	SelectedIndexChanged
HyperLink	Click
ImageButton	Click
ImageMap	Click
LinkButton	Click
ListBox	SelectedIndexChanged
Menu	MenuItemClick
RadioButton	CheckedChanged
RadioButtonList	SelectedIndexChanged

#### **Example 5:**

- 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>

 <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.
";
 if (Page.IsPostBack)
 {
 lblmessage.Text += "Page post back event handled.
";
 }
 }
 protected void Page_Init(object sender, EventArgs e)
 {
 lblmessage.Text += "Page initialization event handled.
";
 }
 protected void Page_PreRender(object sender, EventArgs e)
 {
 lblmessage.Text += "Page prerender event handled.
";
 }
 protected void btnclick_Click(object sender, EventArgs e)
 {
 lblmessage.Text += "Button click event handled.
";
 }
 }
}
```

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

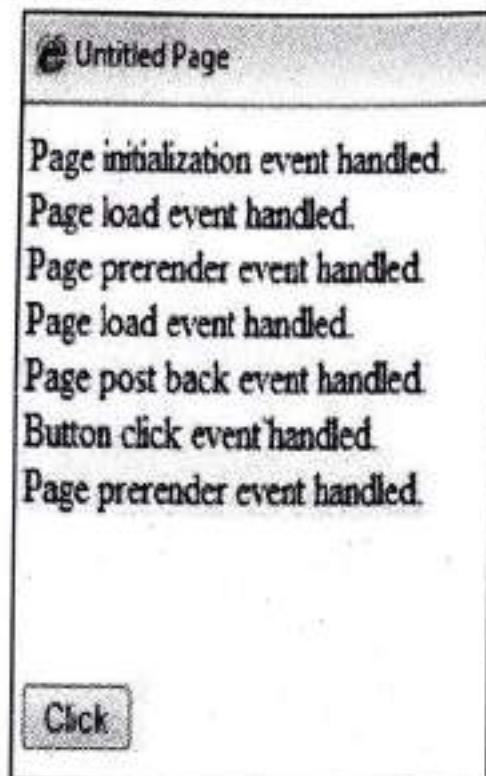


Fig. 4.32

#### 4.9.2 Event Driven Programming and Post Back

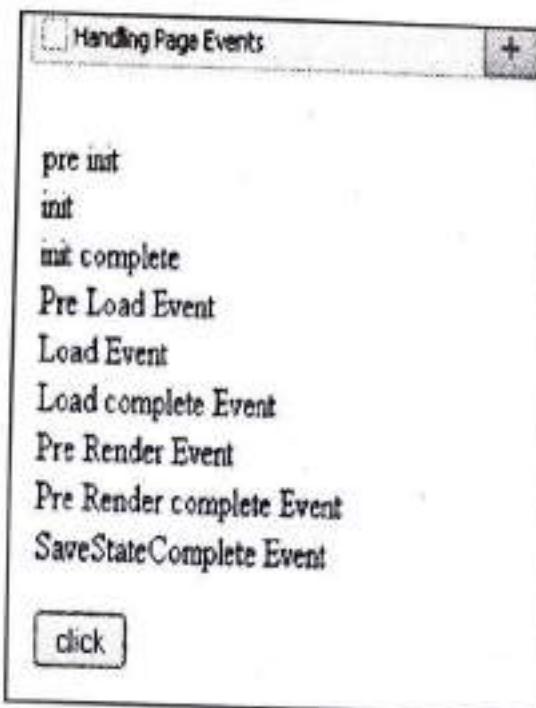
[W-22]

- After you design the look of your ASP.NET page using ASP.NET controls, you need to add the application logic. ASP.NET introduces the concept of an event-driven programming. It enables you to write code that executes in response to events raised by particular user actions. Events can be related to the page or the controls on the page. For example, you can write code to make your application do something when the page loads or a button is clicked.

##### Handling Page Events

- Whenever you request an ASP.NET page, a particular set of events is raised in a particular sequence. This sequence of events is called the page execution life cycle.
- Following is the sequence of events raised whenever you request a page:

1. PreInit
2. Init
3. InitComplete
4. PreLoad
5. Load
6. LoadComplete
7. PreRender
8. PreRenderComplete
9. SaveStateComplete
10. Unload

**Output:****Fig. 4.33****What is PostBack in ASP.NET?**

- PostBack is the name given to the process of submitting an ASP.NET page to the server for processing. PostBack is done if certain credentials of the page are to be checked against some sources (such as verification of username and password using database). This is something that a client machine is not able to accomplish and thus these details have to be 'posted back' to the server.

**What is AutoPostBack Property in ASP.NET?**

- If we create a web Page, which consists of one or more Web Controls that are configured to use AutoPostBack (Every Web controls will have their own AutoPostBack property), the ASP.Net adds a special JavaScript function to the rendered HTML Page. This function is named \_doPostBack(). When Called, it triggers a PostBack, sending data back to the web Server.
- ASP.NET also adds two additional hidden input fields that are used to pass information back to the server. This information consists of ID of the Control that raised the event and any additional information if needed. These fields will empty initially as shown below:

```
<input type="hidden" name="__EVENTTARGET" id="__EVENTTARGET" value="" />
<input type="hidden" name="__EVENTARGUMENT" id="__EVENTARGUMENT" value="" />
```

The `_doPostBack()` function has the responsibility for setting these values with the appropriate information about the event and the submitting the form. The `_doPostBack()` function is shown below:

```
<script language="text/javascript">
```

```
function __doPostBack(eventTarget, eventArgument)
{
 if (!theForm.onsubmit || (theForm.onsubmit() != false))
 {
 theForm.__EVENTTARGET.value = eventTarget;
 theForm.__EVENTARGUMENT.value = eventArgument;
 theForm.submit();
 }
 </script>
}
```

- ASP.NET generates the `_doPostBack()` function automatically, provided at least one control on the page uses automatic postbacks.
- Any Control that has its `AutoPostBack` Property set to true is connected to the `_doPostBack()` function using the `onclick` or `onchange` attributes. These attributes indicate what action Browser should take in response to the Client-Side javascript events `onclick` and `onchange`.
- In other words, ASP.Net automatically changes a client-side javascript event into a server-side ASP.Net event, using the `_doPostBack()` function as an intermediary.

#### Life Cycle of a Web Page:

- To work with the ASP.Net Web Controls events, we need a solid understanding of the web page life cycle. The following actions will be taken place when a user changes a control that has the `AutoPostBack` property set to true :
  - On the client side, the JavaScript `_doPostBack` function is invoked, and the page is resubmitted to the server.
  - ASP.NET re-creates the Page object using the `.aspx` file.
  - ASP.NET retrieves state information from the hidden view state field and updates the controls accordingly.
  - The `Page.Load` event is fired.
  - The appropriate change event is fired for the control. (If more than one control has been changed, the order of change events is undetermined.)
  - The `Page.PreRender` event fires, and the page is rendered (transformed from a set of objects to an HTML page).
  - Finally, the `Page.Unload` event is fired.
  - The new page is sent to the client.
- To watch these events in action, we can create a simple event tracker application. All this application does is writing a new entry to a list control every time one of the events it's monitoring occurs. This allows you to see the order in which events are triggered.

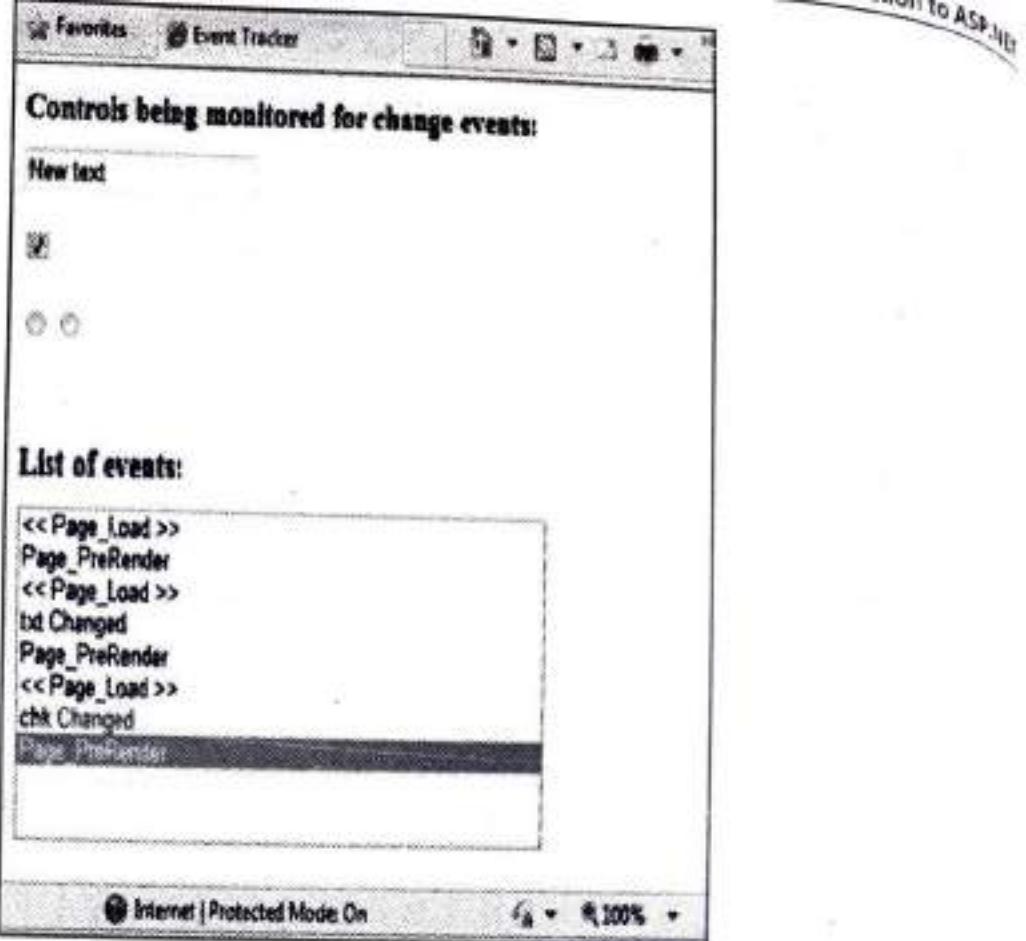


Fig. 4.34

### Event Tracker Web Page:

- I have shown Markup codes and C# Codes below to make this works,

#### EventTracker.aspx

```
<%@ Page Language="C#" AutoEventWireup="true"
 CodeFile="EventTracker.aspx.cs" Inherits="EventTracker" %>
<!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>
 <h1>Controls being monitored for change events:</h1>
 <asp:TextBox ID="txt" runat="server" AutoPostBack="true"
OnTextChanged="Ctrl1Changed" />

 <asp:CheckBox ID="chk" runat="server" AutoPostBack="true"
OnCheckedChanged="Ctrl1Changed"/>

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

```
<asp:RadioButton ID="opt1" runat="server" GroupName="Sample"
AutoPostBack="true" OnCheckedChanged="CtrlChanged"/>
<asp:RadioButton ID="opt2" runat="server" GroupName="Sample"
AutoPostBack="true" OnCheckedChanged="CtrlChanged"/>
<h1>List of events:</h1>
<asp:ListBox ID="lstEvents" runat="server" Width="355px"
Height="150px" />

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

**EventTracker.aspx.cs**

```
using System;
using System.Collections;
using System.Configuration;
using System.Data;
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;
public partial class EventTracker : System.Web.UI.Page
{
 protected void Page_Load(object sender, EventArgs e)
 {
 Log("<< Page_Load >>");
 }
 protected void Page_PreRender(object sender, EventArgs e)
 {
 // When the Page.PreRender event occurs, it is too late
 // to change the list.
 Log("Page_PreRender");
 }
 protected void CtrlChanged(Object sender, EventArgs e)
 {
 // Find the control ID of the sender.
 // This requires converting the Object type into a Control class.
 string ctrlName = ((Control)sender).ID;
 Log(ctrlName + " Changed");
 }
}
```

```

 private void Log(string entry)
 {
 lstEvents.Items.Add(entry);
 // Select the last item to scroll the list so the most recent
 // entries are visible.
 lstEvents.SelectedIndex = lstEvents.Items.Count - 1;
 }
}

```

### What the above Code does?

- The code writes to the ListBox using a private Log() method. The Log() method adds the text and automatically scrolls to the bottom of the list each time a new entry is added, thereby ensuring that the most recent entries remain visible.
- All the change events are handled by the same method, CtrlChanged(). If you look carefully at the .aspx file, you'll notice see that each input control connects its monitored event to the CtrlChanged() method. The event handling code in the CtrlChanged() method uses the source parameter to find out what control sent the event, and it incorporates that information in the log string.
- The page includes event handlers for the Page.Load and Page.PreRender events. As with all page events, these event handlers are connected by method name

## 4.10 INTRODUCTION

### 4.10.1 Web Controls

- Web controls are basically HTML elements wrapped under easy to use scripting tags of ASP+ and provide rich functionality in your FORMs or pages. Web controls range from simple text box to advance grids and lists. Some of the controls like data grid can be bound to data sources much like Visual Basic data bound controls.

#### What are advantages of web controls?

- They are highly customizable and provide common properties and methods throughout.
- Even though web controls provide rich functionality finally they are rendered as plain HTML in the client browser thus avoiding browser compatibility problems.
- They can be used in object oriented manner much like Visual Basic controls.
- They are processed at server side. This makes ASP+ page much readable and cleaner as compared to traditional ASP page.
- They can be data bound.
- State maintenance is automatically taken care by ASP+.

## What are common web controls available?

- Following is a list of common web controls:

- o Label
- o Panel
- o Button
- o TextBox
- o CheckBox
- o RadioButton
- o RadioButtonList
- o ListBox
- o DropDownList
- o Table/TableRow/TableCell
- o DataGrid
- o DataList
- o Repeater
- o Calender
- o Validation controls

## How do I handle Web control events?

- Web controls are processed at server side. This means that all the events generated by a control will be handled at server end. The event handling has changed with ASP+.
- Each event handler now has following syntax:

```
public sub myeventhandler(source as Object, evt as EventArgs) 'access your
web controls here and 'manipulate their properties end sub
```

- Here, source is the object which caused the event to be raised and evt is an object providing more information about the event.

Note: I could not find clearly how to use these parameters from NGWS preview SDK. Also, it is not clear how to pass my own parameters to the event handlers.

## What precautions are needed while coding for web controls?

- All controls must have id attribute set.
- All controls must be set to process at server side.
- All controls must be enclosed within HTML FORM element.
- The form must be set to process server side.
- All controls must have proper start and end tags i.e. <control></control> or <control />.
- If you want to maintain control state you must use POST method.

## How do I use Label web control?

- Label web control is generally used to display labels for other FORM elements. The general syntax is:

```
<asp:Label id="label1" Text="my label text" />
```

Property	Description
Text	Used to set the caption of the label

## How do I use Button web control?

- Button web control is used to represent an action like Ok, Submit, Cancel etc.
- Syntax:**

```
<asp:Button id="button1" Text="My button caption" OnClick="myeventhandler" />
```

Property	Description
Command	Command is used to identify the action to be performed when this button is clicked. Can be any user defined string. Analogous to tag property in VB.
CommandArgument	CommandArgument is used to provide additional information about the action.

Event	Description
OnClick	Fired when user clicks the button. The form is posted to the server. Used to perform action depending on the button clicked.

## How do I use Panel web control?

- It is a wrapper for HTML <DIV> tag.

**Syntax:**

```
<asp:Panel id="Panel1" runat="server" BackImageUrl="url"
 HorizontalAlign="Center|Justify|Left|NotSet|Right" Wrap="True|False"/>
 OR
```

```
<asp:Panel id="Panel1" runat="server" > ----- other controls go here ...
----- </asp:Panel>
```

Property	Description
BackImageUrl	The URL of an image to display behind the control.
HorizontalAlign	The alignment of the panel with respect to surrounding text.
Wrap	True if content should be wrapped within the panel; false otherwise.

## How do I use TextBox web control?

- Text box web control is used to accept user input in your web forms.

**Syntax:**

```
<asp:TextBox id="value" runat="SERVER" Text="text" TextMode="SingleLine | Password" />
```

<b>Property</b>	<b>Description</b>
AutoPostBack	True if client-side changes in the control automatically cause a postback to the server; false otherwise. The default is false.
Columns	The width of the control in characters.
MaxLength	The maximum number of characters allowed within text box.
Rows	Number of rows within the text box.
Text	The text that the user has entered into the box.
TextMode	Possible values are Single, MultiLine, and Password.
Wrap	Indicates whether text should wrap around as users type text into a multiline control.
Event	Description
OnTextChanged	Raised on the server when the contents of the text box change. This event does not cause the Web Forms page to be posted to the server unless the AutoPostBack property is set to true.

### How do I use CheckBox web control?

- Check box is used to indicate on or off state.

#### Syntax:

```
<asp:CheckBox id="CheckBox1" runat="server" Text="label"
 OnCheckedChanged="OnCheckedChangedMethod" />
```

<b>Property</b>	<b>Description</b>
AutoPostBack	True if client-side changes in the check box automatically cause a postback to the server; false otherwise. The default is false.
Checked	True if the check box is checked, false otherwise. The default is false.
TextAlign	The position of the caption. Possible values are Right and Left. The default is Right.
Text	The check box caption.
Event	Description
OnCheckedChanged	Raised when the user clicks the checkbox. This event does not cause the Web Forms page to be posted to the server unless the AutoPostBack property is set to true.

### How do I use RadioButton web control?

#### Syntax:

```
<asp:RadioButton id="RadioButton1" runat="server" Text="label"
 OnCheckedChanged="OnCheckedChangedMethod" />
```

Property	Description
AutoPostBack	true if client-side changes in the control automatically cause a postback to the server; false otherwise. The default is false.
Checked	true if the radio button is checked, false otherwise. The default is false.
GroupName	The name of a group to which the radio button belongs. Radio buttons with the same group name are mutually exclusive.
TextAlign	The position of the caption. Possible values are Right and Left. The default is Right.
Text	The radio button caption.
Event	Description.
OnCheckedChanged	Raised when the user clicks the radio button. This event does not cause the Web Forms page to be posted to the server unless the AutoPostBack property is set to true.

### How do I use ListBox or DropDownList web controls?

- ListBox presents a list of item from which user can select one or many items.

```
<asp:ListBox id="Listbox1" runat="server"
 OnSelectedIndexChanged="OnSelectedIndexChangedMethod"
/>
<asp:Listitem value="value" selected="True|False">
 Text
</asp:Listitem>
</SP:ListBox>
```

**OR**

```
<asp:DropDownList id="Combo1" runat="server"
 OnSelectedIndexChanged="OnSelectedIndexChangedMethod"
/>
<asp:Listitem value="value" selected="True|False">
 Text
</asp:Listitem>
</asp:DropDownList>
```

Properties	Description
AutoPostBack	True if client-side changes in the control automatically cause a postback to the server; false otherwise. The default is false.
DataTextField	The field or property of the object in DataSource that will be the source of data for the Text property of individual list items.

contd...

<b>DataValueField</b>	The field or property of the object in DataSource that will be the source of data for the Value property of individual list items.
<b>DataSource</b>	A source from which list items are fetched
<b>Items</b>	A collection of ListItem objects representing individual items within the drop-down list.
<b>SelectedIndex</b>	The ordinal index of the currently selected item in the drop-down list.
<b>SelectedItem</b>	A reference to the Value property of the currently-selected item in the list.
<b>SelectedItems</b>	The collection of ListItem objects currently selected. If SelectionMode is set to Single, this collection contains the same item as SelectedItem.
<b>Rows</b>	The number of rows to display.
<b>SelectionMode</b>	Single if the user can select only one item; Multiple if the user can hold down SHIFT or CTRL while clicking to select multiple items.
<b>List Item Properties</b>	Description
<b>Selected</b>	true if the item is the one selected in the drop-down list and that appears in the text box; false otherwise. Only one item can be declared as selected.
<b>Text</b>	The text that appears in the list.
<b>Value</b>	The value of the item selected by the user, which can be different from the text displayed in the list.
<b>Event</b>	Description
<b>OnSelectedIndexChanged</b>	Raised on the server when the selection of the ListBox control changes. This event does not cause the Web Forms page to be posted to the server unless the AutoPostBack property is set to true.

### How do I use Table web controls?

- Table web controls represent HTML table. You can manipulate rows and columns programmatically.

```
<asp:Table id="Table1" runat="server" />
 <asp:TableRow>
 <asp:TableCell>
 Cell text
 </asp:TableCell>
 </asp:TableRow>
</asp:Table>
```

Properties of Table	Description
BackImageUrl	The URL of an image to display.
CellPadding	The amount of space, in pixels, between the contents of a cell and the cell's border.
CellSpacing	The amount of space, in pixels, between cells.
GridLines	None, Columns, Rows, Both. Whether grid lines are displayed in the table. The default is None.
HorizontalAlign	Center, Justify, Left, NotSet, Right. The alignment of the table with respect to surrounding text.
Rows	Collection of TableRow objects representing individual rows in the table.
Properties of TableRow	Description
Cells	Collection of TableCellCollection objects representing individual cells within the row.
HorizontalAlign	The default horizontal alignment of the cells in the table row.
VerticalAlign	The default vertical alignment of the cells in the table row.
Properties of TableCell	Description
ColumnSpan	The number of columns that this is cell should occupy horizontally.
HorizontalAlign	The horizontal alignment of the contents of the cell.
RowSpan	The number of rows that this is cell should occupy vertically.
VerticalAlign	The vertical alignment of the contents of the cell.
Wrap	Indicates whether text should wrap within the cell

#### 4.10.2 ASP.NET - Server Controls

- Server Controls are the tags that are understood by the server. There are basically three types of server controls.
  - HTML Server Controls:** Traditional HTML tags.
  - Web Server Controls:** New ASP.NET tags.
  - Validation Server Controls:** For input validation.

##### ASP.NET HTML Server Controls:

- ASP.NET** provides a way to work with HTML Server controls on the server side. programming with a set of controls collectively is called **HTML Controls**.
- These controls are grouped together in the Visual Studio Toolbox in the **HTML Control** tab. The markup of the controls is similar to the HTML control.
- These controls are basically the original HTML controls but enhanced to enable server side processing.

- HTML elements in ASP.NET files are, by default, treated as text. To make these elements programmable, add a runat="server" attribute to the HTML element. This attribute indicates that the element should be treated as a server control.

**Note:**

- All HTML server controls must be within a <form> tag with the runat="server" attribute. The runat="server" attribute indicates that the form should be processed on the server. It also indicates that the enclosed controls can be accessed by server scripts.
- The **System.Web.UI.HtmlControls.HtmlControl** base class contains all of the common properties. HTML server controls derive from this class.
- For example, consider the HTML input control:  
`<input type="text" size="40">`
- It could be converted to a server control, by adding the runat and id attribute:  
`<input type="text" id="texttext" size="40" runat="server">`
- The following table describes the HTML server controls:

Control Name	HTML tag
HtmlHead	<head>element
HtmlInputButton	<input type=button submit reset>
HtmlInputCheckbox	<input type=checkbox>
HtmlInputFile	<input type=file>
HtmlInputHidden	<input type=hidden>
HtmlInputImage	<input type=image>
HtmlInputPassword	<input type=password>
HtmlInputRadioButton	<input type=radio>
HtmlInputReset	<input type=reset>
HtmlText	<input type=text password>
HtmlImage	<img>element
HtmlLink	<link>element
HtmlAnchor	<a>element
HtmlButton	<button>element
HtmlButton	<button>element
HtmlForm	<form>element
HtmlTable	<table>element
HtmlTableCell	<td>and<th>
HtmlTableRow	<tr>element
HtmlTitle	<title>element
HtmlSelect	<select>element

## ASP.NET Web Server Controls

- Web server controls are special ASP.NET tags understood by the server.
- Like HTML server controls, Web server controls are also created on the server and they require a runat="server" attribute to work.
- However, Web server controls do not necessarily map to any existing HTML elements and they may represent more complex elements.
- Mostly all Web Server controls inherit from a common base class, namely the **WebControl** class defined in the **System.Web.UI.WebControls** namespace.
- The syntax for creating a Web server control is:  
`<asp:control_name id="some_id" runat="server"/>`
- The following table describes the WEB server controls:

Control Name	HTML tag
AdRotator	Display a sequence of images
Button	Displays a push button
Calendar	Display a calendar
CalendarDay	A day in a calendar control
CheckBox	Displays a check box
CheckBoxList	Creates a multi-selection check box group
GridView	Displays fields of a data source in a grid
DataSource	Displays items from a data source by using templates
DropDownList	Creates a drop-down list
HyperLink	Creates a hyperlink
Image	Displays a image
ImageButton	Displays a clickable image
Label	Displays static control which is programmable (lets you apply styles to its content)
LinkButton	Creates a hyperlink button
ListBox	Creates a single or multi-selection drop-down list
ListItem	Creates an item in a list
Literal	Displays static control which is programmable (does not let you apply styles to its content)
Panel	Provides a container for other controls
PlaceHolder	Reserves space for controls added by code
RadioButton	Creates a radio button
RadioButtonList	Creates a group of radio buttons
BulletedList	Creates a list in bullet format
Repeater	Displays a repeated list of items bound to the control
Style	Sets the style of controls
Table	Creates a table

## ASP.NET Validation Server Controls

- After you create a web form, you should make sure that mandatory fields of the form elements such as login name and password are not left blank; data inserted is correct and is within the specified range. Validation is the method of scrutinizing (observing) that the user has entered the correct values in input fields.
- A Validation server control is used to validate the data of input control. If the data does not pass validation, it will display an error message to the user.
- In ASP.NET you can use ASP.NET Validation Controls while creating the form and specify what ASP.NET Validation Controls you want to use and to which server control you want to bind this.
- Validation Controls are derived from a common base class and share a common set of properties and methods. You just have to drag and drop the ASP.NET Validation Control in the web form and write one line of code to describe its functionality.
- This reduces the developer time from writing JavaScript for each type of validation. Moreover, through ASP.NET Validation Controls if any invalid data is entered the browser itself detects the error on the client-side and displays the error without requesting the server. This is another advantage because it reduces the server load.

Some Server Validation controls are:

Validation Server Control	Description
CompareValidator	Compares the value of one input control to the value of another input control or to a fixed value.
CustomValidator	Allows you to write a method to handle the validation of the value entered
RangeValidator	Checks that the user enters a value that falls between two values
RegularExpressionValidator	Ensures that the value of an input control matches a specified pattern
RequiredFieldValidator	Makes an input control a required field
ValidationSummary	Displays a report of all validation errors occurred in a Web page

The syntax for creating a Validation server control is:

```
<asp:control_name id="some_id" runat=server>
```

### Default.aspx Page

```
<%@ 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></title>
</head>
<body>
 <form id="form1" runat="server">
 <div style="width: 414px; background-color: #E6E6FA; height: 110px;">
 Name <asp:textbox id="TextBox1" runat="server" style="z-index: 1;
 left: 73px; top: 15px; position: absolute;">
 </asp:textbox>
 <asp:requiredfieldvalidator id="RequiredFieldValidator1"
 runat="server" controltovalidate="TextBox1"
 errormessage="Cannot be Blank" setfocusonerror="True"
 style="position: absolute;
 z-index: 1; left: 238px; top: 21px">
 </asp:requiredfieldvalidator>

 Mobile
 <asp:textbox id="TextBox2" runat="server" style="position:
 relative; top: 4px; left: 12px">
 </asp:textbox>

 <tr>
 <td class="style3">
 Email
 </td>
 <td class="style2">
 <asp:regularexpressionvalidator id="remail" runat="server"
 controltovalidate="txtemail"
 errormessage="Enter in proper format" validationexpression=
 "\w+([-.\.]\w+)*@\w+([-.\.]\w+)*\.\w+([-.\.]\w+)"
 style="z-index: 1; left: 279px; top: 61px; position:
 absolute">
 </asp:regularexpressionvalidator>
 <asp:textbox id="txtemail" runat="server" width="200px"
 style="z-index: 1; left: 73px;
 top: 62px; position: absolute">
 </asp:textbox>
 </td>
 <td>
 </td>
 </tr>
```

```

<asp:button id="Button1" runat="server" text="SUBMIT"
 style="z-index: 1; left: 73px;
 top: 91px; position: absolute; right: 1066px" />

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

```

**Output:**

Name	<input type="text"/>
Mobile	<input type="text"/>
Email:	<input type="text"/>
<input type="button" value="SUBMIT"/>	

- If any input field is left blank then it is handled by the control name as RequiredField Validation control.

Name	<input type="text"/>	Cannot be Blank
Mobile	<input type="text" value="9923465441"/>	
Email:	<input type="text" value="alina11@gmail.com"/>	
<input type="button" value="SUBMIT"/>		

- It ensures that the value of an input control matches a specified pattern and this is handled by the RegularExpression Validation control.

Name	<input type="text" value="alina"/>	
Mobile	<input type="text" value="9923465441"/>	
Email:	<input type="text" value="a878-=+  "/>	Enter in proper format
<input type="button" value="SUBMIT"/>		

#### 4.10.3 .ASP.NET - Client Side 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:

<b>Event</b>	<b>Description</b>
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 <code>ServerClick</code> 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>

<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></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.

#### **4.10.4 Navigation controls**

- Navigation controls are very important for websites. Navigation controls are basically used to navigate the user through webpage. It is more helpful for making the navigation of pages easier. There are three controls in ASP.NET, which are used for Navigation on the webpage.
  - TreeView control
  - Menu Control
  - SiteMapPath control
- There are some Namespaces, which are used for above Navigation controls which are given below:

`Using System.Web.UI.WebControls.TreeView;`

`Using System.Web.UI.WebControls.Menu;`

`Using System.Web.UI.WebControls.SiteMapPath;`

- In this tutorial, i will show you, how to add navigation control on the web page. I will also give you real example of each control. Please read each control very carefully and use it on ASP.NET website. You can download each control application from bottom and implement on your system.

##### **4.10.4.1 The TreeView Control**

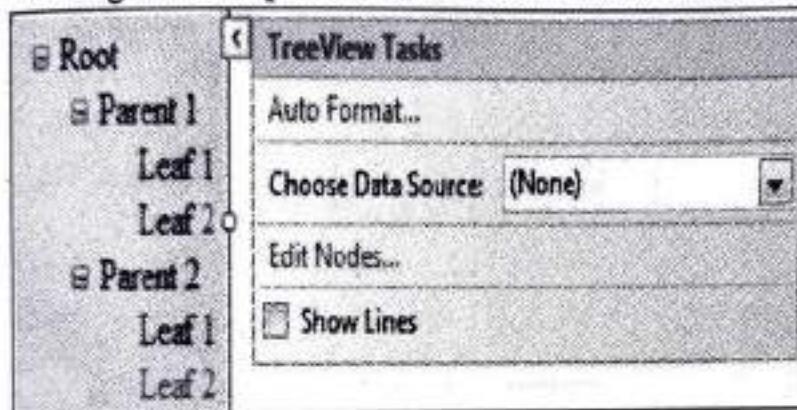
- The TreeView control is used for logically displaying the data in a hierarchical structure. We can use this navigation control for displaying the files and folders on the webpage. We can easily display the XML document, Web.SiteMap files and Database records in a tree structure.

- There are some types to generate navigation on webpage through TreeView control.
  - TreeView Node Editor dialog box
  - Generate TreeView based on XML Data
  - Generate TreeView based on Web.Sitemap data
  - Generate TreeView from Database.

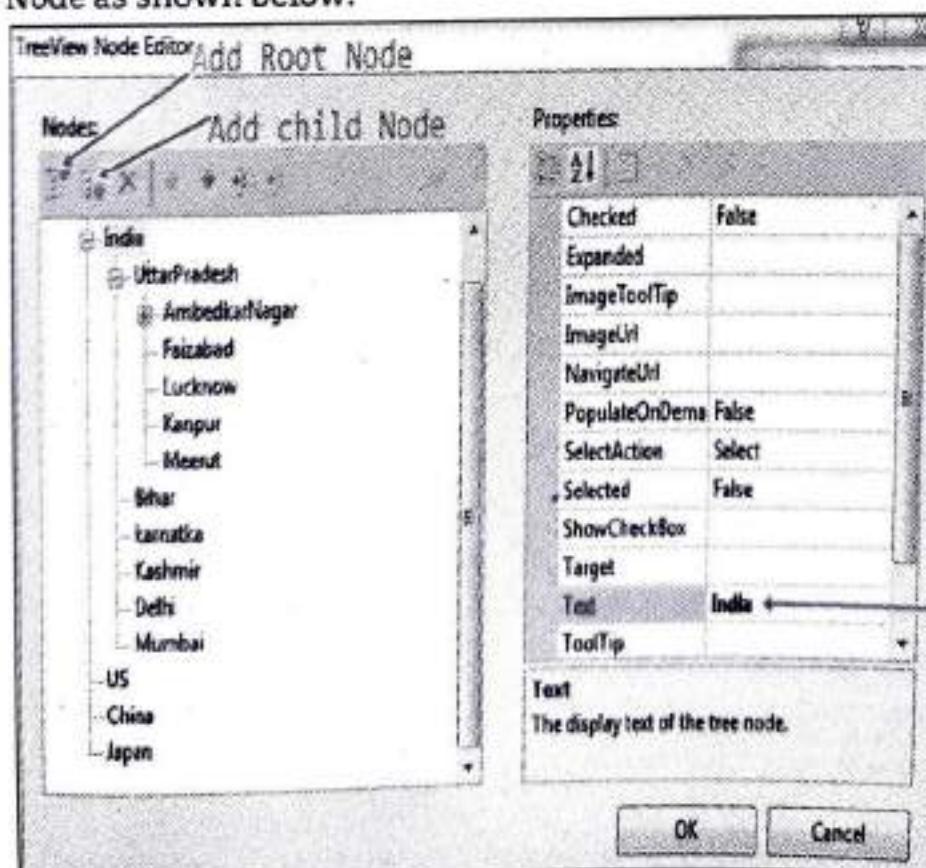
#### 4.10.4.1.1 TreeView Node Editor Dialog Box

There are some steps to generate the Tree structure on the page, which are given below:

**Step 1 :** First open your visual studio → File → New → Website → Select ASP.NET Empty Website → OK → open solution explorer → Add New Web Form → Drag and Drop TreeView control from Toolbox as shown below:



**Step 2 :** Now go properties of TreeView control → Click Nodes → Add Root and child Node as shown below:



**Step 3 :** Now Run the Program (press F5).

**Output:****4.10.4.1.2 Generate TreeView Based on XML Data**

- There are some steps to implement this concepts on the webpage, which are given below:

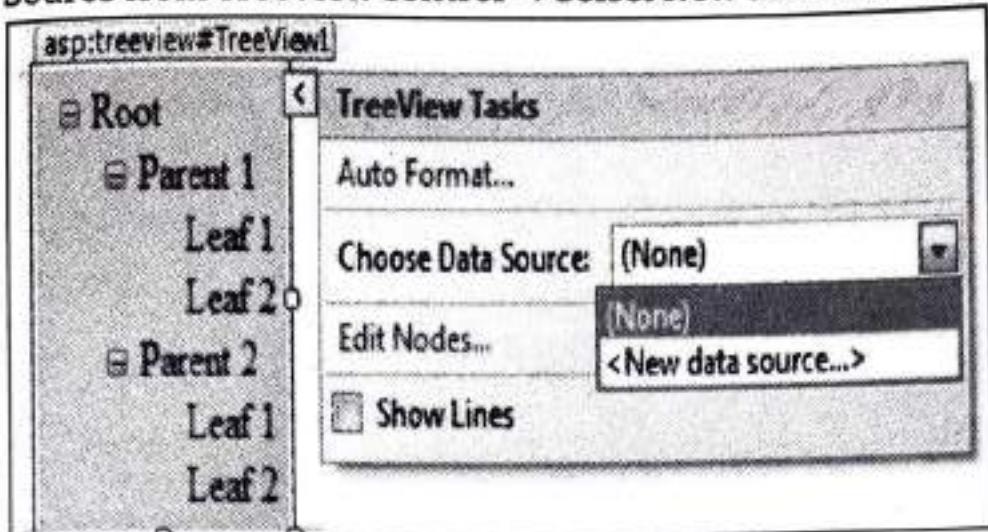
**Step 1 :** Now First Add A Web Form And A XML File In Your Solution Explorer  
 → Now Open The XML File And Write The Following Codes As Shown Below  
 → Now Click Save.

```

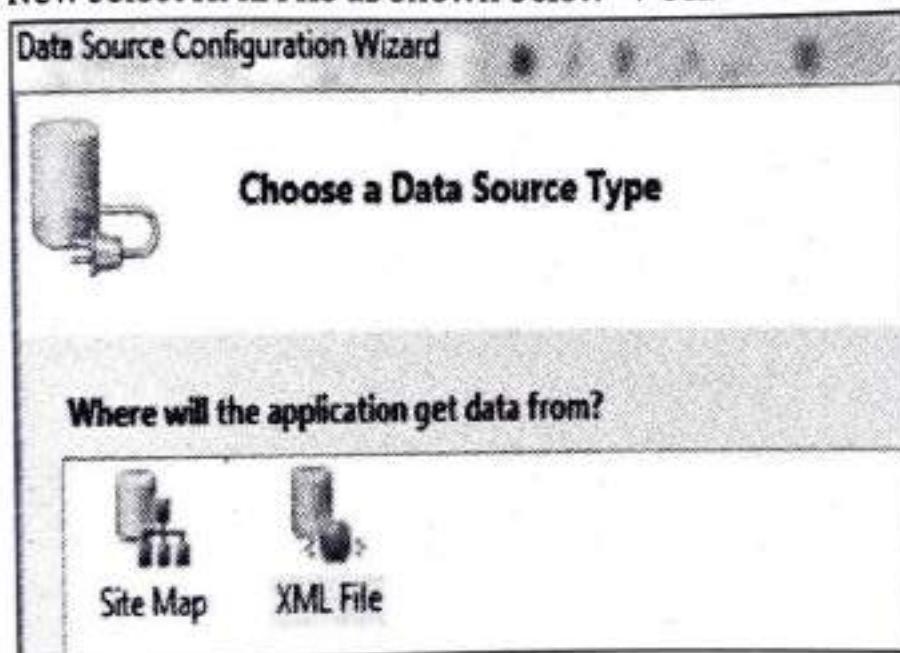
<?xml version="1.0" encoding="utf-8" ?>
<application>
 <homepage title="Country" value="default.aspx">
 <page title ="INDIA" value="default.aspx">
 <subpage title ="up" value="default.aspx"/>
 <subpage title ="delhi" value="default.aspx"/>
 <subpage title ="mumbai" value="default.aspx"/>
 <subpage title ="kolkata" value="default.aspx"/>
 </page>
 <page title ="US" value="default.aspx"/>
 <page title ="CHNIA" value="default.aspx"/>
 <page title ="JAPAN" value="default.aspx"/>
 </homepage>
</application>

```

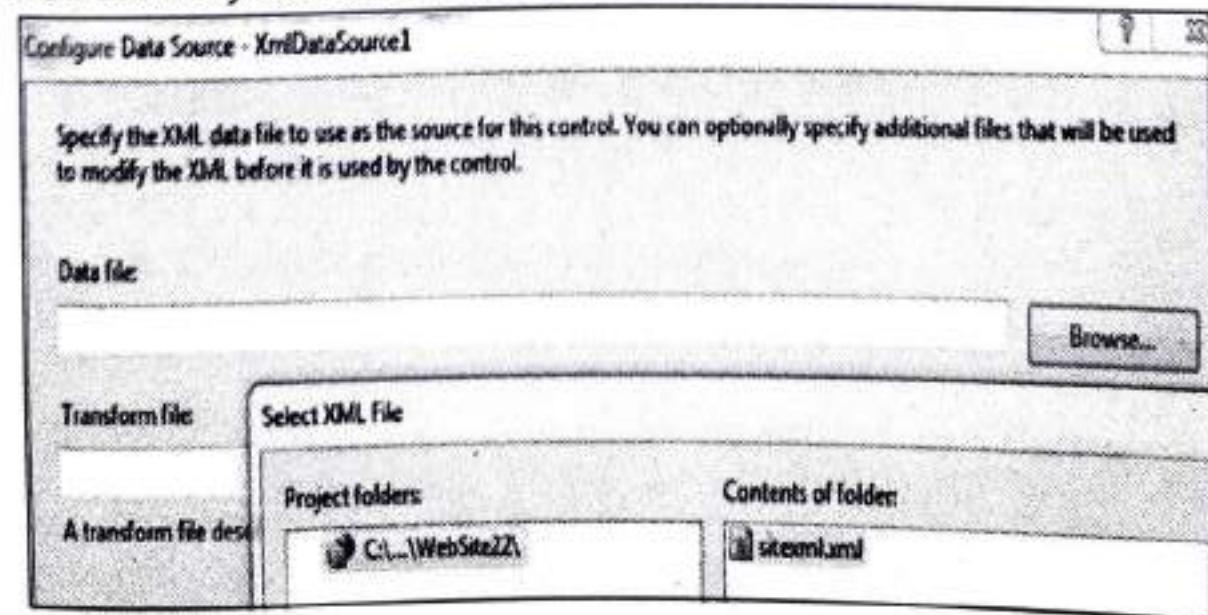
**Step 2 :** Now Drag and drop TreeView control on the Web Form → Now Choose Data Source from TreeView control → Select New data source as shown below:



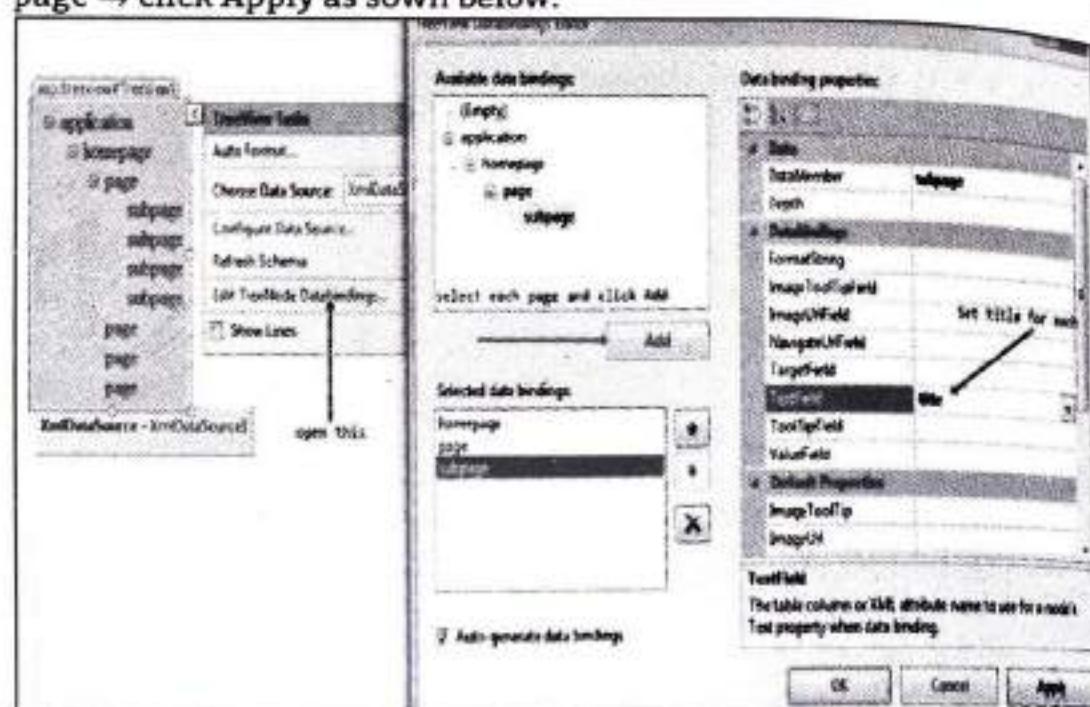
**Step 3 :** Now select XML File as shown below → OK.



**Step 4 :** Now Browse your XML File as shown below → OK.

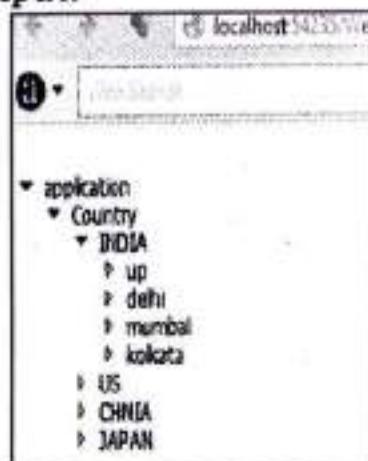


**Step 5 :** Now click Edit TreeNode DataBindings → Select each page one by one → and click Add button → set TextField = title from right side for each page → click Apply as shown below:



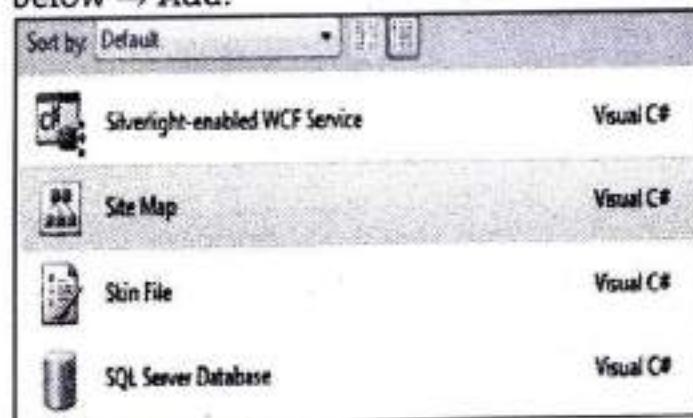
**Step 6 :** Now run the program (press F5).

**Output:**



#### 4.10.4.1.3 Generate TreeView Based On Sitemap Data

**Step 1 :** First Add a Web Form and a SiteMap in Solution Explorer as shown below → Add.



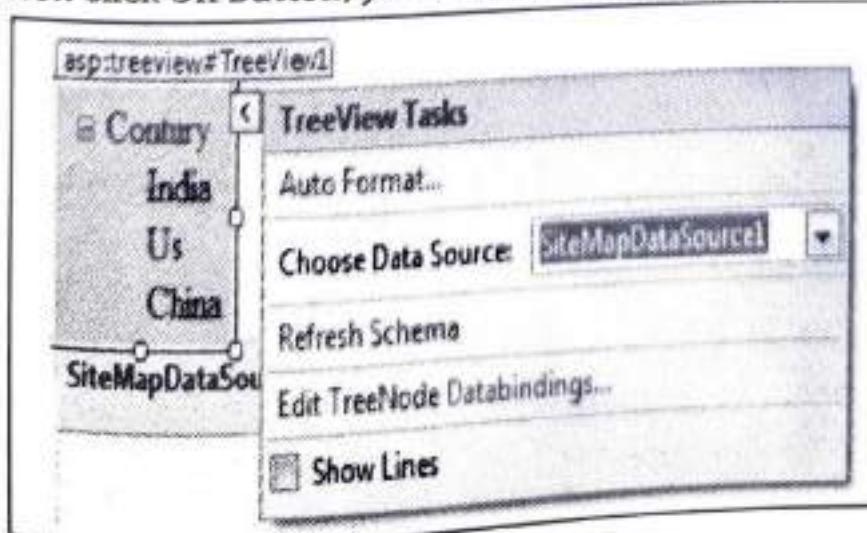
**Step 2 :** Open web.sitemap file and write the following codes, which are given below → Save

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
 <siteMapNode url="default.aspx" title="Contury" description="" >
 <siteMapNode url="treeview1.aspx" title="India" description="" />
 <siteMapNode url="menu.aspx" title="Us" description="" />
 <siteMapNode url="menu1.aspx" title="China" description="" />
 </siteMapNode>
</siteMap>
```

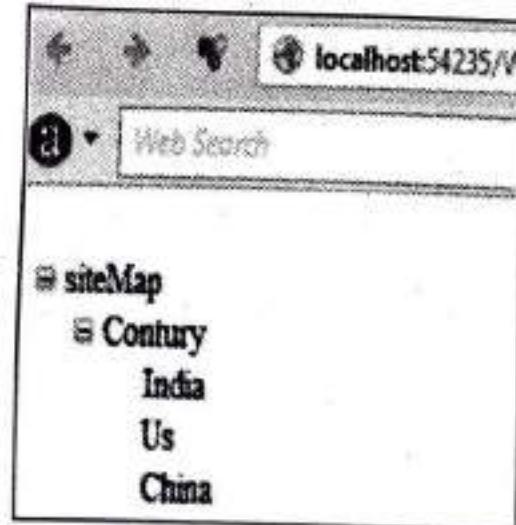
**Step 3 :** Now drag and drop TreeView control on the Form → Now choose Data Source → select New data source → Select SiteMap as shown below:



**Step 4 :** Now click OK Button, you will see following output.



**Step 5 :** Now run the program (press F5).



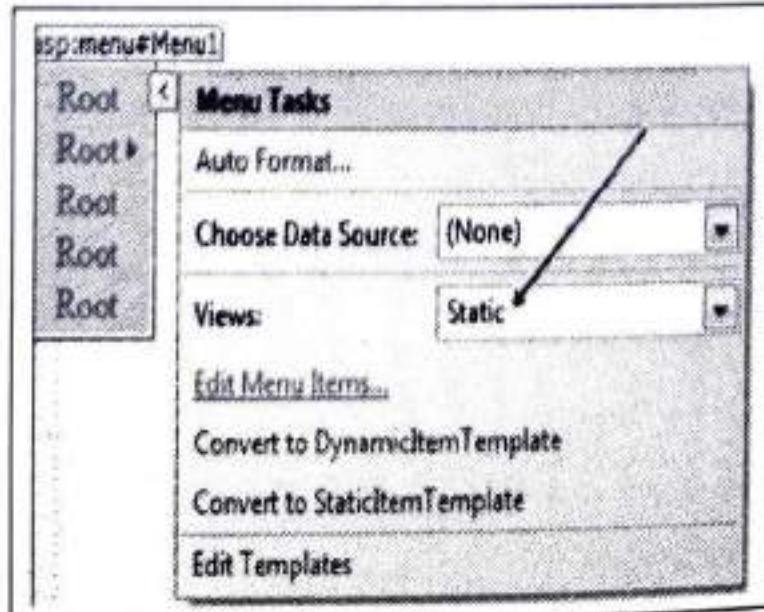
#### 4.10.4.2 The Menu Control

- The menu control is a Navigation control, which is used to display the site navigation information. This can be used to display the site data structure vertically and horizontally. It can be used a binding control as TreeViewcontrol. Means we can easily bind the XML and SiteMap data in menu control.
- The menu control can be used as two types:
  - Static menu:** It is used to display the parent menu items and their sub menu items on the page. Means it is used to display the entire structure of the static menu.
  - Dynamic menu:** It is used to display the static menu as well as dynamic menu on the site. It Means when user passes the mouse over the menu then it will appear on the site.

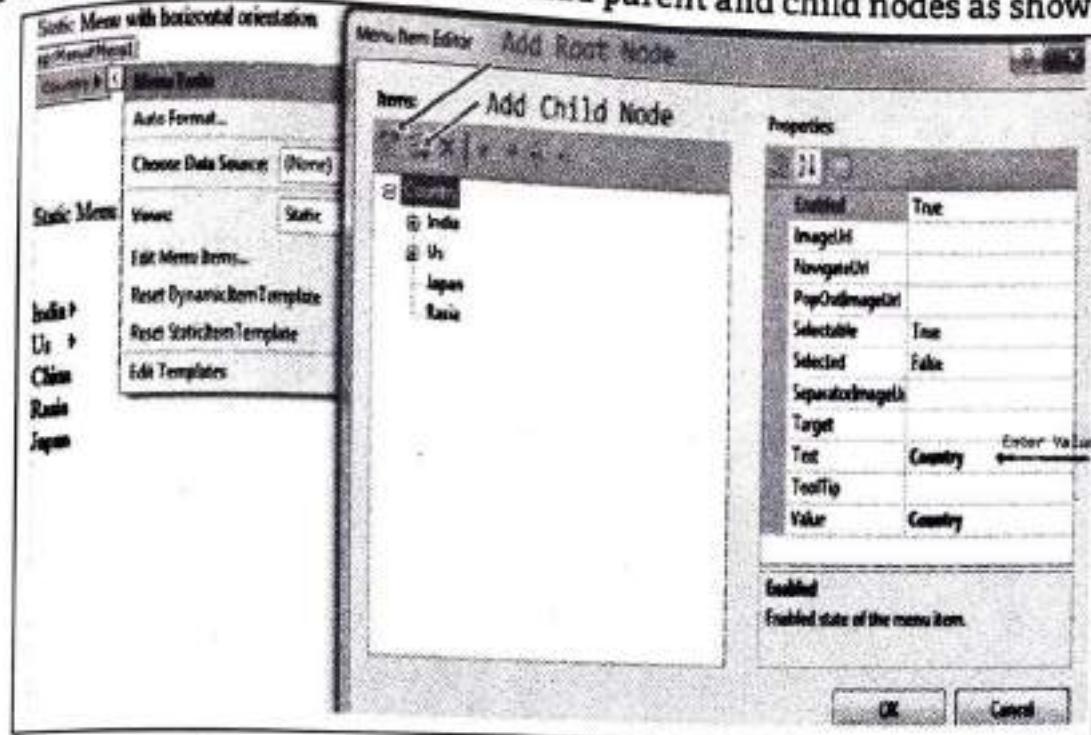
##### 4.10.4.2.1 Static Menu

- We can display site structure vertically as well as horizontally through static menu control on the site. There are some steps to implement the static menu control on the Web Form. Which are given below:

**Step 1 :** First Add a New Web Form in solution Explorer → drag and drop menu control on the Form → now select Views = static as shown below:

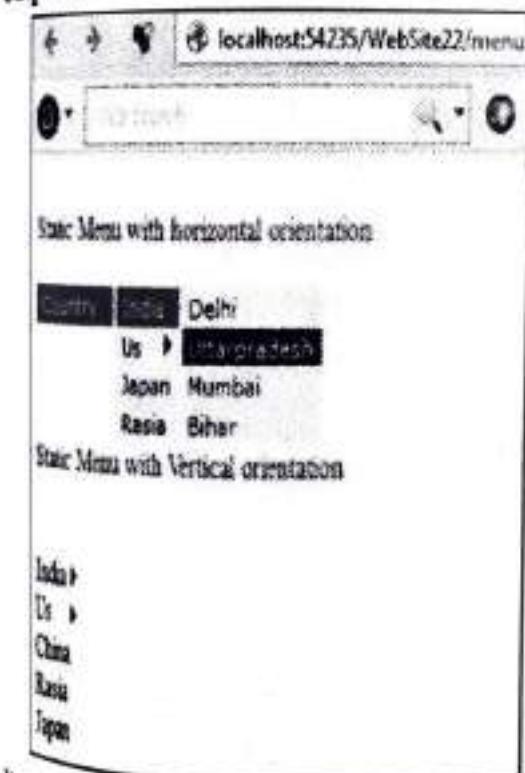


**Step 2 : Now click Edit Menu Items → Add parent and child nodes as shown below:**



**Step 3 : Now Run the program (press F5).**

**Output:**



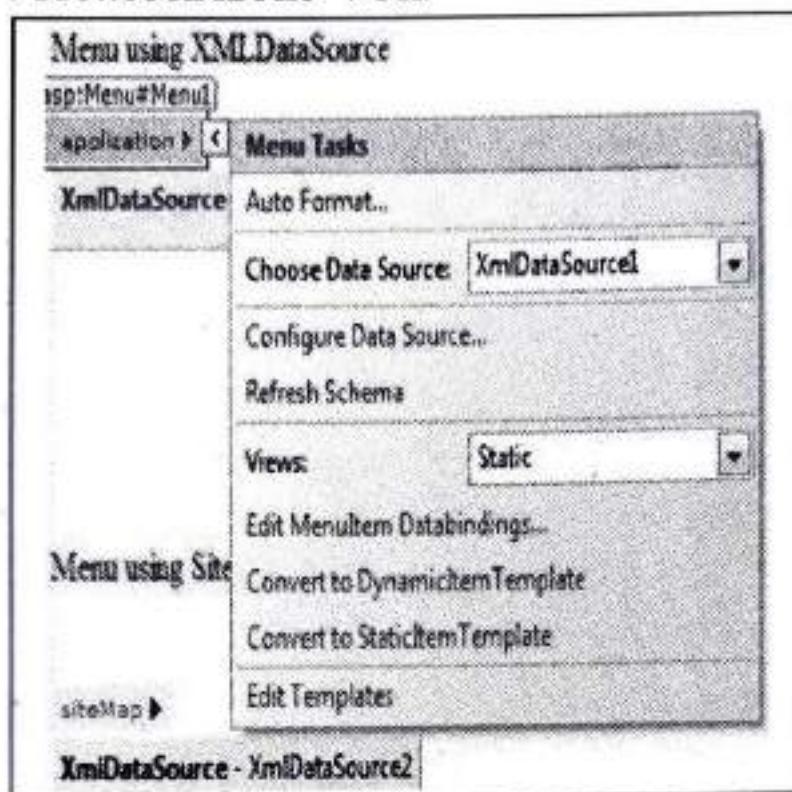
**Note:** You can implement the vertical orientation as horizontal orientation.

#### 4.10.4.2.2 Dynamic Menu

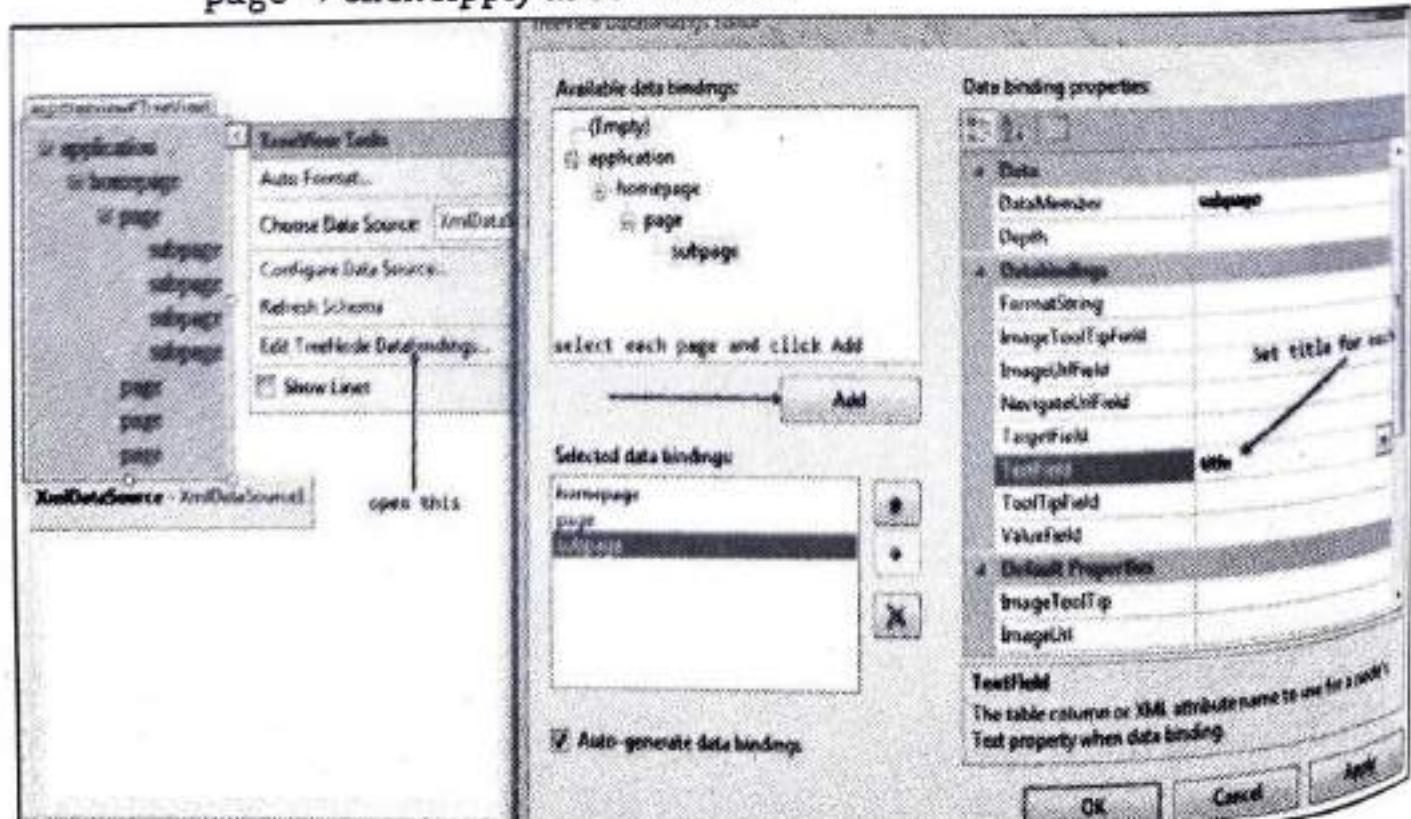
When user passes the mouse over the control, the data will automatically appear on the site. we can generate dynamic menu control in two ways:

1. Generate menu control using Xml Data source.
2. Generate menu control using SiteMap Data source.

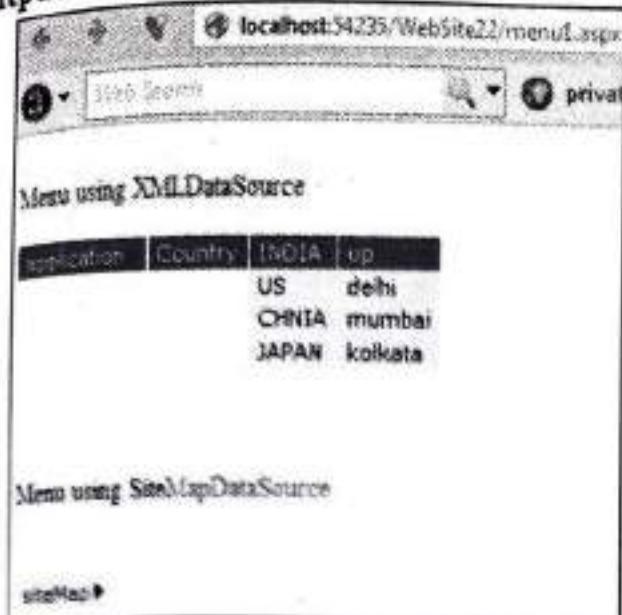
- There are some steps to implement this concepts on the site, which are given below:
- Step 1 :** First Add a Web Form in the solution Explorer → Drag and drop menu control on the Form → choose Data Source → Select XML File → OK → Browse XML File → OK.



**Step 2 :** Now click Edit TreeNode DataBindings → Select each page one by one → and click Add button → set TextField = title from right side for each page → click Apply as shown below:



**Step 3 :** Now run the program (F5) →

**Output:**

**Note:** We can use same process for SiteMap File also.

#### 4.10.4.2.3 The SiteMapPath Control

[S-23]

- The SiteMapPath control is also used to display the Navigation information on the site. It displays the current page's context within the entire structure of a website.
- There are some steps to implement the SiteMapPath control on the web page. Which are given below:

**Step 1 :** First open your visual studio → File → New → Website → Select ASP.NET Empty Website → Open solution Explorer → Add a Web Form (SiteMap.aspx) → Now again Add Site Map File in solution Explorer → open web.sitemap file → write the following codes , which are given below:

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
 <siteMapNode url="SiteMap.aspx" title="Country" description="" >
 <siteMapNode url="page1.aspx" title="India" description="" />
 <siteMapNode url="page2.aspx" title="China" description="" />
 <siteMapNode url="page3.aspx" title="US" description="" />
 </siteMapNode>
</siteMap>
```

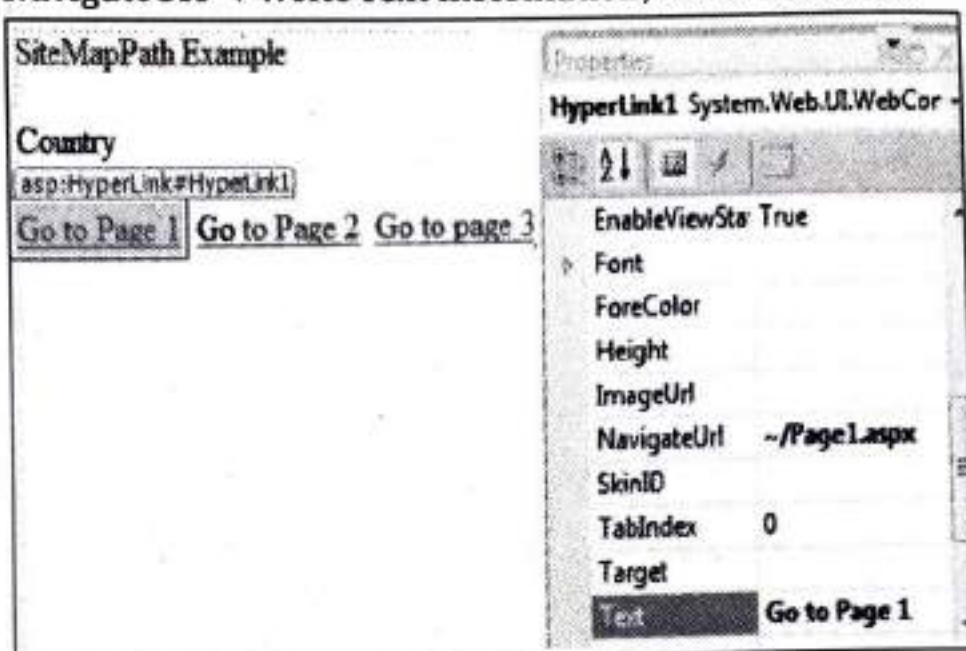
**Step 2 :** Now drag and drop SiteMapPath control on the web Form (SiteMap.aspx) → Now drag and drop HyperLink control on the Form as shown below:

SiteMapPath Example

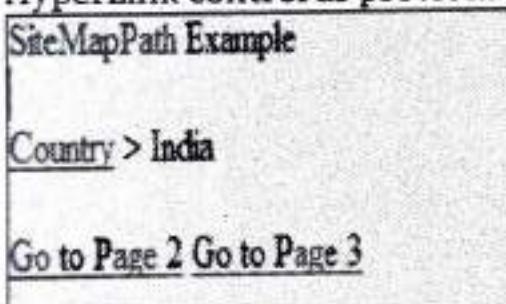
Country

[Go to Page 1](#) [Go to Page 2](#) [Go to page 3](#)

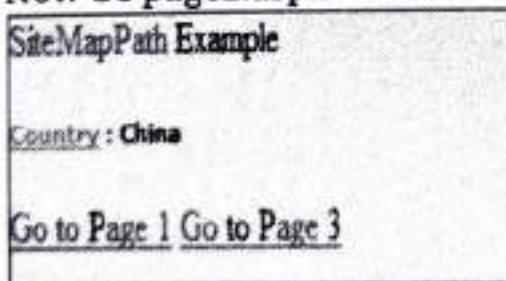
**Step 3 :** Now Add three more Web Form (page1.aspx ,page2.aspx, page3.aspx) in Solution Explorer → Go properties of HyperLink Button control → set NavigateUrl → Write Text Information, as shown below:



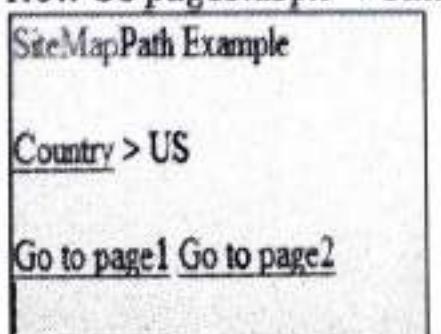
**Step 4 :** Now Go page1.aspx → drag and drop SiteMapPath control and HyperLink control on the Form as shown below → Set the NavigateUrl of each HyperLink control as previous i have done.



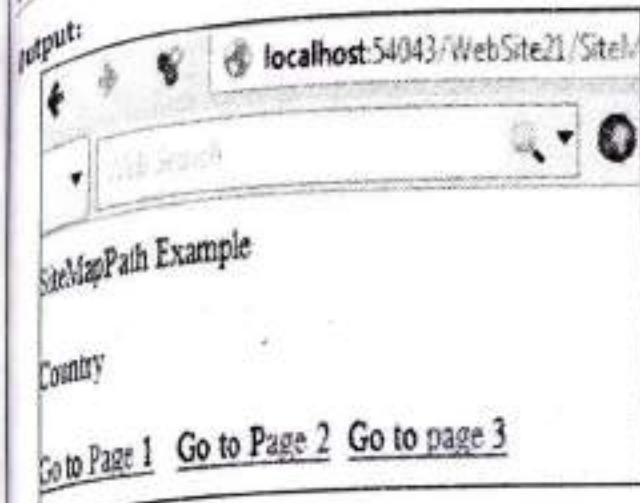
**Step 5 :** Now Go page2.aspx → Same steps perform as step 4.



**Step 6 :** Now Go page3.aspx → Same steps perform as step 4 and step 5.



**Step 7 :** Now run the program (press F5).



## 10.5 ASP.NET - Validation

- ASP.NET validation controls validate the user input data to ensure that useless, unauthenticated, or contradictory data don't get stored.
- ASP.NET provides the following validation controls:
  - RequiredFieldValidator
  - RangeValidator
  - CompareValidator
  - RegularExpressionValidator
  - CustomValidator
  - ValidationSummary

### Why we use validation controls?

- Validation is important part of any web application. User's input must always be validated before sending across different layers of the application.

#### 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.

contd. . .

<b>Text</b>	Error text to be shown if validation fails.
<b>IsValid</b>	Indicates whether the value of the control is valid.
<b>SetFocusOnError</b>	It indicates whether in case of an invalid control, the focus should switch to the related input control.
<b>ValidationGroup</b>	The logical group of multiple validators, where this control belongs.
<b>Validate()</b>	This method revalidates the control and updates the <b>IsValid</b> property.

- Validation controls are used to:
  - Implement presentation logic.
  - To validate user input data.
  - Data format, data type and data range is used for validation.

### **Validation is of two types:**

- Client Side
- Serve Side
- Client side validation is good but we have to be dependent on browser and scripting language support.
- Client side validation is considered convenient for users as they get instant feedback. The main advantage is that it prevents a page from being postback to the server until the client validation is executed successfully.
- For developer point of view serve side is preferable because it will not fail, it is not dependent on browser and scripting language.
- You can use ASP.NET validation, which will ensure client, and server validation. It works on both ends first it will work on client validation and than on server validation. At any cost server validation will work always whether client validation is executed or not. So you have a safety of validation check.
- For client script .NET used JavaScript. WebUIValidation.js file is used for client validation by .NET

### **Validation Controls in ASP.NET**

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

- o RegularExpressionValidator Control
- o CustomValidator Control
- o ValidationSummary

The below table describes the controls and their work:

Validation Control	Description
requiredFieldValidation	Makes an input control a required field.
compareValidator	Compares the value of one input control to the value of another input control or to a fixed value.
rangeValidator	Checks that the user enters a value that falls between two values.
RegularExpressionValidator	Ensures that the value of an input control matches a specified pattern.
CustomValidator	Allows you to write a method to handle the validation of the value entered.
ValidationSummary	Displays a report of all validation errors occurred in a Web page.

- All validation controls are rendered in form as <span> (label are referred as <span> on client by server)

#### Important points for Validation Controls:

- ControlToValidate property is mandatory to all validate controls.
- One validation control will validate only one input control but multiple validate control can be assigned to a input control.

#### Validation Properties:

- Usually, Validation is invoked in response to user actions like clicking submit button or entering data. Suppose, you wish to perform validation on page when user clicks submit button.
- Server validation will only perform when CauseValidation is set to true.
- When the value of the CausesValidation property is set to true, you can also use the ValidationGroup property to specify the name of the validation group for which the Button control causes validation.
- Page has a Validate() method. If it is true this methods is executed. Validate() executes each validation control.
- To make this happen, simply set the CauseValidation property to true for submit button as shown below:

```
<asp:Button ID="Button2" runat="server" Text="Submit" CausesValidation=true />
```

Introduction to ASP.NET

- Lets understand validation controls one by one with practical demonstration:

### **Required Field Validation Control:**

- The RequiredFieldValidator control is simple validation control, which checks to see if the data is entered for the input control. You can have a RequiredFieldValidator control for each form element on which you wish to enforce Mandatory Field rule.
- ```
<asp:RequiredFieldValidator ID="RequiredFieldValidator3" runat="server"
    Style="top: 98px; left: 367px; position: absolute; height: 26px; width: 162px"
    ErrorMessage="password required" ControlToValidate="TextBox2">
</asp:RequiredFieldValidator>
```

Compare Validator Control:

- The Compare Validator control allows you to make comparison to compare data entered in an input control with a constant value or a value in a different control
- It can most commonly be used when you need to confirm password entered by the user at the registration time. The data is always case sensitive.

```
<asp:RequiredFieldValidator ID="RequiredFieldValidator2" runat="server"
    Style="top: 145px;
    left: 367px; position: absolute; height: 26px; width: 162px"
    ErrorMessage="password required"
    ControlToValidate="TextBox3"></asp:RequiredFieldValidator>
```

Range Validator Control:

- The RangeValidator Server Control is another validator control, which checks to see if a control value is within a valid range. The attributes that are necessary to this control are: MaximumValue, MinimumValue, and Type.

```
<asp:RangeValidator ID="RangeValidator1" runat="server"
    Style="top: 194px; left: 365px; position: absolute; height: 22px; width: 105px"
    ErrorMessage="RangeValidator" ControlToValidate="TextBox4"
    MaximumValue="100" MinimumValue="18" Type="Integer"></asp:RangeValidator>
```

RegularExpressionValidator Control:

- A regular expression is a powerful pattern matching language that can be used to identify simple and complex characters sequence that would otherwise require writing code to perform.
- Using RegularExpressionValidator server control, you can check a user's input based on a pattern that you define using a regular expression.
- It is used to validate complex expressions. These expressions can be phone number, email address, zip code and many more. Using Regular Expression Validator is very simple. Simply set the ValidationExpression property to any type of expression you want and it will validate it.
- If you don't find your desired regular expression, you can create your custom one.

In the example I have checked the email id format:

```
<asp:RegularExpressionValidator ID="RegularExpressionValidator1"
    runat="server" Style="top: 234px;
    left: 366px; position: absolute; height: 22px; width: 177px"
    ErrorMessage="RegularExpressionValidator" ControlToValidate="TextBox5"
    ValidationExpression="\w+([-.\'])\w+@\w+([-.\'])\w+\.\w
    +([-.\'])\w+*">></asp:RegularExpressionValidator>
```

The complete code for the above 4 controls is as:

Default.aspx Design:

The screenshot shows a form with five input fields and their corresponding validation messages:

- Enter your name:** An empty text box with the message "name is mandatory" positioned to its right.
- Password:** An empty text box with the message "password required" positioned to its right.
- Confirm Password:** An empty text box with the message "password required" positioned to its right, followed by the text "CompareValidator".
- Enter your age:** An empty text box with the message "Range Validator" positioned to its right.
- Enter your email id:** An empty text box with the message "RegularExpressionValidator" positioned to its right.

At the bottom center of the form is a **Submit** button.

Default.aspx Source code

```
<%@ 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>
            <asp:Label ID="Label3" runat="server" Style="top: 241px; left:
                70px; position: absolute;
                height: 22px; width: 128px; bottom: 282px;" Text="Enter your
                email id:"></asp:Label>
            <asp:Label ID="Label1" runat="server" Style="top: 54px;
                left: 74px; position: absolute;
                height: 22px; width: 128px; bottom: 282px;" Text="Name:></asp:Label>
            <asp:Textbox ID="TextBox1" runat="server" Style="top: 54px;
                left: 142px; position: absolute;
                height: 22px; width: 128px; bottom: 282px;" Text=""/>
            <asp:Label ID="Label2" runat="server" Style="top: 106px; left:
                70px; position: absolute;
                height: 22px; width: 128px; bottom: 282px;" Text="Age:></asp:Label>
            <asp:Textbox ID="TextBox2" runat="server" Style="top: 106px;
                left: 142px; position: absolute;
                height: 22px; width: 128px; bottom: 282px;" Text=""/>
            <asp:Label ID="Label4" runat="server" Style="top: 158px; left:
                70px; position: absolute;
                height: 22px; width: 128px; bottom: 282px;" Text="Email ID:></asp:Label>
            <asp:Textbox ID="TextBox3" runat="server" Style="top: 158px;
                left: 142px; position: absolute;
                height: 22px; width: 128px; bottom: 282px;" Text=""/>
            <asp:Label ID="Label5" runat="server" Style="top: 210px; left:
                70px; position: absolute;
                height: 22px; width: 128px; bottom: 282px;" Text="Password:></asp:Label>
            <asp:Textbox ID="TextBox4" runat="server" Style="top: 210px;
                left: 142px; position: absolute;
                height: 22px; width: 128px; bottom: 282px;" Text=""/>
            <asp:Label ID="Label6" runat="server" Style="top: 262px; left:
                70px; position: absolute;
                height: 22px; width: 128px; bottom: 282px;" Text="Confirm Password:></asp:Label>
            <asp:Textbox ID="TextBox5" runat="server" Style="top: 262px;
                left: 142px; position: absolute;
                height: 22px; width: 128px; bottom: 282px;" Text=""/>
            <asp:Label ID="Label7" runat="server" Style="top: 314px; left:
                70px; position: absolute;
                height: 22px; width: 128px; bottom: 282px;" Text="Age Range:></asp:Label>
            <asp:Textbox ID="TextBox6" runat="server" Style="top: 314px;
                left: 142px; position: absolute;
                height: 22px; width: 128px; bottom: 282px;" Text=""/>
            <asp:Label ID="Label8" runat="server" Style="top: 366px; left:
                70px; position: absolute;
                height: 22px; width: 128px; bottom: 282px;" Text="Email Format:></asp:Label>
            <asp:Textbox ID="TextBox7" runat="server" Style="top: 366px;
                left: 142px; position: absolute;
                height: 22px; width: 128px; bottom: 282px;" Text=""/>
            <asp:Label ID="Label9" runat="server" Style="top: 418px; left:
                70px; position: absolute;
                height: 22px; width: 128px; bottom: 282px;" Text="Submit:></asp:Label>
            <asp:Button ID="Button1" runat="server" Text="Submit" OnClick="Button1_Click"/>
        </div>
    </form>
</body>
```

Lesson 10: Validation

```
height: 22px; width: 128px" Text="Enter your name:"></asp:Label>
<asp:TextBox ID="TextBox1" runat="server" Style="top: 54px; left: 221px; position: absolute; height: 22px; width: 128px; right: 396px;"></asp:TextBox>
<asp:RequiredFieldValidator ID="RequiredFieldValidator1" runat="server" Style="top: 56px; left: 378px; position: absolute; height: 22px; width: 128px" ErrorMessage="RequiredFieldValidator" ControlToValidate="TextBox1">name is mandatory </asp:RequiredFieldValidator>
</div>
<p>
    <asp:Button ID="Button1" runat="server" Style="top: 311px; left: 267px; position: absolute; height: 26px; width: 61px" Text="Submit" />
</p>
<asp:TextBox ID="TextBox3" runat="server" Style="top: 145px; left: 217px; position: absolute; height: 22px; width: 131px" TextMode="Password"></asp:TextBox>
<p>
    <asp:TextBox ID="TextBox2" runat="server" Style="top: 101px; left: 218px; position: absolute; height: 22px; width: 131px" TextMode="Password"></asp:TextBox>
    <asp:Label ID="Label4" runat="server" Style="top: 105px; left: 74px; position: absolute; height: 22px; width: 128px" Text="Password"></asp:Label>
    <asp:TextBox ID="TextBox5" runat="server" Style="top: 239px; left: 210px; position: absolute; height: 22px; width: 134px"></asp:TextBox>
</p>
<asp:RequiredFieldValidator ID="RequiredFieldValidator3" runat="server" Style="top: 380px; left: 367px; position: absolute; height: 26px; width: 162px" ErrorMessage="password required" ControlToValidate="TextBox2"></asp:RequiredFieldValidator>
<asp:RequiredFieldValidator ID="RequiredFieldValidator2" runat="server" Style="top: 145px; left: 367px; position: absolute; height: 26px; width: 162px" ErrorMessage="password required" ControlToValidate="TextBox3"></asp:RequiredFieldValidator>
<asp:CompareValidator ID="CompareValidator1" runat="server" Style="top: 149px; left: 512px; width: 128px" ControlToCompare="TextBox2" ControlToValidate="TextBox3" Operator="LessThanEqual" Type="Text" Value="100" />
```

```

    position: absolute; height: 26px; width: 162px"
                           ErrorMessage="CompareValidator"
ControlToValidate="TextBox3"
                           ValueToCompare="hello"></asp:CompareValidator>
<p>
<asp:Label ID="Label5" runat="server"
           Style="top: 148px; left: 71px; position: absolute;
           height: 22px; width: 128px; bottom: 375px;"
           Text="Confirm Password"></asp:Label>
<asp:TextBox ID="TextBox4" runat="server"
           Style="top: 194px; left: 212px; position: absolute;
           height: 22px; width: 140px"></asp:TextBox>
<asp:Label ID="Label6" runat="server"
           Style="top: 194px; left: 71px; position: absolute;
           height: 22px; width: 128px; bottom: 329px;"
           Text="Enter your age:"></asp:Label>
</p>
<asp:RangeValidator ID="RangeValidator1" runat="server"
           Style="top: 194px; left: 365px;
           position: absolute; height: 22px; width: 105px"
           ErrorMessage="RangeValidator"
ControlToValidate="TextBox4" MaximumValue="100" MinimumValue="18"
           Type="Integer"></asp:RangeValidator>
<asp:RegularExpressionValidator ID="RegularExpressionValidator1"
           runat="server" Style="top: 234px;
           left: 366px; position: absolute; height: 22px; width: 177px"
           ErrorMessage="RegularExpressionValidator"
           ControlToValidate="TextBox5"
ValidationExpression="\w+([-.\'])\w+)*@\w+
+([-.\'])\w+*\.\w+([-.\'])\w+*"></asp:RegularExpressionValidator>
</form>
</body>
</html>

```

CustomValidator Control

- You can solve your purpose with ASP.NET validation control. But if you still don't find solution you can create your own custom validator control.
- The CustomValidator Control can be used on client side and server side. JavaScript is used to do client validation and you can use any .NET language to do server side validation.
- I will explain you CustomValidator using server side. You should rely more on server side validation.

- To write CustomValidator on server side you override ServerValidate event.

User ID: User id required User ID should have atleast a capital, small and digit and should be greater than 5 and less than 26 letters

Source Code:

```
<%@ 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>
            <asp:Label ID="Label1" runat="server" Text="User ID:"></asp:Label>
            <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
            <asp:RequiredFieldValidator ID="RequiredFieldValidator1" runat="server"
                ControlToValidate="TextBox1"
                ErrorMessage="User id required"></asp:RequiredFieldValidator>
            <asp:CustomValidator ID="CustomValidator1" runat="server"
                OnServerValidate="UserCustomValidate"
                ControlToValidate="TextBox1"
                ErrorMessage="User ID should have atleast a capital,
                small and digit and should be greater than
                5 and less than 26 letters"
                SetFocusOnError="True"></asp:CustomValidator>
        </div>
        <asp:Button ID="Button1" runat="server"
            onclick="Button1_Click" Text="Submit" />
    </form>
</body>
</html>
```

Code behind file

```
using System;
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 _Default : System.Web.UI.Page
{
    protected void UserCustomValidate(object source,
                                      ServerValidateEventArgs args)
    {
        string str = args.Value;
        args.IsValid = false;
        //checking for input length greater than 6 and less than 25 characters
        if (str.Length < 6 || str.Length > 25)
        {
            return;
        }
        //checking for atleast a single capital letter
        bool capital = false;
        foreach (char ch in str)
        {
            if (ch >= 'A' && ch <= 'Z')
            {
                capital = true;
                break;
            }
        }
        if (!capital)
        {
            return;
        }
        //checking for atleast a single lower letter
        bool lower = false;
        foreach (char ch in str)
        {
            if (ch >= 'a' && ch <= 'z')
            {
                lower = true;
                break;
            }
        }
    }
}
```

```

        if (!lower)
        {
            return;
        }
        bool digit = false;
        foreach (char ch in str)
        {
            if (ch >= '0' && ch <= '9')
            {
                digit = true;
                break;
            }
        }
        if (!digit)
        {
            return;
        }
        args.IsValid = true;
    }
}
protected void Page_Load(object sender, EventArgs e)
{
}
protected void Button1_Click(object sender, EventArgs e)
{
}
}
}

```

User ID:

User ID should have atleast a capital, small and digit and should be greater than 5 and less than 16 letters

- The server side validation you write does not need to provide the exact same validation as that of the client side validation. The client side validation can check for the user input data for range and type and server side validation can check for matching of data with database. Both server side and client side validation can be used for total solution.

ValidationSummary:

- ASP.NET has provided an additional control that complements the validator controls.
- The ValidationSummary control is reporting control, which is used by the other validation controls on a page.
- You can use this validation control to consolidate errors reporting for all the validation errors that occur on a page instead of leaving this up to each and every individual validation control.

- The validation summary control will collect all the error messages of all the non-valid controls and put them in a tidy list.
- ```
<asp:ValidationSummary ID="ValidationSummary1" runat="server"
 style="top: 390px; left: 44px; position: absolute; height: 38px; width:
 625px" />
```
- Both ErrorMessage and Text properties are used to display error messages. Text error message have precedence.
  - If you are using ValidationSummary than only ErrorMessage and Text property is used.
  - The complete code for the above ValidationSummary is as:

**Default.aspx Design**

Enter your name:  \*

Enter Password:  \*

Confirm Password:  \*

Enter your Age:  \*

\* Error message 1.  
\* Error message 2.

**Default.aspx Source code**

```
<%@ 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>
 <asp:Label ID="Label1" runat="server" Style="top: 239px;
 left: 75px; position: absolute;
 height: 22px; width: 128px" Text="Enter your
 Age:"></asp:Label>
```

```
<asp:Label ID="Label2" runat="server" Style="top: 94px;
 left: 81px; position: absolute;
 height: 22px; width: 128px" Text="Enter your name:"></asp:Label>
</div>
<asp:TextBox ID="TextBox1" runat="server" Style="top: 95px;
 left: 250px; position: absolute;
 height: 22px; width: 128px"></asp:TextBox>
<p>
 <asp:TextBox ID="TextBox4" runat="server" Style="top: 195px;
 left: 249px; position: absolute;
 height: 22px; width: 127px"></asp:TextBox>
</p>
<p>
 <asp:Label ID="Label3" runat="server" Style="top: 148px;
 left: 76px; position: absolute;
 height: 22px; width: 128px" Text="Enter Password:"></asp:Label>
</p>
<p>
 <asp:TextBox ID="TextBox3" runat="server" Style="top: 146px;
 left: 249px; position: absolute;
 height: 22px; width: 127px" TextMode="Password"></asp:TextBox>
</p>
<p>
 <asp:Label ID="Label4" runat="server" Style="top: 197px;
 left: 75px; position: absolute;
 height: 22px; width: 128px" Text="Confirm Password:"></asp:Label>
</p>
<asp:TextBox ID="TextBox2" runat="server" Style="top: 236px;
 left: 250px; position: absolute;
 height: 22px; width: 127px" TextMode="Password"></asp:TextBox>
<asp:CompareValidator ID="CompareValidator1" runat
 ="server" Style="top: 197px; left: 522px;
 position: absolute; height: 22px;
 width: 17px" ErrorMessage="CompareValidator"
ControlToCompare="TextBox2"
ControlToValidate="TextBox3">*</asp:CompareValidator>
<p>
 <asp:Button ID="Button1" runat="server" Style="top: 333px;
 left: 248px; position: absolute;
 height: 26px; width: 56px" Text="Submit" />
 <asp:RequiredFieldValidator ID="RequiredFieldValidator1"
 runat="server" Style="top: 196px;
```

Introduction to ASP.NET

4.100

```

 left: 393px; position: absolute; height: 22px; width: 22px"
 ErrorMessage="Confirm Password mandatory & should match password"
 ControlToValidate="TextBox3">*</asp:RequiredFieldValidator>
<asp:RangeValidator ID="RangeValidator1" runat="server"
 Style="top: 235px; left: 388px;
 position: absolute; height: 22px; width: 156px;
 bottom: 288px;" ErrorMessage="age between 18-100"
 ControlToValidate="TextBox4" MaximumValue="100"
 MinimumValue="18" Type="Integer">*</asp:RangeValidator>
<asp:RequiredFieldValidator ID="RequiredFieldValidator2"
 runat="server" Style="top: 92px;
 left: 393px; position: absolute; height: 22px;
 width: 156px" ErrorMessage="Name is required"
 ControlToValidate="TextBox1">*</asp:RequiredFieldValidator>
<asp:RequiredFieldValidator ID="RequiredFieldValidator3"
 runat="server" Style="top: 146px;
 left: 391px; position: absolute; height: 22px;
 width: 156px" ErrorMessage="Password mandatory"
 ControlToValidate="TextBox2">*</asp:RequiredFieldValidator>
</p>
<asp:ValidationSummary ID="ValidationSummary1"
 runat="server" Style="top: 390px;
 left: 44px; position: absolute; height: 38px; width: 625px" />
</form>
</body>
</html>
```

### Output of ValidationSummary program

Enter your name:	<input type="text"/>
Enter Password:	<input type="password"/>
Confirm Password:	<input type="password"/>
Enter your Age:	<input type="text"/>
<input type="button" value="Submit"/>	
<ul style="list-style-type: none"> <li>* Confirm Password mandatory &amp; should match password</li> <li>* Name is required</li> <li>* Password mandatory</li> </ul>	

### 4.10.6 Master Page In Asp.net

- ASP.NET master pages allow you to create a consistent layout for the pages in your application. A single master page defines the look and feel and standard behavior that you want for all of the pages (or a group of pages) in your application. You can then create individual content pages that contain the content you want to display. When users request the content pages, they merge with the master page to produce output that combines the layout of the master page with the content from the content page.

#### MasterPage:

- The extension of MasterPage is '.master'.
- MasterPage cannot be directly accessed from the client because it just acts as a template for the other Content Pages.
- In a MasterPage we can have content either inside ContentPlaceHolder or outside it. Only content inside the ContentPlaceHolder can be customized in the Content Page.
- We can have multiple masters in one web application.
- A MasterPage can have another MasterPage as Master to it.
- The content page content can be placed only inside the content tag.
- Controls of MasterPage can be programmed in the MasterPage and content page but a content page control will never be programmed in MasterPage.
- A master page of one web application cannot be used in another web application.
- The MasterPageFile property of a webform can be set dynamically and it should be done either in or before the Page\_PreInit event of the WebForm. Page.MasterPageFile = "MasterPage.master". The dynamically set Master Page must have the ContentPlaceHolder whose content has been customized in the WebForm.
- The order in which events are raised: Load (Page) a Load (Master) a LoadComplete (Page) i.e. if we want to overwrite something already done in Load event handler of Master then it should be coded in the LoadComplete event of the page.
- Page\_Load is the name of method for event handler for Load event of Master. (it's not Master\_Load).

#### How Master Pages Work?

- A master page is an ASP.NET file with the extension .master (for example, MySite.master) with a predefined layout that can include static text, HTML elements, and server controls. The master page is identified by a special @ Master directive that looks like the following.
- ```
<%@ Master Language="VB" %>
```

The @ Master directive can contain most of the same directives that a @ Control directive can contain. For example, the following master-page directive includes the name of a code-behind file, and assigns a class name to the master page.

```
<%@ Master Language="VB" CodeFile="MasterPage.master.vb"
```

In addition to the @ Master directive, the master page also contains all of the top-level HTML elements for a page, such as html, head, and form. For example, on a master page you might use an HTML table for the layout, an img element for your company logo, static text for the copyright notice, and server controls to create standard navigation for your site. You can use any HTML and any ASP.NET elements as part of your master page.

Inherits="MasterPage" %>

Replaceable Content Placeholders:

In addition to static text and controls that will appear on all pages, the master page also includes one or more ContentPlaceholder controls. These placeholder controls define regions where replaceable content will appear. In turn, the replaceable content is defined in content pages. After you have defined the ContentPlaceholder controls, a master page might look like the following.

```
<%@ Master Language="VB" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML
1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server" >
    <title>Master page title</title>
</head>
<body>
    <form id="form1" runat="server">
        <table>
            <tr>
                <td><asp:contentplaceholder id="Main" runat="server"/></td>
                <td><asp:contentplaceholder id="Footer" runat="server" /></td>
            </tr>
        </table>
    </form>
</body>
</html>
```

Content Pages:

You define the content for the master page's placeholder controls by creating individual content pages, which are ASP.NET pages (.aspx files and, optionally, code-behind files) that are bound to a specific master page. The binding is established in the content page's @ Page directive by including a MasterPageFile attribute that

points to the master page to be used. For example, a content page might have the following @ Page directive, which binds it to the Master1.master page.

```
<%@ Page Language="VB" MasterPageFile="~/MasterPages/Master1.master"
Title="Content Page" %>
```

- In the content page, you create the content by adding Content controls and mapping them to ContentPlaceholder controls on the master page. For example, the master page might have content placeholders called Main and Footer. In the content page, you can create two Content controls, one that is mapped to the ContentPlaceholder control Main and the other mapped to the ContentPlaceholder control Footer, as shown in the following figure.

Replacing placeholder content

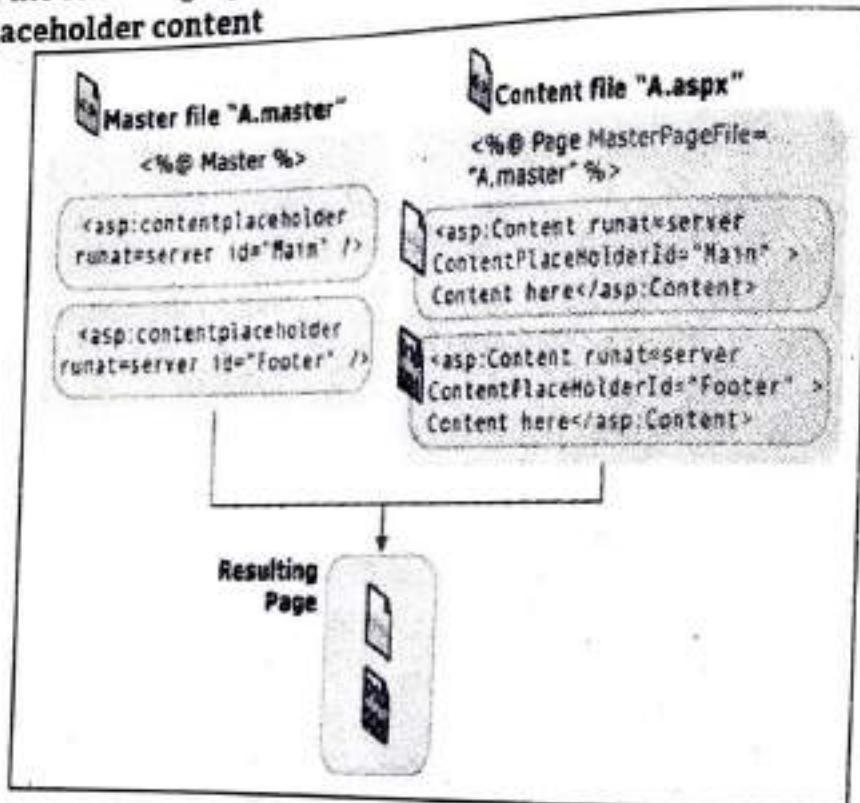


Fig. 4.35

- After creating Content controls, you add text and controls to them. In a content page, anything that is not inside the Content controls (except script blocks for server code) results in an error. You can perform any tasks in a content page that you do in an ASP.NET page. For example, you can generate content for a Content control using server controls and database queries or other dynamic mechanisms.
- A content page might look like the following.

```
<% @ Page Language="VB" MasterPageFile="~/Master.master" Title="Content
Page 1" %>
<asp:Content ID="Content1" ContentPlaceholderID="Main" Runat="Server">
    Main content.
</asp:Content>
<asp:Content ID="Content2" ContentPlaceholderID="Footer" Runat="Server">
    Footer content.
</asp:Content>
```

- The @ Page directive binds the content page to a specific master page, and it defines a title for the page that will be merged into the master page. Note that the content page contains no other markup outside of the Content controls. (The master page must contain a head element with the attribute runat="server" so that the title setting can be merged at run time.)
- You can create multiple master pages to define different layouts for different parts of your site, and a different set of content pages for each master page.

Run-time Behavior of Master Pages:

- At run time, master pages are handled in the following sequence:
 - Users request a page by typing the URL of the content page.
 - When the page is fetched, the @ Page directive is read. If the directive references a master page, the master page is read as well. If this is the first time the pages have been requested, both pages are compiled.
 - The master page with the updated content is merged into the control tree of the content page.
 - The content of individual Content controls is merged into the corresponding ContentPlaceholder control in the master page.
 - The resulting merged page is rendered to the browser.

- The process is illustrated in the following diagram.

Master pages at run time:

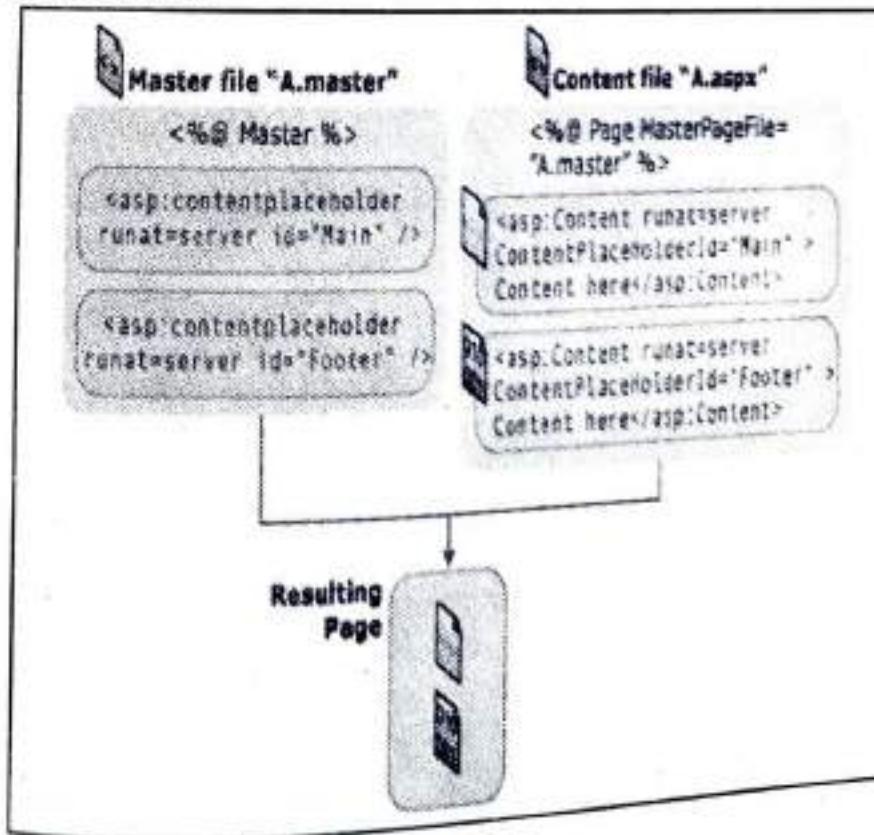


Fig. 4.36

Advantages of Master Pages:

- Master pages provide functionality that developers have traditionally created by copying existing code, text, and control elements repeatedly; using framesets; using include files for common elements; using ASP.NET user controls; and so on.
- Advantages of master pages include the following:
 - They allow you to centralize the common functionality of your pages so that you can make updates in just one place.
 - They make it easy to create one set of controls and code and apply the results to a set of pages. For example, you can use controls on the master page to create a menu that applies to all pages.
 - They give you fine-grained control over the layout of the final page by allowing you to control how the placeholder controls are rendered.
 - They provide an object model that allows you to customize the master page from individual content pages.

4.10.7 State Management in ASP.NET

- In This article does an overview of state management techniques in ASP.NET. we will be discussing about the various types of state management techniques both client side and server side. This article goes over client side methods like hidden field, view state, cookies, and server side aspects like session management, application state, and session events in ASP.NET.
- State management means to preserve state of a control, web page, object/data, and user in the application explicitly because all ASP.NET web applications are stateless, i.e., by default, for each page posted to the server, the state of controls is lost. Nowadays all web apps demand a high level of state management from control to application level.
- 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.
- ASP.NET manages four types of states:
 - View State
 - Control State
 - Session State
 - Application State

The view state is 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.

contd. ...

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 6:

- 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 7:
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:

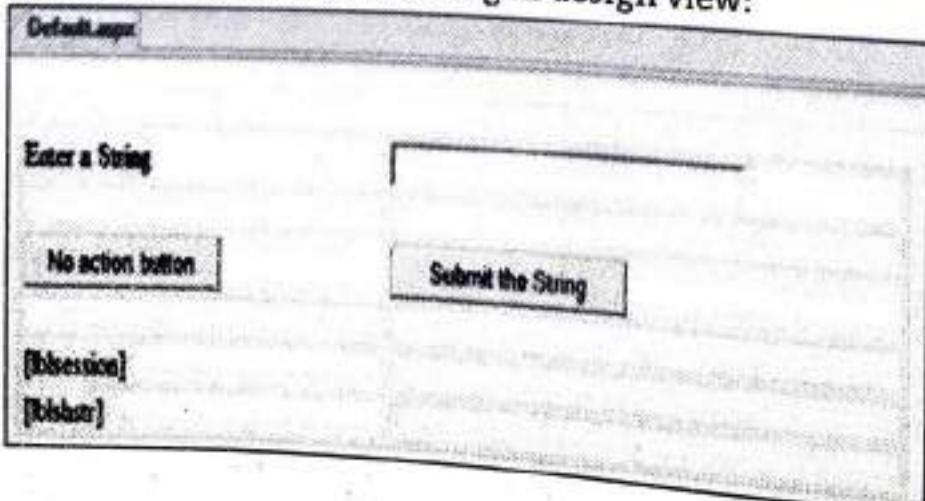
```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"
Inherits="_Default" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
    <head runat="server">
        <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>

```

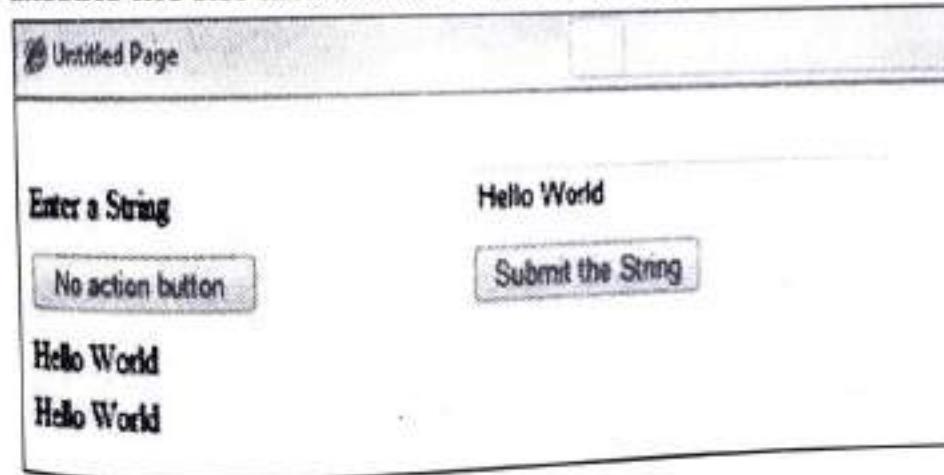
- It should look like the following in design view:



- 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:



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
<code>Item(name)</code>	The value of the application state item with the specified name. This is the default property of the <code>HttpApplicationState</code> class.
<code>Count</code>	The number of items in the application state collection.

The `HttpApplicationState` class has the following methods:

Methods	Description
<code>Add(name, value)</code>	Adds an item to the application state collection.
<code>Clear</code>	Removes all the items from the application state collection.
<code>Remove(name)</code>	Removes the specified item from the application state collection.
<code>RemoveAll</code>	Removes all objects from an <code>HttpApplicationState</code> collection.
<code>RemoveAt</code>	Removes an <code>HttpApplicationState</code> object from a collection by index.
<code>Lock()</code>	Locks the application state collection so only the current user can access it.
<code>Unlock()</code>	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.";
```
- This article is all about how to maintain, clear or hold the states of your pages in ASP.NET applications. In this article I tried to briefly summarize the concept of State Management but I'll include Client-Side State Management only.

Agenda:

- The agenda of this article will be as follows:

- State Overview
- State Introduction
- State Outline
- State Management Types
- Client-side Management
- Server-side Management
- State Management Scenario
- State Management Techniques
- Client-side Techniques
- View
- Hidden
- Cookies
- Control State
- Query Strings
- Server-side Techniques
- Session State
- Application State

State Overview:

- As we all know, browsers are generally stateless.
- Now the question arises here, what does stateless actually mean?
- Stateless means, whenever we visit a website, our browser communicates with the respective server depending on our requested functionality or the request. The browser communicates with the respective server using the HTTP or HTTPS protocol.
- But after that response, what's next or what will happen when we visit that website again after closing our web browser?
- In this case HTTP/HTTPS doesn't remember what website or URL we visited, or in other words we can say it doesn't hold the state of a previous website that we visited before closing our browser, that is called stateless.
- Now I guess you have at least an idea of what state and stateless actually means.
- So our browsers are stateless.

State Introduction:

- In this article I'll try to give you a feel of state and why we need states and State Management in ASP.NET. I'll take you through several State Management techniques along with their respective general case examples.

State Outline:

- As I said in the beginning, HTTP is a stateless protocol. It just cleans up or we can say removes all the resources/references that were serving a specific request in the past. These resources can be:
 - Objects
 - Allocated Memory
 - Sessions ID's
 - Some URL info and so on.

State Management Types:

- In ASP.NET there are the following 2 State Management methodologies:

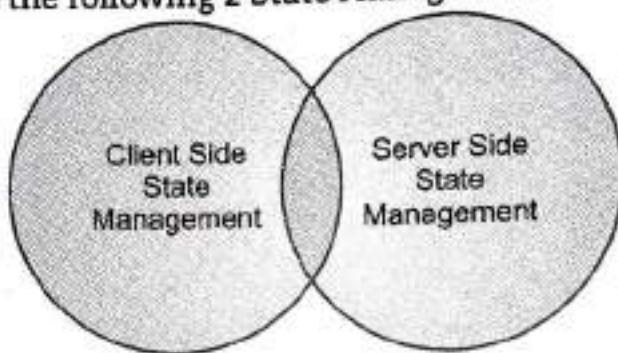


Fig. 4.37

Client-Side State Management:

- Whenever we use Client-Side State Management, the state related information will directly get stored on the client-side. That specific information will travel back and communicate with every request generated by the user then afterwards provides responses after server-side communication.
- This architecture is something like the following:

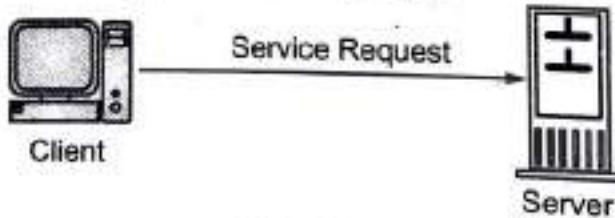


Fig. 4.38

Server-Side State Management:

- Server-Side State Management is different from Client-Side State Management but the operations and working is somewhat the same in functionality. In Server-Side State Management all the information is stored in the user memory. Due to this functionality there is more secure domains at the server side in comparison to Client-Side State Management.

- The structure is something like the following:

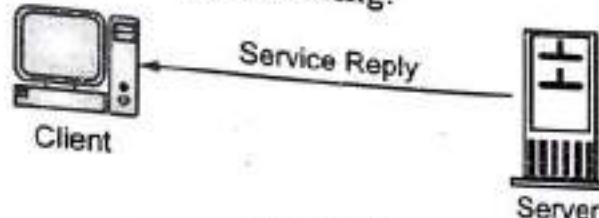


Fig. 4.39

State Management Scenario:

- It will be a little difficult to directly evaluate what will be better for our application. We cannot directly say that we will use client-side or server-side architecture of State Management.

State Management Techniques:

- State Management techniques are based on client side and server side. Their functionality differs depending on the change in state, so here is the hierarchy:

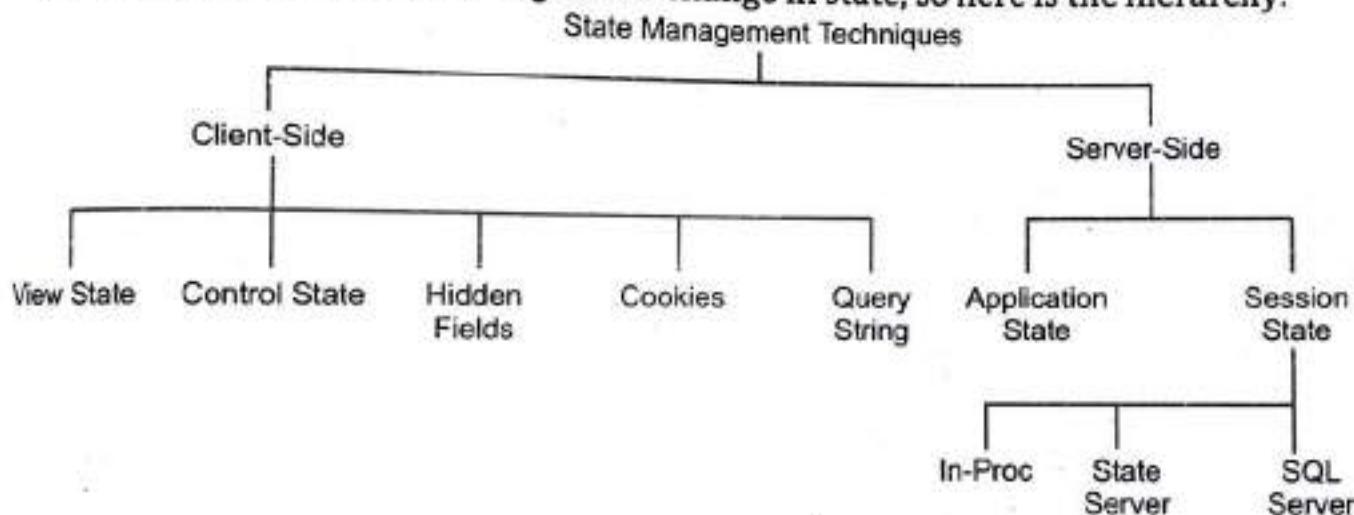


Fig. 4.40

Client-side | Techniques:

- Client-Side State Management techniques are,
 - View State
 - Hidden field
 - Cookies
 - Control State
 - Query Strings

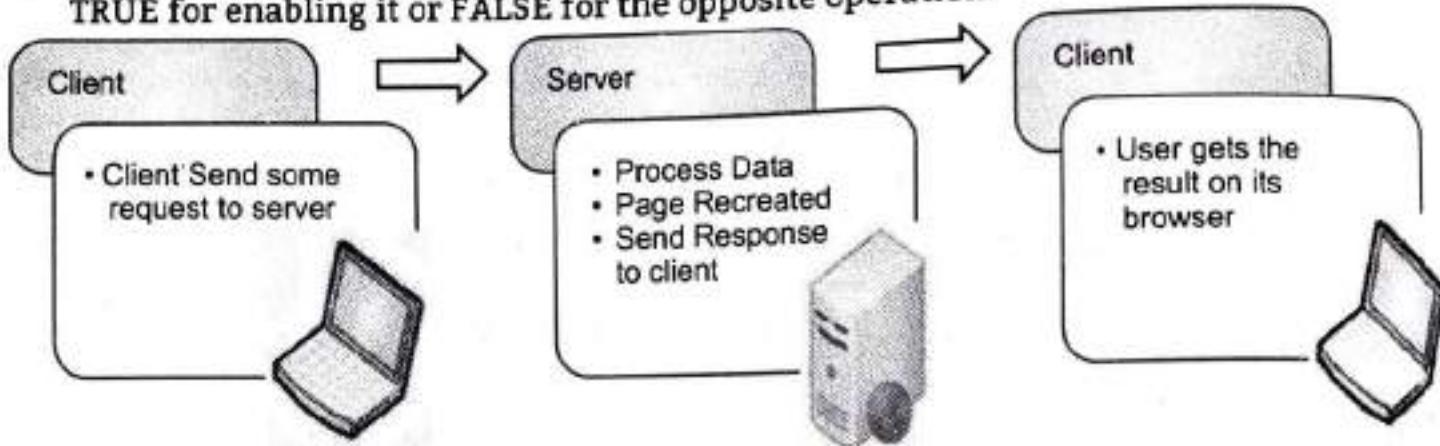
Server-side | Technique:

- Server-Side State Management techniques are:
 - Session State
 - Application State

Now I am defining each and every technique in detail with their reference example.

View State:

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

**Fig. 4.41: View State Management**

```

// Page Load Event
protected void Page_Load(object sender, EventArgs e)
{
    if (IsPostBack)
    {
        if (ViewState["count"] != null)
        {
            int ViewstateVal = Convert.ToInt32(ViewState["count"]) + 1;
            View.Text = ViewstateVal.ToString();
            ViewState["count"] = ViewstateVal.ToString();
        }
        else
        {
            ViewState["count"] = "1";
        }
    }
}
// Click Event
protected void Submit(object sender, EventArgs e)
{
    View.Text = ViewState["count"].ToString();
}

```

Hidden Field:

- A hidden field is used for storing small amounts of data on the client side. In most simple words it's just a container of some objects but their result is not rendered on our web browser. It is invisible in the browser.
- It stores a value for the single variable and it is the preferable way when a variable's value is changed frequently but we don't need to keep track of that every time in our application or web program.

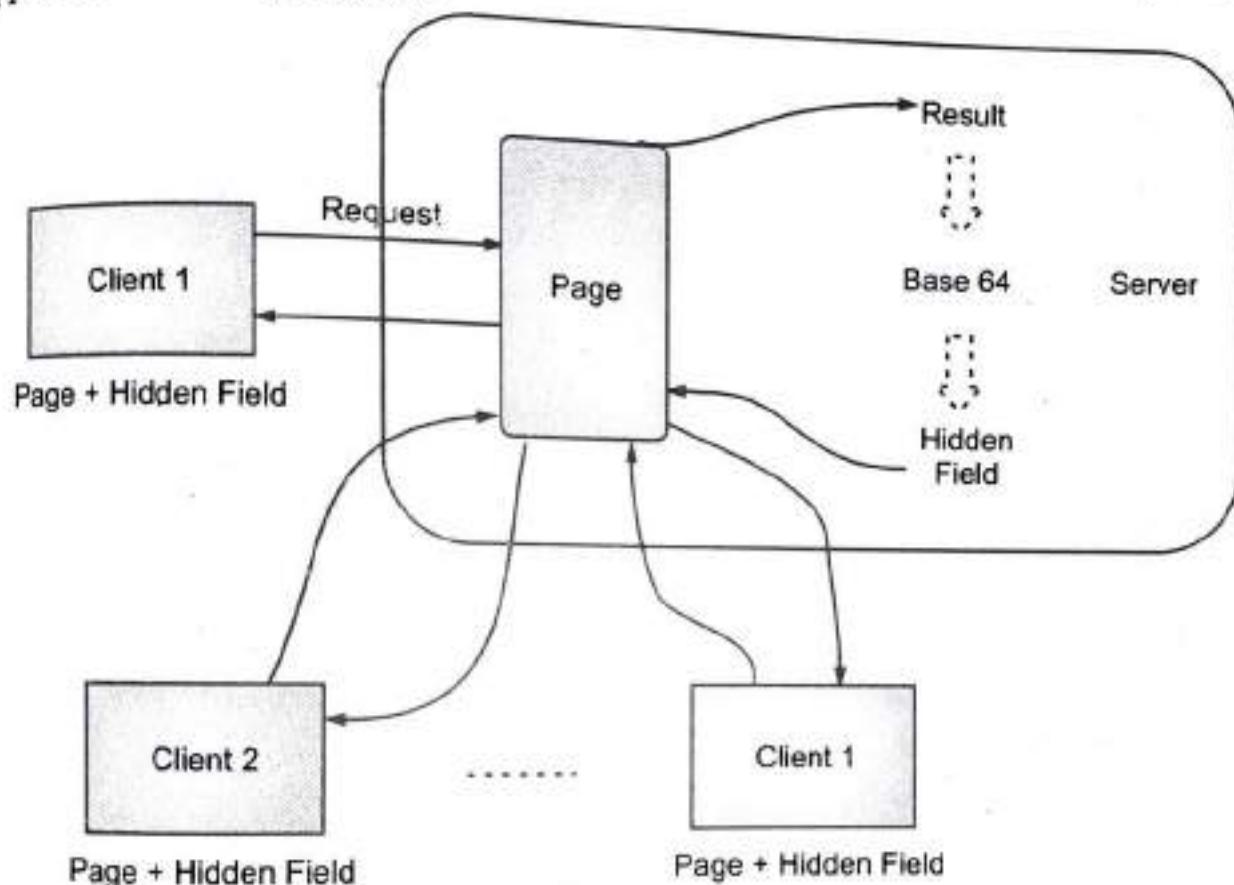
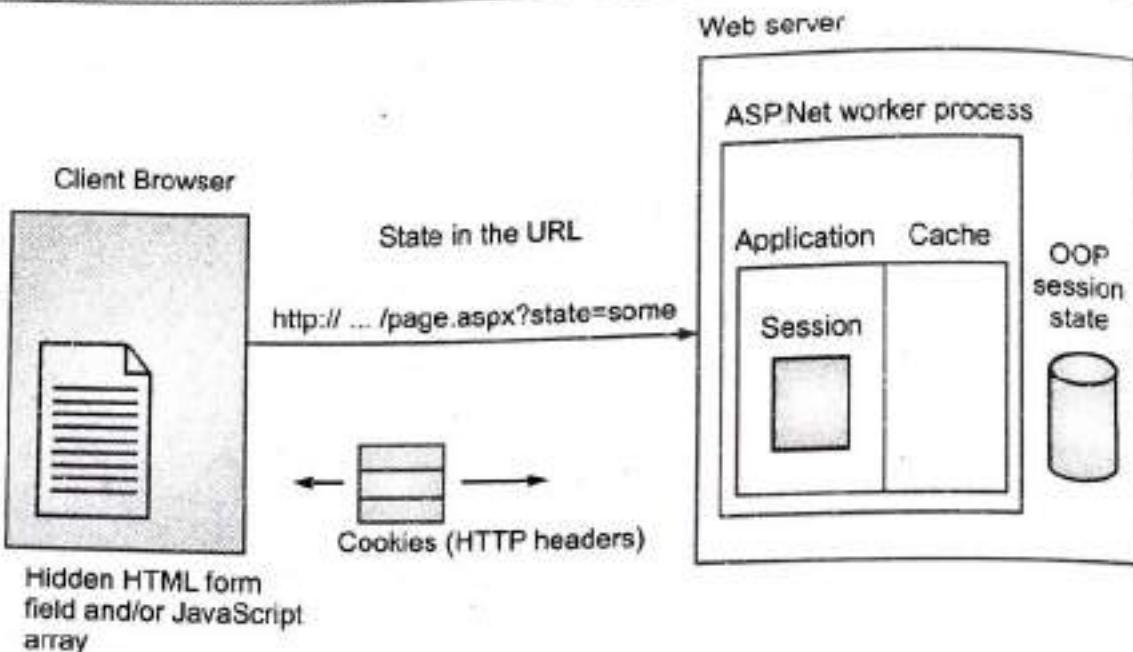


Fig. 4.42: Hidden Field Management

```
// Hidden Field
int newVal = Convert.ToInt32(HiddenField1.Value) + 1;
HiddenField1.Value = newVal.ToString();
Label2.Text = HiddenField1.Value;
```

Cookies:

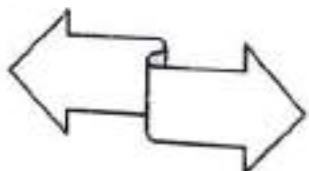
- A set of Cookies is a small text file that is stored in the user's hard drive using the client's browser. Cookies are just used for the sake of the user's identity matching as it only stores information such as sessions id's, some frequent navigation or post-back request objects.
- Whenever we get connected to the internet for accessing a specific service, the cookie file is accessed from our hard drive via our browser for identifying the user. The cookie access depends upon the life cycle or expiration of that specific cookie file.

**Fig. 4.43: Cookie Management**

```

int postbacks = 0;
if (Request.Cookies["number"] != null)
{
    postbacks = Convert.ToInt32(Request.Cookies["number"].Value) + 1;
}
// Generating Response
else
{
    postbacks = 1;
}
Response.Cookies["number"].Value = postbacks.ToString();
Result.Text = Response.Cookies["number"].Value;

```

Cookie | Types:**Fig. 4.44****Persistent Cookie:**

- Cookies having an expiration date are called a persistent cookie. This type of cookie reaches their end as their expiration dates comes to an end. In this cookie we set an expiration date.

```
response.Cookies["UserName"].Value = "Abhishek";
response.Cookies["UserName"].Expires = DateTime.Now.AddDays(1);
HttpCookie aCookie = new HttpCookie("Session");
aCookie.Value = DateTime.Now.ToString();
aCookie.Expires = DateTime.Now.AddDays(1);
response.Cookies.Add(aCookie);
```

Non-Persistent Cookie:

- Non-persistent types of cookies aren't stored in the client's hard drive permanently. It maintains user information as long as the user access or uses the services. Its simply the opposite procedure of a persistent cookie.

```
HttpCookie aCookie = new HttpCookie("Session");
aCookie.Value = DateTime.Now.ToString();
aCookie.Expires = DateTime.Now.AddDays(1);
Response.Cookies.Add(aCookie);
```

Control State:

- Control state is based on the custom control option. For expected results from CONTROL STATE we need to enable the property of view state. As I already described you can manually change those settings.

Points to Remember:

- Some features of query strings are:
 - Used for enabling the View State Property.
 - Defines a custom view.
 - View State property declaration.
 - Can't be modified.
 - Accessed directly or disabled.

Query Strings:

- Query strings are used for some specific purpose. These in a general case are used for holding some value from a different page and move these values to the different page. The information stored in it can be easily navigated to one page to another or to the same page as well.

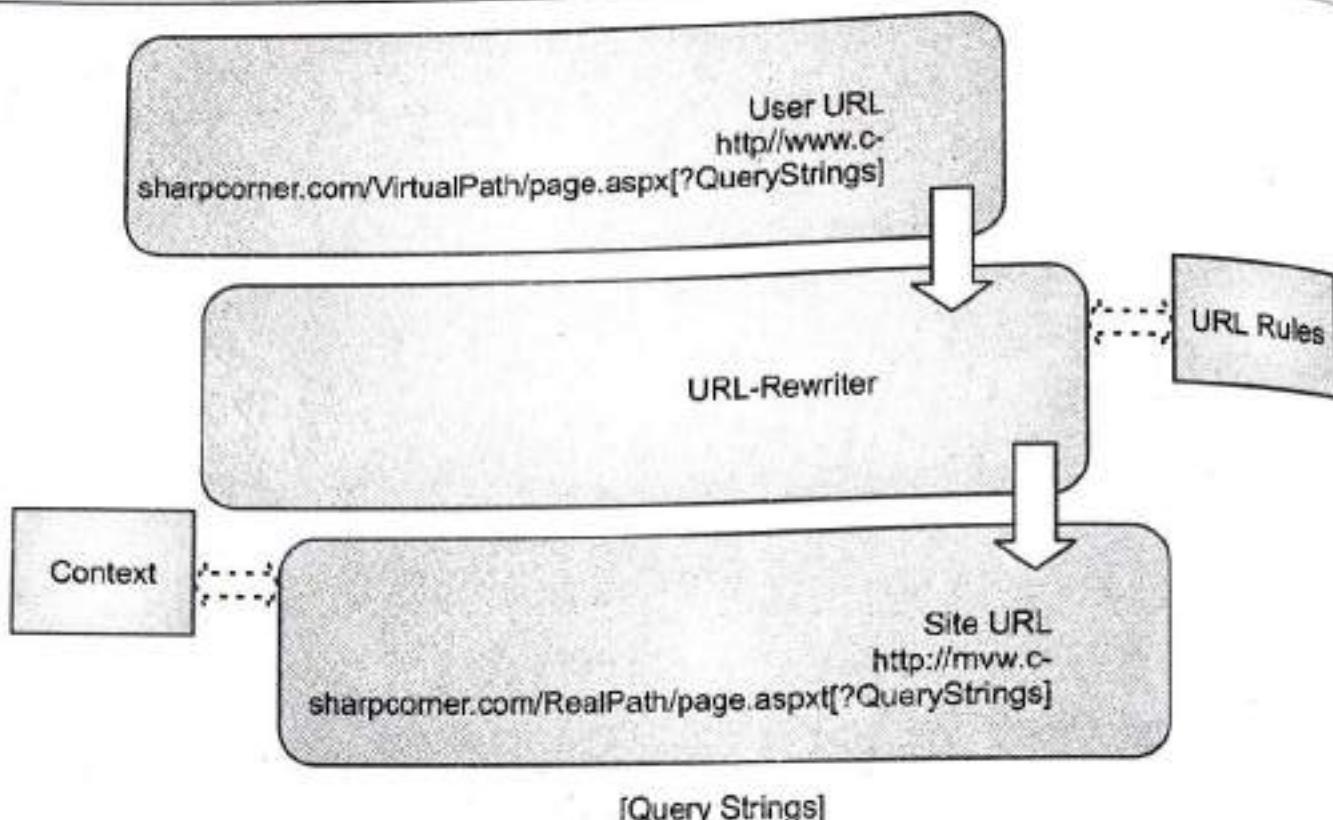


Fig. 4.45: Query Strings

```

// Getting data
if (Request.QueryString["number"] != null)
{
    View.Text = Request.QueryString["number"];
}
// Setting query string
int postbacks = 0;
if (Request.QueryString["number"] != null)
{
    postbacks = Convert.ToInt32(Request.QueryString["number"]) + 1;
}
else
{
    postbacks = 1;
}
Response.Redirect("default.aspx?number=" + postbacks);
    
```

Summary

- The Path property of the HttpRequest class retrieves the virtual path of the current request. The UserHostAddress property of HttpRequest class retrieves the host IP address of the remote client making the request. GetType() also retrieves the type of the current instance.
- Some of the features of view state are:
 - It is page-level State Management.
 - Used for holding data temporarily.

- o Can store any type of data.
- o Property dependent.

Some features of hidden fields are:

- o Contains a small amount of memory.
- o Direct functionality access.

Some features of cookies are:

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

Some of the features are:

- o It is generally used for holding values.
- o Works temporarily.
- o Switches info from one to another page.
- o Increase performance.
- o Uses real and virtual path values for URL routing.

Practice Questions

1. What is ASP.NET? Write Components of ASP.NET.
2. Explain the page life cycle of ASP.NET in detail
3. Explain the architecture of ASP.NET.
4. What are HTML controls? Explain.
5. What are properties and methods of `HTTPRequest Class`?
6. What are the properties and methods of the `Server object`?
7. What are the properties and methods of the `Response Object`?
8. Explain the event handling in ASP.NET.
9. What is web controls?
10. What are the advantages of web controls in ASP.NET?
11. What is server controls in ASP.NET?
12. What are client side controls in ASP.NET?
13. Explain the types of Menu Control.
14. What is validation in ASP.NET?
15. What are the types of states in ASP.NET?



Basics of ADO.NET

Learning Objectives ...

Students will be able to:

- Consistent access to data sources such as SQL Server and XML, and to data sources exposed through OLE DB and ODBC.
- Data-sharing consumer applications can use ADO.NET to connect to these data sources and retrieve, handle, and update the data that they contain.
- Functionality to the developers who write managed code similar to the functionality provided to native component object model (COM) developers by ActiveX Data Objects (ADO). We recommend that you use ADO.NET, not ADO, for accessing data in your .NET applications.
- The most direct method of data access within the .NET Framework.

5.1 WHAT IS ADO.NET ALL ABOUT?

[W-22]

- ADO.NET is a set of classes that expose data access services for .NET Framework programmers. ADO.NET provides a rich set of components for creating distributed, data-sharing applications. It is an integral part of the .NET Framework, providing access to relational, XML, and application data. ADO.NET supports a variety of development needs, including the creation of front-end database clients and middle-tier business objects used by applications, tools, languages, or Internet browsers.
- The .NET Framework includes its own data access technology i.e. **ADO.NET**. ADO.NET is the latest implementation of Microsoft's Universal Data Access strategy. ADO.NET consists of managed classes that allows .NET applications to connect to data sources such as Microsoft SQL Server, Microsoft Access, Oracle, XML, etc., execute commands and manage disconnected data
- In This Chapter we will discover the most important elements of ADO.NET such as...
ADO.NET Overview:
- ADO.NET provides consistent access to data sources such as SQL Server and XML and to data sources exposed through OLE DB and ODBC. Data-sharing consumer applications can use ADO.NET to connect to these data sources and retrieve, handle, and update the data that they contain.

- ADO.NET separates data access from data manipulation into discrete components that can be used separately or in tandem. ADO.NET includes .NET Framework data providers for connecting to a database, executing commands, and retrieving results. Those results are either processed directly, placed in an ADO.NET DataSet object in multiple sources, or passed between tiers. The DataSet object can also be used independently of a .NET Framework data provider to manage data local to the application or sourced from XML.
- The ADO.NET classes are found in System.Data.dll, and are integrated with the XML classes found in System.Xml.dll. For sample code that connects to a database, retrieves data from it, and then displays that data in a console window.
- ADO.NET provides functionality to developers who write managed code similar to the functionality provided to native Component Object Model (COM) developers by ActiveX Data Objects (ADO). We recommend that you use ADO.NET, not ADO, for accessing data in your .NET applications.
- ADO.NET provides the most direct method of data access within the .NET Framework. For a higher-level abstraction that allows applications to work against a conceptual model instead of the underlying storage model, see the ADO.NET Entity Framework.
- Data processing has traditionally relied primarily on a connection-based, two-tier model. As data processing increasingly uses multi-tier architectures, programmers are switching to a disconnected approach to provide better scalability for their applications.

ADO.NET Components:

- The two main components of ADO.NET for accessing and manipulating data are the .NET Framework data providers and the DataSet.
- The .NET Framework Data Providers are components that have been explicitly designed for data manipulation and fast, forward-only, read-only access to data. The **Connection** object provides connectivity to a data source. The **Command** object enables access to database commands to return data, modify data, run stored procedures, and send or retrieve parameter information. The **DataReader** provides a high-performance stream of data from the data source. Finally, the **DataAdapter** provides the bridge between the **DataSet** object and the data source. The **DataAdapter** uses **Command** objects to execute SQL commands at the data source to both load the **DataSet** with data and reconcile changes that were made to the data in the **DataSet** back to the data source.

The DataSet:

- The ADO.NET **DataSet** is explicitly designed for data access independent of any data source. As a result, it can be used with multiple and differing data sources, used with XML data, or used to manage data local to the application. The **DataSet** contains a

collection of one or more **DataTable** objects consisting of rows and columns of data, and also primary key, foreign key, constraint, and relation information about the data in the **DataTable** objects. The following diagram illustrates the relationship between a .NET Framework data provider and a **DataSet**.

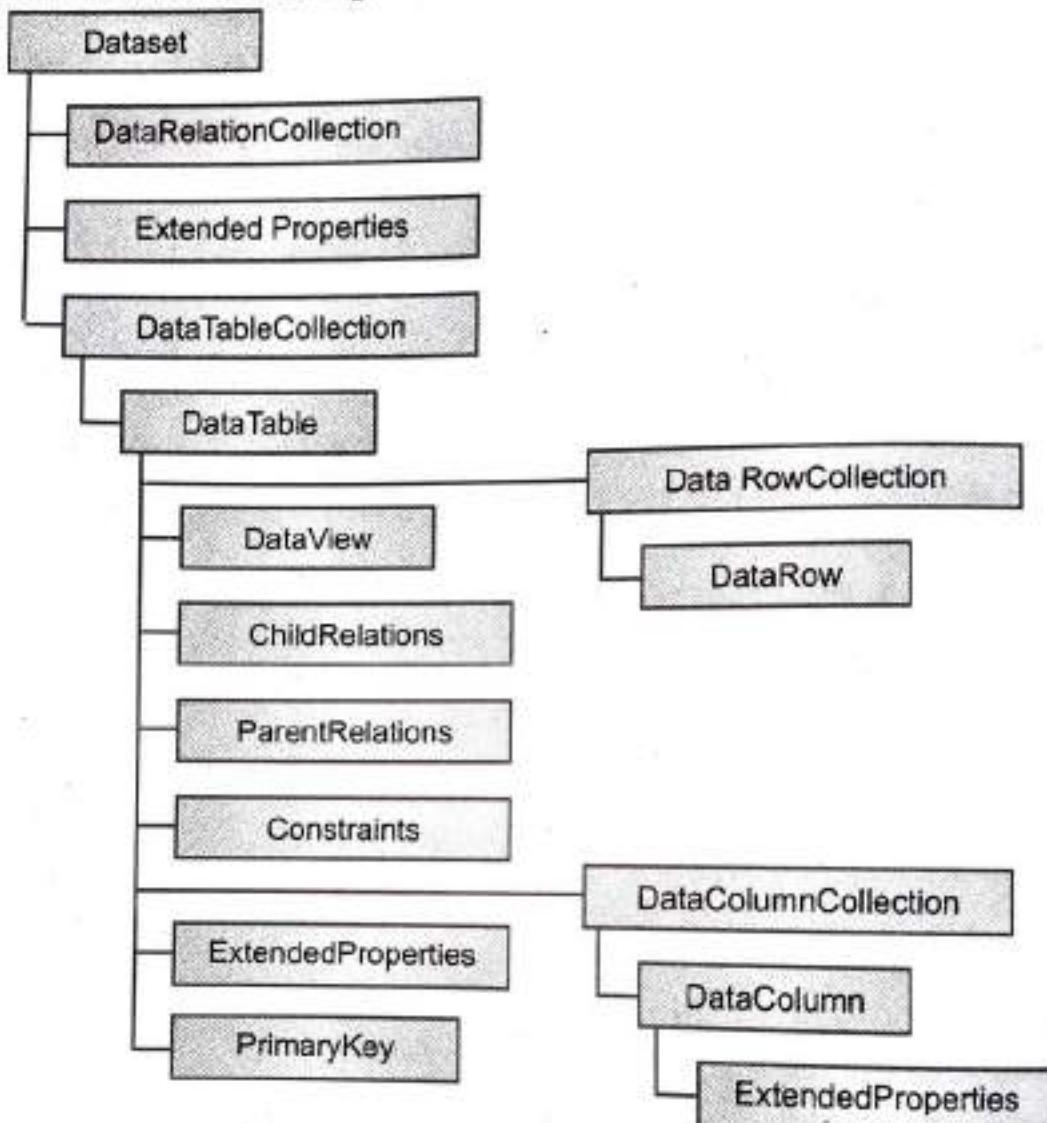


Fig. 5.1: ADO.NET architecture

Choosing a **DataReader** or a **DataSet**:

- When you decide whether your application should use a **DataReader** or a **DataSet** consider the type of functionality that your application requires. Use a **DataSet** to do the following:
 - Cache data locally in your application so that you can manipulate it. If you only need to read the results of a query, the **DataReader** is the better choice.
 - Remote data between tiers or from an XML Web service.
 - Interact with data dynamically such as binding to a Windows Forms control or combining and relating data from multiple sources.
 - Perform extensive processing on data without requiring an open connection to the data source, which frees the connection to be used by other clients.

If you do not require the functionality provided by the DataSet, you can improve the performance of your application by using the DataReader to return your data in a forward-only, read-only manner. Although the DataAdapter uses the DataReader to fill the contents of a DataSet by using the DataReader you can boost performance because you will save memory that would be consumed by the DataSet, and avoid the processing that is required to create and fill the contents of the DataSet.

XML and ADO.NET:

- ADO.NET leverages the power of XML to provide disconnected access to data. ADO.NET was designed hand-in-hand with the XML classes in the .NET Framework; both are components of a single architecture.
- ADO.NET and the XML classes in the .NET Framework converge in the DataSet object. The DataSet can be populated with data from an XML source, whether it is a file or an XML stream. The DataSet can be written as World-Wide Web Consortium (W3C) compliant XML that includes its schema as XML schema definition language (XSD) schema, regardless of the source of the data in the DataSet. Because of the native serialization format of the DataSet is XML, it is an excellent medium for moving data between tiers, making the DataSet an optimal choice for remoting data and schema context to and from an XML Web service

5.1.1 Connection Object

[W-22]

- The Connection object is the first component of ADO.NET that you should be looking at. A connection sets a link between a data source and ADO.NET. A Connection object sits between a data source and a DataAdapter (via Command). You need to define a data provider and a data source when you create a connection. With these two, you can also specify the user ID and password depending on the type of data source. Fig. 5.2 shows the relationship between a connection, a data source, and a data adapter. The Connection Object is a part of ADO.NET Data Provider and it is a unique session with the Data Source. In .Net Framework the Connection Object is Handling the part of physical communication between the application and the Data Source. Depends on the parameter specified in the Connection String, ADO.NET Connection Object connect to the specified Database and open a connection between the application and the Database. When the connection is established, SQL Commands may be executed, with the help of the Connection Object, to retrieve or manipulate data in the Database. Once the Database activity is over, Connection should be closed and releases the resources.

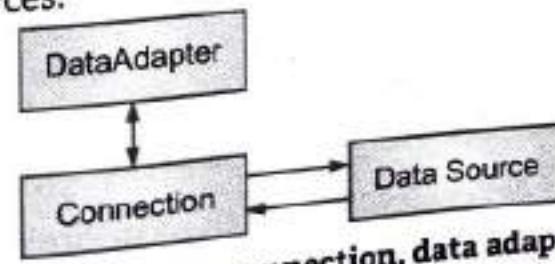


Fig. 5.2: The relationship between connection, data adapter, and a data source

- Connection can also be connected to a Command object to execute SQL queries, which can be used to retrieve, add, update, and delete data to a data source. Fig. 5.3 shows the relationship between the Command and Connection objects.
- In ADO.NET the type of the Connection is depend on what Database system you are working with. The following are the commonly using the connections in the ADO.NET

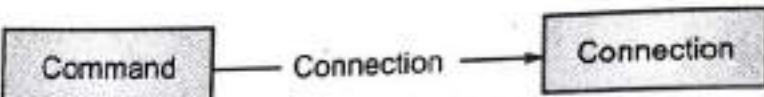


Fig. 5.3: The relationship between the command object and the connection object

- The Connection object also plays a useful role in creating a transaction. Transactions are stored in transaction objects, and transaction classes have all those nice features for dealing with transactions such as commit and rollback. Fig. 5.4 shows the relationship between the connection object and the transaction.

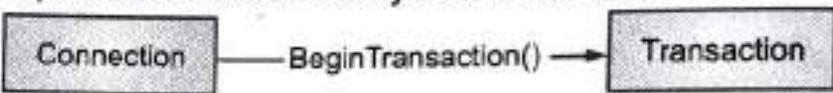


Fig. 5.4: Creating a transaction from a connection object

- Each data provider has a Connection class. Below Table 5.1 shows the name of various connection classes for data providers.

Table 5.1: Data provider connection classes

Data provider	Connection class
OleDb	OleDbConnection
Sql	SqlConnection
ODBC	OdbeConnection

- The SqlConnection Object is handling the part of physical communication between the application and the SQL Server Database. An instance of the SqlConnection class in .NET Framework is supported the Data Provider for SQL Server Database. The SqlConnection instance takes Connection String as argument and passes the value to the Constructor statement. When the connection is established, SQL Commands may be executed, with the help of the Connection Object, to retrieve or manipulate data in the database. Once the Database activities over, Connection should be closed and release the database resources.

Sql Server connection string:

- ```

connectionString = "Data Source=ServerName;
Initial Catalog=DatabaseName;User ID=UserName;Password=Password"

```
- If you have a named instance of SQL Server, you'll need to add that as well "Server=localhost\sqlexpress"
  - The Close() method in SqlConnection class is used to close the Database Connection. The Close() method rolls back any pending transactions and releases the Connection from the SQL Server Database.

[W-22]

**Example 1:**

A Sample VB.Net Program that connect SQL Server using connection string.

```

Imports System.Data.SqlClient
public Class Form1
 private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
 System.EventArgs) Handles Button1.Click
 Dim connetionString As String
 Dim cnn As SqlConnection
 connetionString = "Data Source=ServerName;
 Initial Catalog=DatabaseName;User ID=UserName;Password=Password"
 cnn = New SqlConnection(connetionString)
 Try
 cnn.Open()
 MsgBox("Connection Open ! ")
 cnn.Close()
 Catch ex As Exception
 MsgBox("Can not open connection ! ")
 End Try
 End Sub
End Class

```

**Connecting to SQL Server using windows authentication:**

"Server= localhost; Database= employeedetails;Integrated Security=SSPI;"  
 1433 is the default port for SQL Server.

**Example 2:**

- A sample VB.Net program that demonstrate how to execute sql statement and read data from SQL server.

```

imports System.Data.SqlClient
public Class Form1
 private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
 System.EventArgs) Handles Button1.Click
 Dim connetionString As String
 Dim connection As SqlConnection
 Dim command As SqlCommand
 Dim sql As String
 connetionString = "Data Source=ServerName;
 Initial Catalog=DatabaseName;User ID=UserName;Password=Password"
 sql = "Your SQL Statement Here, like Select * from product"
 connection = New SqlConnection(connetionString)
 Try
 connection.Open()
 command = New SqlCommand(sql, connection)

```

```

 Dim sqlReader As SqlDataReader = command.ExecuteReader()
 While sqlReader.Read()
 MsgBox(sqlReader.Item(0) & " - " & sqlReader.Item(1)
 & " - " & sqlReader.Item(2))
 End While
 sqlReader.Close()
 command.Dispose()
 connection.Close()
 Catch ex As Exception
 MsgBox("Can not open connection ! ")
 End Try
End Sub
End Class

```

**Example 3:**

- A sample VB.Net program that perform Data Manipulation tasks like Insert, Update, Delete etc. also perform by the ExecuteNonQuery() of SqlCommand Object.

```

imports System.Data.SqlClient
public Class Form1
 private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
 System.EventArgs) Handles Button1.Click
 Dim connetionString As String
 Dim connection As SqlConnection
 Dim command As SqlCommand
 Dim sql As String

 connetionString = "Data Source=ServerName;
 Initial Catalog=DatabaseName;User ID=UserName;Password=Password"
 sql = "Your SQL Statement Here"
 connection = New SqlConnection(connetionString)
 Try
 connection.Open()
 command = New SqlCommand(Sql, connection)
 command.ExecuteNonQuery()
 command.Dispose()
 connection.Close()
 MsgBox(" ExecuteNonQuery in SqlCommand executed !!")
 Catch ex As Exception
 MsgBox("Can not open connection ! ")
 End Try
 End Sub
End Class

```

**5.1.2 Command Object**

- The Command Object in ADO.NET executes SQL statements and Stored Procedures against the data source specified in the Connection Object. The Command Object

required an instance of a Connection Object for executing the SQL statements. That is, for retrieving data or executes an SQL statement against a Data Source, you have to create a Connection Object and open a connection to the Data Source, and assign the open connection to the connection property of the Command Object. When the Command Object return result set, a DataReader is used to retrieve the result set.

- The Command Object has a property called CommandText, which contains a String value that represents the command that will be executed in the Data Source. When the CommandType property is set to StoredProcedured, the CommandText property should be set to the name of the stored procedure.
- Some important built in methods uses in the Command Object to execute the SQL statements.

1. ExecuteNonQuery
2. ExecuteReader
3. ExecuteScalar

#### **ADO.NET ExecuteNonQuery in SqlCommand Object**

- ExecuteNonQuery() is one of the most frequently used method in SqlCommand Object and is used for executing statements that do not return result set. ExecuteNonQuery() performs Data Definition tasks as well as Data Manipulation tasks also. The Data Definition tasks like creating Stored Procedures and Views perform by ExecuteNonQuery(). Also Data Manipulation tasks like Insert, Update and Delete perform by ExecuteNonQuery().

#### **Example 4:**

- The following example shows how to use the method ExecuteNonQuery() through SqlCommand Object.

```
imports System.Data.SqlClient
public Class Form1
 private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
 System.EventArgs) Handles Button1.Click
 Dim connetionString As String
 Dim cnn As SqlConnection
 Dim cmd As SqlCommand
 Dim sql As String
 connetionString = "Data Source=ServerName;
 Initial Catalog=DatabaseName;User ID=UserName;Password=Password"
 sql = "Your SQL Statement Here"
 cnn = New SqlConnection(connetionString)
 Try
 cnn.Open()
 cmd = New SqlCommand(sql, cnn)
 cmd.ExecuteNonQuery()
```

```

 cmd.Dispose()
 cnn.Close()
 MsgBox(" ExecuteNonQuery in SqlCommand executed !!")
 Catch ex As Exception
 MsgBox("Can not open connection ! ")
 End Try
End Sub
End Class

```

### **ADO.NET ExecuteReader in SqlCommand Object**

- **ExecuteReader()** in **SqlCommand** Object send the SQL statements to Connection Object and populate a **SqlDataReader** Object based on the SQL statement. When the **ExecuteReader** method in **SqlCommand** Object execute, it instantiate a **SqlClient.SqlDataReader** Object.
- The **SqlDataReader** Object is a stream-based, forward-only, read-only retrieval of query results from the Data Source, which do not update the data. The **SqlDataReader** cannot be created directly from code, they created only by calling the **ExecuteReader** method of a **Command** Object.

#### **Example 5:**

```

imports System.Data.SqlClient
public Class Form1
 private Sub Button1_Click(ByVal sender As System.Object,
 ByVal e As System.EventArgs) Handles Button1.Click
 Dim connetionString As String
 Dim cnn As SqlConnection
 Dim cmd As SqlCommand
 Dim sql As String
 Dim reader As SqlDataReader

 connetionString = "Data Source=ServerName;
 Initial Catalog=DatabaseName;User ID=UserName;Password=Password"
 sql = "Your SQL Statement Here, like Select * from product"
 cnn = New SqlConnection(connetionString)
 Try
 cnn.Open()
 cmd = New SqlCommand(sql, cnn)
 reader = cmd.ExecuteReader()
 While reader.Read()
 MsgBox(reader.Item(0) & " - " & reader.Item(1) & " - " & reader.Item(2))
 End While
 reader.Close()
 cmd.Dispose()
 cnn.Close()
 End Sub
 End Class

```

```

 Catch ex As Exception
 MsgBox("Can not open connection ! ")
 End Try
End Sub
End Class

```

### **ADO.NET ExecuteScalar in SqlCommand Object**

- **ExecuteScalar()** in **SqlCommand** Object is used for get a single value from Database after its execution. It executes SQL statements or Stored Procedure and returned a scalar value on first column of first row in the Result Set. If the Result Set contains more than one columns or rows, it takes only the first column of first row, all other values will ignore. If the Result Set is empty it will return a Null reference.
- It is very useful to use with aggregate functions like **Count(\*)** or **Sum()** etc. When compare to **ExecuteReader()**, **ExecuteScalar()** uses fewer System resources.

#### **Example 6:**

```

imports System.Data.SqlClient
public Class Form1
 private Sub Button1_Click(ByVal sender As System.Object,
 ByVal e As System.EventArgs) Handles Button1.Click
 Dim connetionString As String
 Dim cnn As SqlConnection
 Dim cmd As SqlCommand
 Dim sql As String

 connetionString = "Data Source=ServerName;
 Initial Catalog=DatabaseName;User ID=UserName;Password=Password"
 sql = "Your SQL Statement Here like Select Count(*) from product"
 cnn = New SqlConnection(connetionString)
 Try
 cnn.Open()
 cmd = New SqlCommand(sql, cnn)
 Dim count As Int32 = Convert.ToInt32(cmd.ExecuteScalar())
 cmd.Dispose()
 cnn.Close()
 MsgBox(" No. of Rows " & count)
 Catch ex As Exception
 MsgBox("Can not open connection ! ")
 End Try
 End Sub
End Class

```

### **ADO.NET ExecuteNonQuery in SqlCommand Object**

- **ExecuteNonQuery()** is one of the most frequently used method in **SqlCommand** Object and is used for executing statements that do not return result set.

ExecuteNonQuery() performs Data Definition tasks as well as Data Manipulation tasks also. The Data Definition tasks like creating Stored Procedures and Views perform by ExecuteNonQuery(). Also Data Manipulation tasks like Insert, Update and Delete perform by ExecuteNonQuery().

- The following example shows how to use the method ExecuteNonQuery() through SqlCommand Object.

#### **Example 7:**

```

imports System.Data.SqlClient
public Class Form1
 private Sub Button1_Click(ByVal sender As System.Object,
 ByVal e As System.EventArgs) Handles Button1.Click
 Dim connetionString As String
 Dim cnn As SqlConnection
 Dim cmd As SqlCommand
 Dim sql As String

 connetionString = "Data Source=ServerName;
 Initial Catalog=DatabaseName;User ID=UserName;Password=Password"
 sql = "Your SQL Statement Here"

 cnn = New SqlConnection(connetionString)
 Try
 cnn.Open()
 cmd = New SqlCommand(Sql, cnn)
 cmd.ExecuteNonQuery()
 cmd.Dispose()
 cnn.Close()
 MsgBox("ExecuteNonQuery in SqlCommand executed !!")
 Catch ex As Exception
 MsgBox("Can not open connection ! ")
 End Try
 End Sub
End Class

```

### **5.1.3 ADO.NET DataSet**

[S-23, 22]

- It is a collection of data tables that contain the data. It is used to fetch data without interacting with a Data Source that's why, it also known as **disconnected** data access method. It is an in-memory data store that can hold more than one table at the same time. We can use DataRelation object to relate these tables. The DataSet can also be used to read and write data as XML document.
- ADO.NET provides a DataSet class that can be used to create DataSet object. It contains constructors and methods to perform data related operations.

**Example 8:**

Here, in this example, we are implementing **DataSet** and displaying data into a **GridView**. Create a web form and drag a **GridView** from the **toolbox** to the form. We can find it inside the **data** category.



```
// DataSetDemo.aspx
<%@ Page Language="C#" AutoEventWireup="true"
 CodeBehind="DataSetDemo.aspx.cs"
 Inherits="DataSetExampleDataSetDemo" %>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
 <title></title>
</head>
<body>
 <form id="form1" runat="server">
 <div>
 </div>
 <asp:GridView ID="GridView1" runat="server"
 CellPadding="4" ForeColor="#333333" GridLines="None">
 <AlternatingRowStyle BackColor="White" />
 <EditRowStyle BackColor="#2461BF" />
 <FooterStyle BackColor="#507CD1" Font
 Bold="True" ForeColor="White" />
 <HeaderStyle BackColor="#D9EAF7" Font
 Bold="True" ForeColor="Black" />
 <RowStyle BackColor="#F7F7F7" />
 </asp:GridView>
 </form>
</body>
</html>
```

```
<HeaderStyle BackColor="#507CD1" Font
 Bold="True" ForeColor="White" />
<PagerStyle BackColor="#2461BF" ForeColor="White"
 HorizontalAlign="Center" />
<RowStyle BackColor="#EFF3FB" />
<SelectedRowStyle BackColor="#D1DDF1" Font
 Bold="True" ForeColor="#333333" />
<SortedAscendingCellStyle BackColor="#F5F7FB" />
<SortedAscendingHeaderStyle BackColor="#6D95E1" />
<SortedDescendingCellStyle BackColor="#E9EBEF" />
<SortedDescendingHeaderStyle BackColor="#4870BE" />
</asp:GridView>
</form>
</body>
</html>
```

**CodeBehind**

```
// DataSetDemo.aspx.cs
using System;
using System.Data.SqlClient;
using System.Data;
namespace DataSetExample
{
 public partial class DataSetDemo : System.Web.UI.Page
 {
 protected void Page_Load(object sender, EventArgs e)
 {
 using (SqlConnection con = new SqlConnection("data source=.; database=student; integrated security=SSPI"))
 {
 SqlDataAdapter sde = new SqlDataAdapter
 ("Select * from student", con);
 DataSet ds = new DataSet();
 sde.Fill(ds);
 GridView1.DataSource = ds;
 GridView1.DataBind();
 }
 }
 }
}
```

## 5.4 DataTables

A **DataSet** is made up of a collection of tables, relationships, and constraints. In ADO.NET, **DataTable** objects are used to represent the tables in a **DataSet**. A **DataTable** represents one table of in-memory relational data; the data is local to the .NET-based application in which it resides, but can be populated from a data source such as Microsoft SQL Server using a **DataAdapter**.

The **DataTable** class is a member of the **System.Data** namespace within the .NET Framework class library. You can create and use a **DataTable** independently or as a member of a **DataSet**, and **DataTable** objects can also be used in conjunction with other .NET Framework objects, including the **DataView**. You access the collection of tables in a **DataSet** through the **Tables** property of the **DataSet** object.

The schema or structure of a table is represented by columns and constraints. You define the schema of a **DataTable** using  **DataColumn** objects as well as **ForeignKeyConstraint** and **UniqueConstraint** objects. The columns in a table can map to columns in a data source, contain calculated values from expressions, automatically increment their values, or contain primary key values.

In addition to a schema, a **DataTable** must also have rows to contain and order data. The **DataRow** class represents the actual data contained in a table. You use the **DataRow** and its properties and methods to retrieve, evaluate, and manipulate the data in a table. As you access and change the data within a row, the **DataRow** object maintains both its current and original state.

You can create parent-child relationships between tables using one or more related columns in the tables. You create a relationship between **DataTable** objects using a **DataRelation**. **DataRelation** objects can then be used to return the related child or parent rows of a particular row.

The **DataTable** class in C# ADO.NET is a database table representation and provides a collection of columns and rows to store data in a grid form. The code sample in this article explains how to create a **DataTable** at run-time in C#. You will also learn how to create a **DataTable** columns and rows, add data to a **DataTable** and bind a **DataTable** to a **DataGridView** control using data binding.

Table 5.2 describes some of the common **DataTable** properties.

**Table 5.2: The **DataTable** class properties**

Property	Description
Columns	Represents all table columns
Constraints	Represents all table constraints
DataSet	Returns the dataset for the table
DefaultView	Customized view of the data table

*contd....*

ChildRelation	Return child relations for the data table
ParentRelation	Returns parent relations for the data table
PrimaryKey	Represents an array of columns that function as primary key for the table
Rows	All rows of the data table
TableName	Name of the table

- Table 5.3 describes some of the common DataTable methods.

**Table 5.3: The DataTable Class Methods**

Method	Description
AcceptChanges	Commits all the changes made since last AcceptChanges was called
Clear	Deletes all data table data
Clone	Creates a clone of a DataTable including its schema
Copy	Copies a data table including its schema
NewRow	Creates a new row, which is later added by calling the Rows.Add method
RejectChanges	Reject all changes made after last AcceptChanges was called
Reset	Resets a data table's original state
Select	Gets an array of rows based on the criteria

#### How to create a DataTable in C#?

- In order to create a DataTable in C#, first, we need to create an instance of the **DataTable** class, and then we need to add **DataColumn** objects that define the type of data to be held and insert **DataRow** objects that contain the data. Let us discuss this step by step.

#### Step 1: Creating DataTable instance

Please have a look at the following code. Here, we are using the constructor which takes the table name as a parameter

```
DataTable dataTable = new DataTable("Student");
```

The above code will create an empty data table for which the TableName property is set to Student. Later, you can use this property to access this data table from a DataTableCollection. Once the DataTable is created, the next important step is to add the data columns and define the schema for the columns.

#### Step 2: Adding DataColumn and Defining Schema

A **DataTable** is actually a collection of **DataColumn** objects which is referenced by the Columns property of the data table. A **DataTable** object is useless until it has a schema. You can create the schema by adding **DataColumn** objects and setting the

constraints of columns. As we already know from SQL's point of view, Constraints are basically used to maintain data integrity. Let us see how to Create DataColumn and set the schema. The following code shows creating a data column using all the available properties.

```
//Add the DataColumn using all properties
DataColumn Id = new DataColumn("ID");
Id.DataType = typeof(int);
Id.Unique = true;
Id.AllowDBNull = false;
Id.Caption = "Student ID";
dataTable.Columns.Add(Id);
```

The following code shows adding Data Column using few properties.

```
DataColumn Name = new DataColumn("Name");
Name.MaxLength = 50;
Name.AllowDBNull = false;
dataTable.Columns.Add(Name);
```

The following code shows creating a Data Column with the default properties

```
DataColumn Email = new DataColumn("Email");
dataTable.Columns.Add(Email);
```

### **Creating Primary Key Column in Datatable:**

- Like SQL, the primary key of a DataTable object also consists of a column or columns that make up a unique identity for each data row. The following code shows how to set the PrimaryKey property on the Id column of the Student DataTable object.

```
//Setting the Primary Key
dataTable.PrimaryKey = new DataColumn [] { Id };
```

### **Creating DataRow Objects in C#:**

- Once you created the DataColumns for the DataTable object, then you can populate the DataTable object by adding DataRow objects. You need to use the DataRow object and its properties and methods to retrieve, insert, update and delete the values in the DataTable. The DataRowCollection represents the actual DataRow objects in the DataTable and it has an Add method that accepts a DataRow object. The Add method is also overloaded to accept an array of objects instead of a DataRow object. The following code shows how to create and add data into the Student DataTable object.

```
//Add New DataRow by creating the DataRow object
DataRow row1 = dataTable.NewRow();
row1["Id"] = 101;
row1["Name"] = "John";
row1["Email"] = "john.america@gmail.com";
dataTable.Rows.Add(row1);
```

- You can also add a new DataRow by simply adding the values as shown in the below code.

```
//Adding new DataRow by simply adding the values
dataTable.Rows.Add(102, "jessica", "jessica@gmail.com");
```

#### Iterating the DataTable in C#:

- You can use a for each loop to loop through the rows and columns of a data table. The following code shows how to enumerate through the rows and columns of a data table.

```
for each (DataRow row in dataTable.Rows)
{
 Console.WriteLine(row["Id"] + ", " + row["Name"] + ", " + row["Email"]);
}
```

- The complete code is given below:

```
using System;
using System.Data;
using System.Data.SqlClient;
namespace AdoNetConsoleApplication
{
 class Program
 {
 static void Main(string[] args)
 {
 try
 {
 //Creating data table instance
 DataTable dataTable = new DataTable("Student");
 //Add the DataColumn using all properties
 DataColumn Id = new DataColumn("ID");
 Id.DataType = typeof(int);
 Id.Unique = true;
 Id.AllowDBNull = false;
 Id.Caption = "Student ID";
 dataTable.Columns.Add(Id);

 //Add the DataColumn few properties
 DataColumn Name = new DataColumn("Name");
 Name.MaxLength = 50;
 Name.AllowDBNull = false;
 dataTable.Columns.Add(Name);
 }
 }
 }
}
```

```
//Add the DataColumn using defaults
DataColumn Email = new DataColumn("Email");
dataTable.Columns.Add(Email);

//Setting the Primary Key
dataTable.PrimaryKey = new DataColumn [] { Id };

//Add New DataRow by creating the DataRow object
DataRow row1 = dataTable.NewRow();
row1["Id"] = 101;
row1["Name"] = "John";
row1["Email"] = "john.america@gmail.com";
dataTable.Rows.Add(row1);

//Adding new DataRow by simply adding the values
dataTable.Rows.Add(102, "jessica", "jessica@gmail.com");

foreach (DataRow row in dataTable.Rows)
{
 Console.WriteLine(row["Id"] + ", " + row["Name"]
 + ", " + row["Email"]);
}

}
catch (Exception e)
{
 Console.WriteLine ("OOPS, something went wrong.\n" + e);
}
Console.ReadKey ();
}
}
}
```

[S-22, W-22]

## 5.1.5 DataReader in ADO.NET

There are two common objects in ADO.NET to read data, DataSet and DataReader. C# DataSet and C# DataReader classes represent these objects. In this article, we'll learn about ADO.NET DataReader and how to use a DataReader in C#.

A data reader provides an easy way for the programmer to read data from a database as if it were coming from a stream. The DataReader is the solution for forward streaming data through ADO.NET. The data reader is also called a firehose cursor or forward read-only cursor because it moves forward through the data. The data reader not only allows you to move forward through each record of database, but it also enables you to parse the data from each column. The DataReader class represents a data reader in ADO.NET.

- Similar to other ADO.NET objects, each data provider has a data reader class for example; `OleDbDataReader` is the data reader class for OleDb data providers. Similarly, `SqlDataReader` and `ODBC DataReader` are data reader classes for SQL and ODBC data providers, respectively.
- The `IDataReader` interface defines the functionality of a data reader and works as the base class for all data provider-specific data reader classes such as `OleDbDataReader`, `SqlDataReader`, and `OdbcDataReader`.

### **Initializing DataReader**

- You call the `ExecuteReader` method of the `Command` object, which returns an instance of the `DataReader`.

```
SqlCommand cmd = new SqlCommand(SQL, conn);
// Call ExecuteReader to return a DataReader
SqlDataReader reader = cmd.ExecuteReader();
```

- Once you're done with the data reader, call the `Close` method to close a data reader:

```
reader.Close();
```

### **Reading with the DataReader**

- Once the `SqlDataReader` is initialize, you can utilize its various methods to read your data records. Foremost, you can use the `Read` method, which, when called repeatedly, continues to read each row of data into the `DataReader` object. The `DataReader` also provides a simple indexer that enables you to pull each column of data from the row.
- This class is used to read data from SQL Server database. It reads data in forward-only stream of rows from a SQL Server database. It is sealed class so that cannot be inherited. It inherits `DbDataReader` class and implements `Disposable` interface

### **Example 9:**

In the following program, we are using `SqlDataReader` to get data from the SQL Server. A C# code is given below.

```
// Program.cs
using System;
using System.Data.SqlClient;
namespace AdoNetConsoleApplication
{
 class Program
 {
 static void Main(string[] args)
 {
 new Program().GetData();
 }
 public void GetData()
 {
 SqlConnection con = null;
```

```
try
{
 // Creating Connection
 con = new SqlConnection("data source=.; database=student;
 integrated security=SSPI");
 // writing sql query
 SqlCommand cm = new SqlCommand("select * from student", con);
 // Opening Connection
 con.Open();
 // Executing the SQL query
 SqlDataReader sdr = cm.ExecuteReader();
 while (sdr.Read())
 {
 Console.WriteLine(sdr["name"] + " " + sdr["email"]);
 }
}
catch (Exception e)
{
 Console.WriteLine("OOPS, something went wrong." + e);
}
// Closing the connection
finally
{
 con.Close();
}
```

#### Output:

Execute this program by combination of **Ctrl+F5** and it will produce the following output.



#### 5.1.6 DataAdapter Object

**DataAdapter** is a part of the ADO.NET Data Provider. DataAdapter provides the communication between the **Dataset** and the Datasource. We can use the DataAdapter in combination with the **DataSet** Object. DataAdapter provides this

combination by mapping Fill method, which changes the data in the DataSet to match the data in the data source, and Update, which changes the data in the data source to match the data in the DataSet. That is, these two objects combine to enable both data access and data manipulation capabilities.

- The **DataAdapter** can perform Select, Insert, Update and Delete SQL operations in the Data Source. The Insert, Update and Delete SQL operations, we are using the continuation of the Select command perform by the DataAdapter.
- The **SelectCommand** property of the DataAdapter is a Command Object that retrieves data from the data source. The **InsertCommand**, **UpdateCommand**, and **DeleteCommand** properties of the DataAdapter are Command objects that manage updates to the data in the data source according to modifications made to the data in the DataSet.
- A DataAdapter plays a vital role in ADO.NET architecture. It sits between a data source and a dataset and passes data from the data source to the dataset, and vice versa, with or without using commands. Now is the time you'll be using disconnected classes such as DataSet, DataTable, DataView, and DataViewManager to write Windows Forms and Web Forms-based interactive database GUI applications.
- By reading this article, you will learn key points in ADO.NET, as given below.
  - What is Data Adapter?
  - Primary Data Adapters for the databases.
  - Data Adapter properties.
  - Methods used by Data Adapter.
  - How to create Data Adapter with an example?

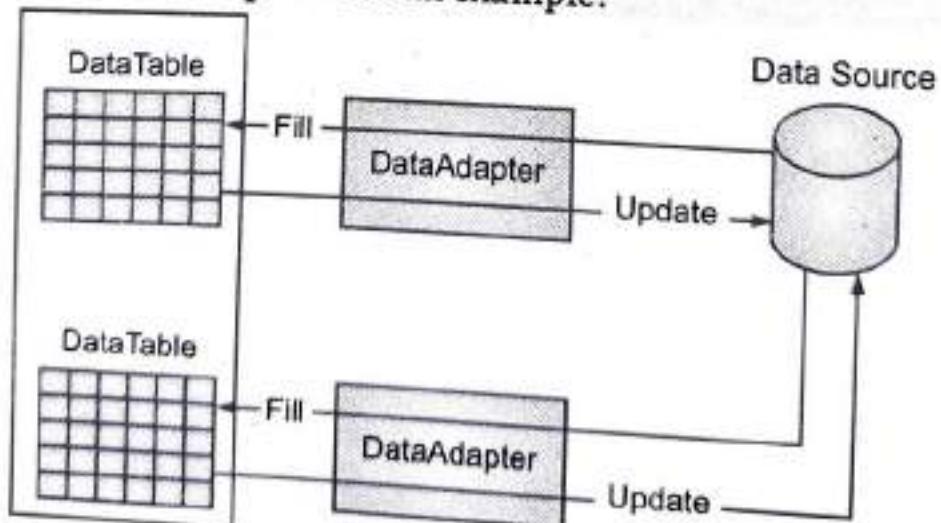


Fig. 5.5

#### Data Adapter introduction:

- Actually, we use Data Adapter object to establish the connection to the data source and manage the movement of date to and from the database.

## **What is Data Adapter?**

- A data adapter object serves as a bridge between a data set object and Data Source such as a database to retrieve and save the data.
- Data adapter contains a set of database commands and a database connection, which we use to fill a dataset object and update the Data Source.

## **Primary data adapters for databases**

- .NET makes two primary data adapters available for use with the databases. Other data adapters can also be integrated with Visual Studio .NET.
- Primary Data Adapters are mentioned below.
- **OleDbDataAdapter**, which is suitable for use with certain OLE DB providers.
- **SQLDataAdapters**, which is specific to a Microsoft SQL server. This is faster than the **OleDbDataAdapter** because it works directly with SQL servers and does not go through an OLE Db Layer.

## **Data adapter properties**

- We use data adapter objects to act on records from a Data Source. We can also specify, which action we want to perform by using one of following four data adapter properties, which executes a SQL statement.
- The properties are given below:
  - Select command retrieves rows from Data Source.
  - Insert command writes inserted rows from data set into Data Source.
  - Update command writes modified rows from data set into Data Source.
  - Delete command deletes rows from Data Source.

## **Methods used by a data adapter**

- Actually, we use data adapters to fill or to make changes in a data set table to a data store. These methods comprises of following.
  - **FillUse:** This method of a SQL data adapter to add or refresh row from a Data Source and place them in a Data Set table. The fill method uses Select statement, which is specified in the Select command property
  - **UpdateUse:** This method of data adapter objects to transmit the changes to a dataset table to the corresponding Data Source. This method calls the corresponding insert, delete or update command for each specified row in a data table in a data set.
  - **CloseUse:** This method for the connection to a database.
- Creating Data Adapter with Example the examples given below use a **SQLDataAdapter** object to define a query in the database.

### **Example 10:**

```
using System;
using System.Data;
using System.Data.SqlClient;
```

```

namespace DataAdapter
{
 class Program
 {
 static void Main(string[] args)
 {
 string connectionString = @ "data source=localhost;initial
catalog=northwind;integrated security = SSPI";
 string commandString = @ "SELECT * from customers";
 SqlDataAdapter da = new SqlDataAdapter
 (commandString, connectionString);

 DataSet ds = new DataSet();
 DataAdapter.Fill(ds);
 DataTable dt = ds.Tables[0];
 int rows = dt.Rows.Count;
 }
 }
}

```

- Hence, this is the way of how can we create Data Adapter object, using ADO.NET.
- SqlDataAdapter Class is a part of the C# ADO.NET Data Provider and it resides in the System.Data.SqlClient namespace. SqlDataAdapter provides the communication between the Dataset and the SQL database. We can use SqlDataAdapter Object in combination with Dataset Object. DataAdapter provides this combination by mapping Fill method, which changes the data in the DataSet to match the data in the data source, and Update, which changes the data in the data source to match the data in the DataSet.

```

SqlDataAdapter adapter = new SqlDataAdapter();
adapter.Fill(ds);

```

- The SqlDataAdapter Object and DataSet objects are combine to perform both data access and data manipulation operations in the SQL Server Database. When the user perform the SQL operations like Select, Insert etc. in the data containing in the Dataset Object, it won't directly affect the Database, until the user invoke the Update method in the SqlDataAdapter.

```

using System;
using System.Windows.Forms;
using System.Data;
using System.Data.SqlClient;
namespace WindowsApplication1
{
 public partial class Form1 : Form
 {

```

```
public Form1()
{
 InitializeComponent();
}
private void button1_Click(object sender, EventArgs e)
{
 string connetionString = null;
 SqlConnection sqlCnn ;
 SqlCommand sqlCmd ;
 SqlDataAdapter adapter = new SqlDataAdapter();
 DataSet ds = new DataSet();
 int i = 0;
 string sql = null;
 connetionString = "Data Source=ServerName;Initial
Catalog=DatabaseName;User ID=UserName;Password=Password";
 sql = "Select * from product";
 sqlCnn = new SqlConnection(connetionString);
 try
 {
 sqlCnn.Open();
 sqlCmd = new SqlCommand(sql, sqlCnn);
 adapter.SelectCommand = sqlCmd;
 adapter.Fill(ds);
 for (i = 0; i <= ds.Tables[0].Rows.Count - 1; i++)
 {
 MessageBox.Show(ds.Tables[0].Rows[i].ItemArray[0]
 + " -- " + ds.Tables[0].Rows[i].ItemArray[1]);
 }
 adapter.Dispose();
 sqlCmd.Dispose();
 sqlCnn.Close();
 }
 catch (Exception ex)
 {
 MessageBox.Show("Can not open connection ! ");
 }
}
```

- The DataAdapter constructor has many overloaded forms. You can create a constructor with no arguments, pass a Command object, pass a Command object with Connection object as arguments, or any combination of these. You can also

specify a SQL statement as a string for querying a particular table or more than one table. You can also specify the connection string or a Connection object to connect to the database.

- The codes given creates a connection, builds a SQL statement using the SELECT query, and passes the SQL string and the connection objects as SqlDataAdapter constructor arguments

```
string ConnectionString = "Integrated Security = SSPI;" + "Initial catalog = Northwind;" + "Data Source = localhost;";
string SQL = "SELECT CustomerID, CompanyName FROM Customers";
SqlConenction conn = new SqlConnection(ConnectionString);
conn.Open();
SqlDataAdapter adapter = new SqlDataAdapter(SQL, conn);
```

- The DataAdapter works as a bridge between a DataSet and a data source to retrieve data. DataAdapter is a class that represents a set of SQL commands and a database connection. It can be used to fill the DataSet and update the data source.

### Example 11:

```
// DataSetDemo.aspx
<%@ Page Language="C#" AutoEventWireup="true"
 CodeBehind="DataSetDemo.aspx.cs"
Inherits="DataSetExample.DataSetDemo" %>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
 <title></title>
</head>
<body>
 <form id="form1" runat="server">
 <div>

 </div>
 <asp:GridView ID="GridView1" runat="server"
 CellPadding="3" BackColor="#DEBA84"
BorderColor="#DEBA84" BorderStyle="None"
 BorderWidth="1px" CellSpacing="2">
 <FooterStyle BackColor="#F7DFB5" ForeColor="#8C4510" />
 <HeaderStyle BackColor="#A55129"
 Font-Bold="True" ForeColor="White" />
 <PagerStyle BackColor="#8C4510" HorizontalAlign="Center" />
 <RowStyle BackColor="#FFF7E7" ForeColor="#8C4510" />
 <SelectedRowStyle BackColor="#738A9C"
 Font-Bold="True" ForeColor="White" />
```

```

<SortedAscendingCellStyle BackColor="#FFF1D4" />
<SortedAscendingHeaderStyle BackColor="#B95C30" />
<SortedDescendingCellStyle BackColor="#F1E5CE" />
<SortedDescendingHeaderStyle BackColor="#93451F" />
</asp:GridView>
</form>
</body>
</html>

```

**CodeBehind**

```

using System;
using System.Data.SqlClient;
using System.Data;
namespace DataSetExample
{
 public partial class DataSetDemo : System.Web.UI.Page
 {
 protected void Page_Load(object sender, EventArgs e)
 {
 using (SqlConnection con = new SqlConnection("data source=.;
database=student; integrated security=SSPI"))
 {
 SqlDataAdapter sde = new SqlDataAdapter
 ("Select * from student", con);
 DataSet ds = new DataSet();
 sde.Fill(ds);
 GridView1.DataSource = ds;
 GridView1.DataBind();
 }
 }
 }
}

```

**5.2 DATAGRIDVIEW DATA BINDING( Insert, Update, Delete)**

- The DataGridView control supports the standard Windows Forms data binding model, so it can bind to a variety of data sources. Usually, you bind to a BindingSource that manages the interaction with the data source. The BindingSource can be any Windows Forms data source, which gives you great flexibility when choosing or modifying your data's location. For more information about data sources the DataGridView control supports, see the DataGridView control overview.
- Visual Studio has extensive support for data binding to the DataGridView control.

- To connect a DataGridView control to data:
  1. Implement a method to handle the details of retrieving the data. The following code example implements a `GetData` method that initializes a `SqlDataAdapter`, and uses it to populate a `DataTable`. It then binds the `DataTable` to the `BindingSource`.
  2. In the form's Load event handler, bind the `DataGridView` control to the `BindingSource`, and call the `GetData` method to retrieve the data.
- **Example:** This complete code example retrieves data from a database to populate a `DataGridView` control in a Windows form. The form also has buttons to reload data and submit changes to the database.
- This example requires:
  - Access to a Northwind SQL Server sample database. For more information about installing the Northwind sample database.
  - References to the `System`, `System.Windows.Forms`, `System.Data`, and `System.Xml` assemblies.
- To build and run this example, paste the code into the `Form1` code file in a new Windows Forms project. For information about building from the C# or Visual Basic command line.
- Populate the `ConnectionString` variable in the example with the values for your Northwind SQL Server sample database connection. Windows Authentication, also called integrated security, is a more secure way to connect to the database than storing a password in the connection string. For more information about connection security.

**Example 12:**

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Globalization;
using System.Windows.Forms;
namespace WindowsFormsApp
{
 public partial class Form1 : Form
 {
 public Form1()
 {
 InitializeComponent();
 }
 }
}
```

```
public class Form1 : Form
{
 private DataGridView dataGridView1 = new DataGridView();
 private BindingSource bindingSource1 = new BindingSource();
 private SqlDataAdapter dataAdapter = new SqlDataAdapter();
 private Button reloadButton = new Button();
 private Button submitButton = new Button();

 [STAThread()]
 public static void Main()
 {
 Application.Run(new Form1());
 }

 // Initialize the form.
 public Form1()
 {
 dataGridView1.Dock = DockStyle.Fill;
 reloadButton.Text = "Reload";
 submitButton.Text = "Submit";
 reloadButton.Click += new EventHandler(ReloadButton_Click);
 submitButton.Click += new EventHandler(SubmitButton_Click);
 FlowLayoutPanel panel = new FlowLayoutPanel
 {
 Dock = DockStyle.Top,
 AutoSize = true
 };
 panel.Controls.AddRange(new Control[]
 {
 reloadButton, submitButton
 });

 Controls.AddRange(new Control[] { dataGridView1, panel });
 Load += new EventHandler(Form1_Load);
 Text = "DataGridView data binding and updating demo";
 }

 private void GetData(string selectCommand)
 {
 try
 {
 // Specify a connection string.
 // Replace <SQL Server> with the SQL Server for your Northwind
 // sample database.
 // Replace "Integrated Security=True" with user login
 // information if necessary.
 String connectionString =
 "Data Source=<SQL Server>;Initial Catalog=Northwind;" +
 "Integrated Security=True";
 }
 }
}
```

```

 // Create a new data adapter based on the specified query.
 dataAdapter = new SqlDataAdapter(selectCommand,
connectionString);

 // Create a command builder to generate SQL update, insert, and
 // delete commands based on selectCommand.
 SqlCommandBuilder commandBuilder =
 new SqlCommandBuilder(dataAdapter);

 // Populate a new data table and bind it to the BindingSource.
 DataTable table = new DataTable
 {
 Locale = CultureInfo.InvariantCulture
 };
 dataAdapter.Fill(table);
 bindingSource1.DataSource = table;

 // Resize the DataGridView columns to fit the newly loaded content.
 dataGridView1.AutoResizeColumns(
 DataGridViewAutoSizeColumnsMode.AllCellsExceptHeader);
}

catch (SqlException)
{
 MessageBox.Show("To run this example, replace the value of the "
 + "ConnectionString variable with a connection string that is "
 + "valid for your system.");
}

private void Form1_Load(object sender, EventArgs e)
{
 // Bind the DataGridView to the BindingSource
 // and load the data from the database.
 dataGridView1.DataSource = bindingSource1;
 GetData("select * from Customers");
}

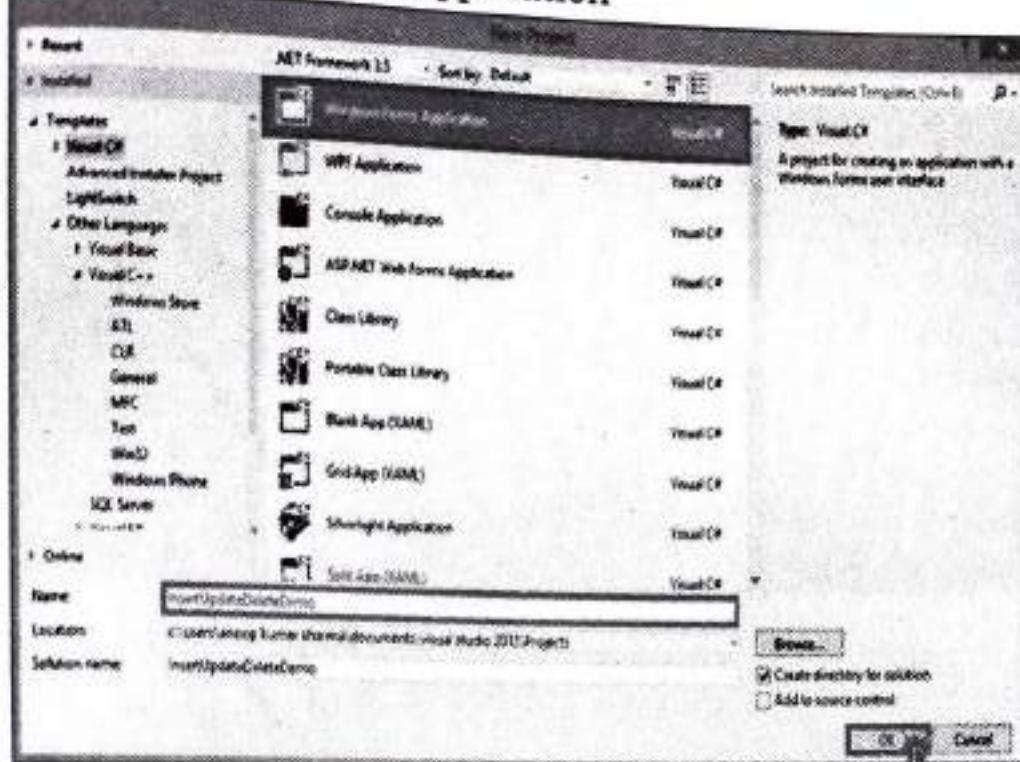
private void ReloadButton_Click(object sender, EventArgs e)
{
 // Reload the data from the database.
 GetData(dataAdapter.SelectCommand.CommandText);
}

private void SubmitButton_Click(object sender, EventArgs e)
{
 // Update the database with changes.
 dataAdapter.Update((DataTable)bindingSource1.DataSource);
}
}

```

- This article shows how to insert, update, and delete records in a DataGridView in a C# Windows Forms application. In a previous article, we saw how to use a Chart Control in Windows Forms Application.

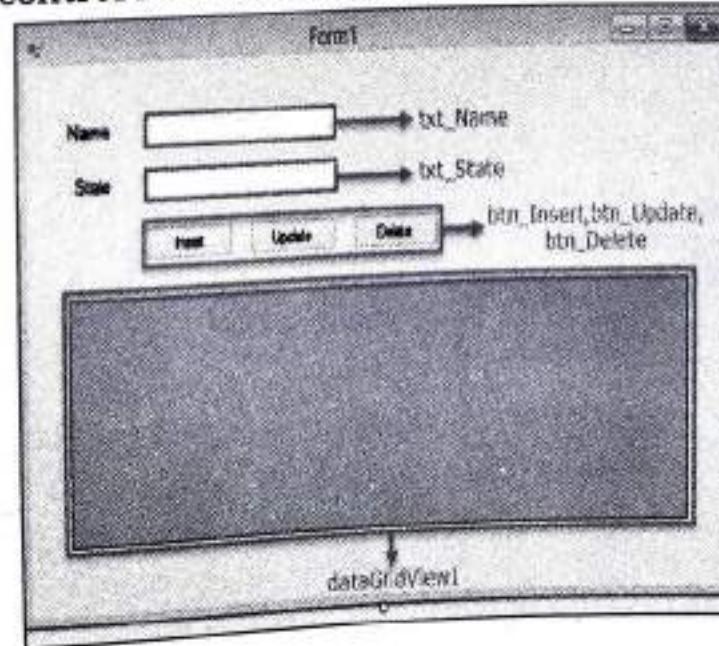
# 1. Create a new Windows Forms application



2. Create a database (named Sample). Add a table `tbl_Record`. The following is the table schema for creating `tbl_Record`.

Column Name	Data Type	Allow Nulls
ID	int	<input type="checkbox"/>
Name	varchar(50)	<input checked="" type="checkbox"/>
State	varchar(50)	<input checked="" type="checkbox"/>

3. Create a form (named frmMain) and drop a Label, TextBox, Button, and DataGridView control from the ToolBox.



- Now, go to frmMain.cs code and add the System.Data and System.Data.SqlClient namespaces.

```
using System;
using System.Data;
using System.Windows.Forms;
using System.Data.SqlClient;
namespace InsertUpdateDeleteDemo
{
 public partial class frmMain : Form
 {
 SqlConnection con= new SqlConnection("Data Source=.;Initial Catalog=Sample;Integrated Security=true;");
 SqlCommand cmd;
 SqlDataAdapter adapt;
 //ID variable used in Updating and Deleting Record
 int ID = 0;
 public frmMain()
 {
 InitializeComponent();
 DisplayData();
 }
 //Insert Data
 private void btn_Insert_Click(object sender, EventArgs e)
 {
 if (txt_Name.Text != "" && txt_State.Text != "")
 {
 cmd = new SqlCommand("insert into tbl_Record(Name,State)
values(@name,@state)", con);
 con.Open();
 cmd.Parameters.AddWithValue("@name", txt_Name.Text);
 cmd.Parameters.AddWithValue("@state", txt_State.Text);
 cmd.ExecuteNonQuery();
 con.Close();
 MessageBox.Show("Record Inserted Successfully");
 DisplayData();
 ClearData();
 }
 else
 {
 MessageBox.Show("Please Provide Details!");
 }
 }
 }
}
```

```
//Display Data in DataGridView
private void DisplayData()
{
 con.Open();
 DataTable dt=new DataTable();
 adapt=new SqlDataAdapter("select * from tbl_Record",con);
 adapt.Fill(dt);
 dataGridView1.DataSource = dt;
 con.Close();
}
//Clear Data
private void ClearData()
{
 txt_Name.Text = "";
 txt_State.Text = "";
 ID = 0;
}
//dataGridView1 RowHeaderMouseClick Event
private void dataGridView1_RowHeaderMouseClick
 (object sender, DataGridViewCellMouseEventArgs e)
{
 ID = Convert.ToInt32(dataGridView1.Rows[e.RowIndex].Cells[0].
 Value.ToString());
 txt_Name.Text = dataGridView1.Rows[e.RowIndex].Cells[1].
 Value.ToString();
 txt_State.Text = dataGridView1.Rows[e.RowIndex].Cells[2].
 Value.ToString();
}
//Update Record
private void btn_Update_Click(object sender, EventArgs e)
{
 if (txt_Name.Text != "" && txt_State.Text != "")
 {
 cmd = new SqlCommand("update tbl_Record set Name=@name,
 State=@state where ID=@id", con);

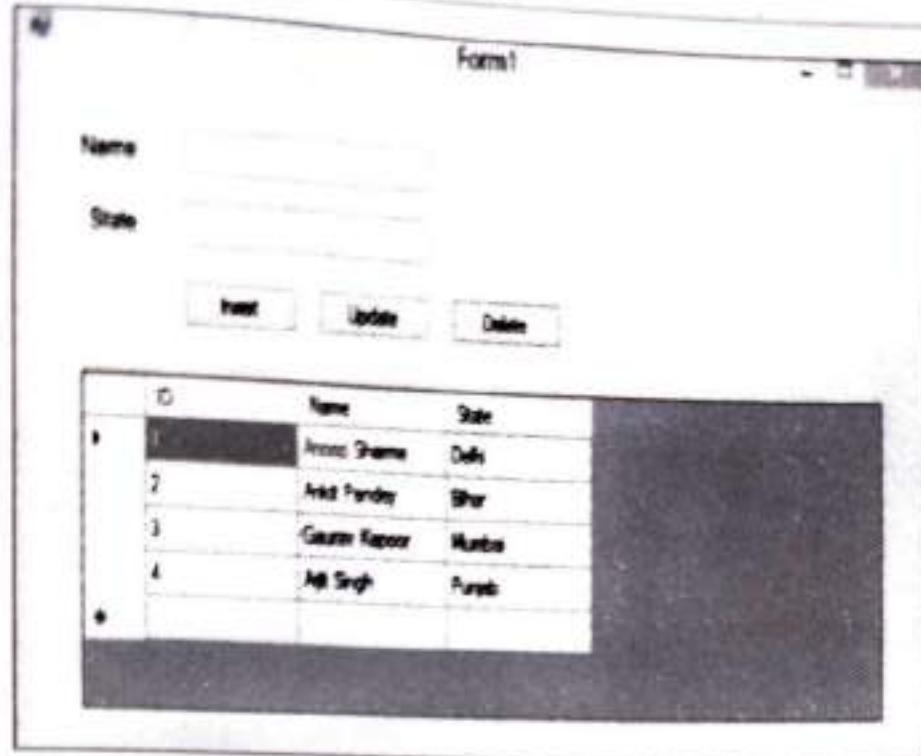
 con.Open();
 cmd.Parameters.AddWithValue("@id", ID);
 cmd.Parameters.AddWithValue("@name", txt_Name.Text);
 cmd.Parameters.AddWithValue("@state", txt_State.Text);
 }
}
```

```

 cmd.ExecuteNonQuery();
 MessageBox.Show("Record Updated Successfully");
 con.Close();
 DisplayData();
 ClearData();
 }
 else
 {
 MessageBox.Show("Please Select Record to Update");
 }
}
//Delete Record
private void btn_Delete_Click(object sender, EventArgs e)
{
 if(ID!=0)
 {
 cmd = new SqlCommand("delete tbl_Record where ID=@id",con);
 con.Open();
 cmd.Parameters.AddWithValue("@id",ID);
 cmd.ExecuteNonQuery();
 con.Close();
 MessageBox.Show("Record Deleted Successfully!");
 DisplayData();
 ClearData();
 }
 else
 {
 MessageBox.Show("Please Select Record to Delete");
 }
}
}

```

- In the preceding code, I created a `dataGridView1_RowHeaderMouseClick` Event for updating and deleting the selected record. When the user clicks on the Row Header of a row then the data present in the cell of the row is stored into the TextBoxes. The `DisplayData()` method fills in the data in the DataGridView.
- The `Clear()` method clears the data present in the TextBox as well as in the `ID(int)` variable.



### 5.3 NAVIGATION USING DATASOURCE IN ADO.NET

- 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.

### 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 5.4 provides the properties of the `DataSourceView` class:

**Table 5.4: Properties of the `DataSourceView` class**

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

### Demo

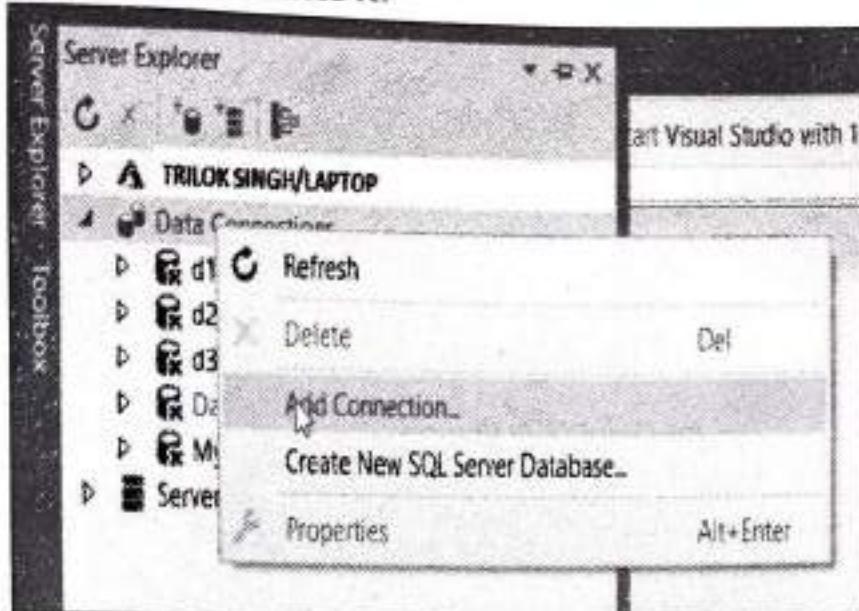
#### Database Record Navigation Example Using ADO.NET

- This article demonstrates a Database Record Navigation Example Using ADO.NET. The following example uses VB.NET. While the C# code is available here. Basically, .NET Framework provides the ADO.NET library that comprises classes and methods for connecting the application with a database and performing various operations on it.

#### Creating Database in SQL Server:

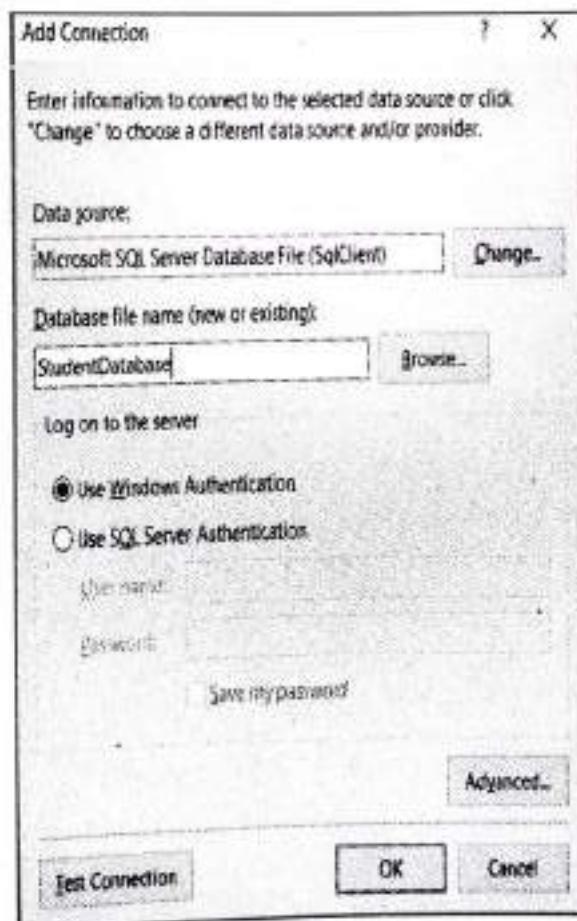
- At first, start Visual Studio and create a Windows Forms application in VB.NET. The following application is created using Visual Studio 2019. Once, the application is created, select *Server Explorer* from the *View* menu. Now click on the *Server Explorer* and then right-click on the *Add Connection* option.

- The following figure demonstrates it:



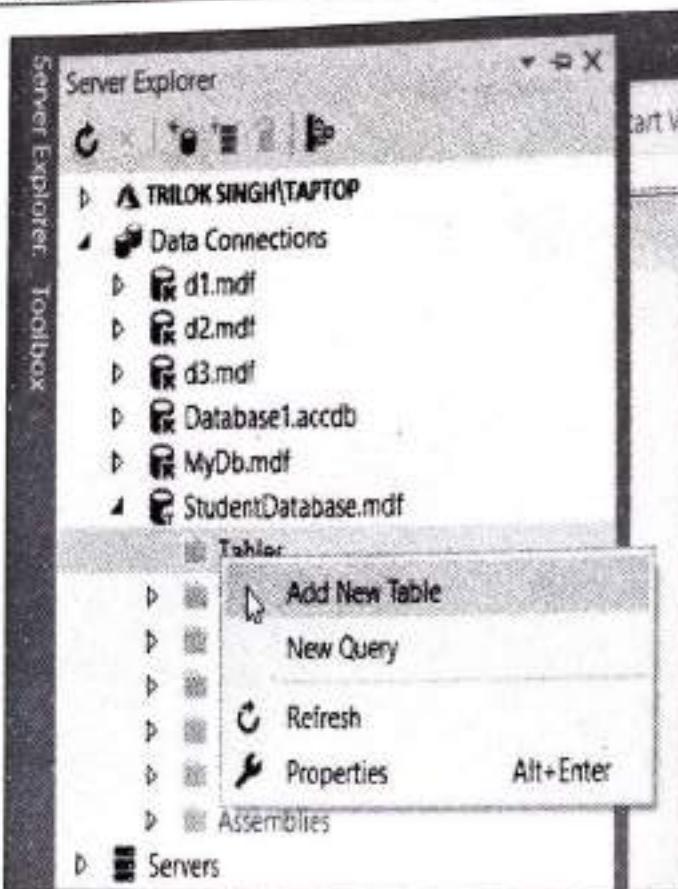
### Add Connection in SQL Server

- After that, specify the name of the database and click on OK as the following figure shows.

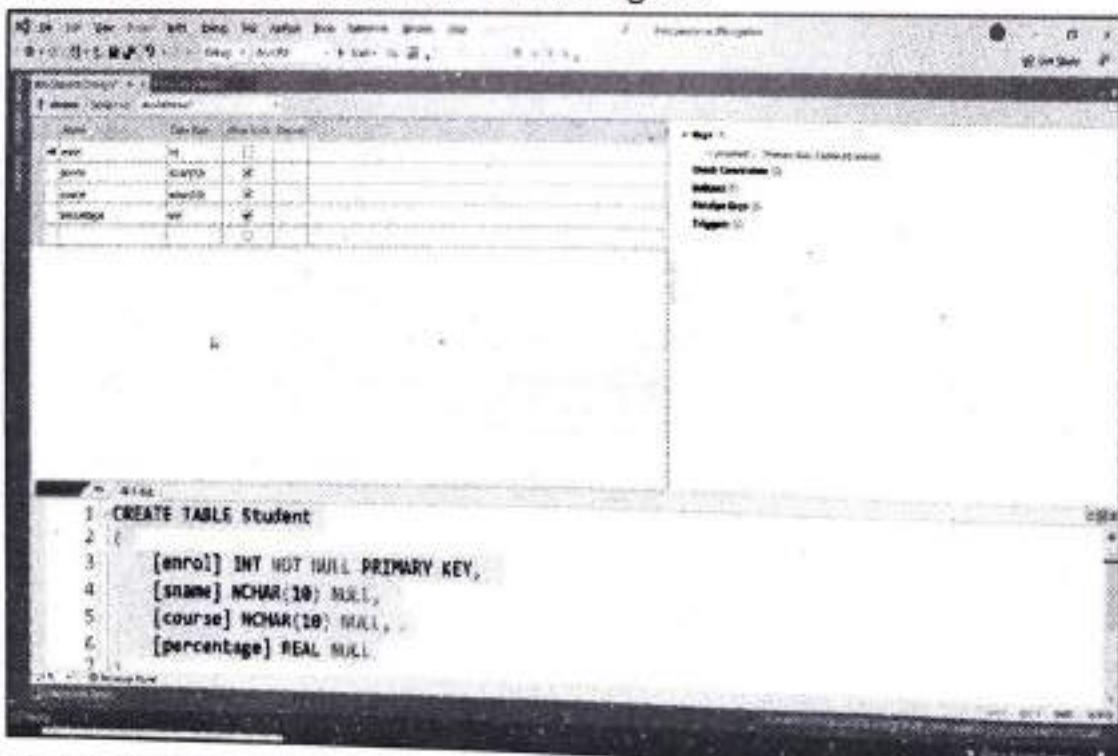


### Create a New Database in SQL Server

- If the database doesn't exist, click on the "Yes" option to create it. Now the database is visible in the Server Explorer. Further, expand the database and click on the Tables folder. After that right-click on the Add New Table option. The following figure demonstrates it.



- Add New Table in the SQL Server Database.
- Further, create a new table as shown in the figure.



### Creating a Table in SQL Server:

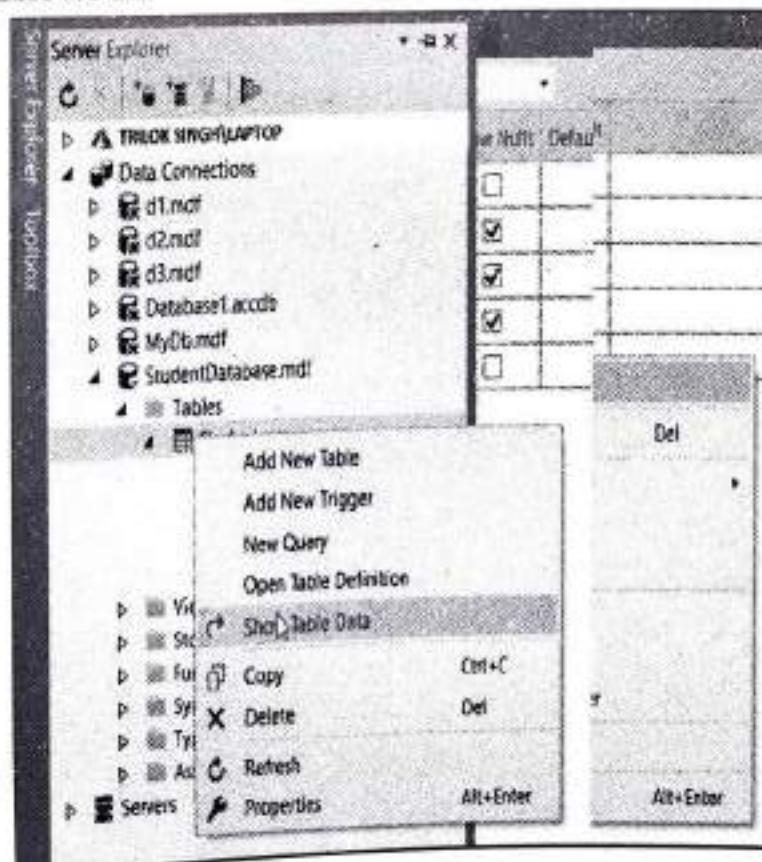
- Now click on the Update button and then click on the Update Database. When you click on the Refresh option in the Server Explorer, the database table becomes visible as shown below.



## Database Table for Database Record Navigation Example

### Creating a Database Table

- After that right-click on the Student table and select the option Show Table Data. The following figure shows it.



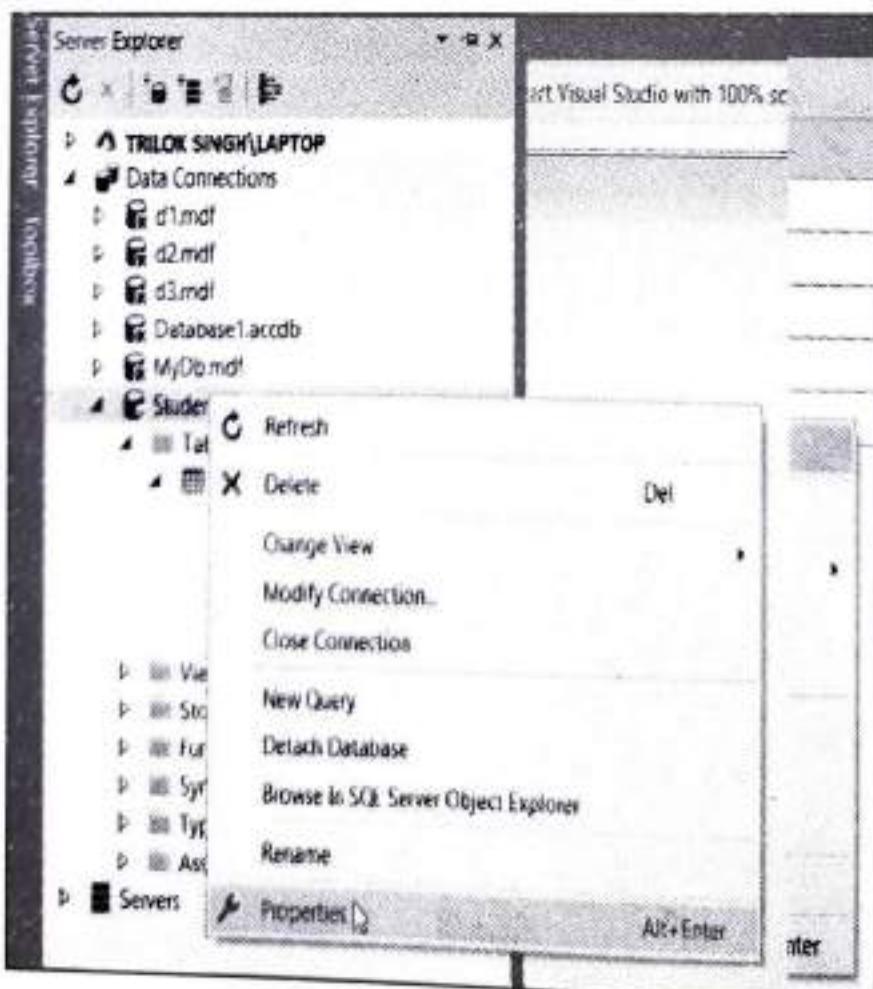
## Adding Records in the Table for Database Record Navigation Example

- Once the database table opens, add some records, save the table data and close it.

enrol	sname	course	percentage
101	Ankit	MCA	78
102	Himanshi	MCA	89
103	Kirti	BCA	85
104	Shubham	BCA	75
105	Robin	BCA	88
106	Ganga	BCA	87
107	Simran	MCA	92
108	Palak	MCA	80
109	Vishal	BCA	86
110	Minal	BCA	73
*	NULL	NULL	NULL

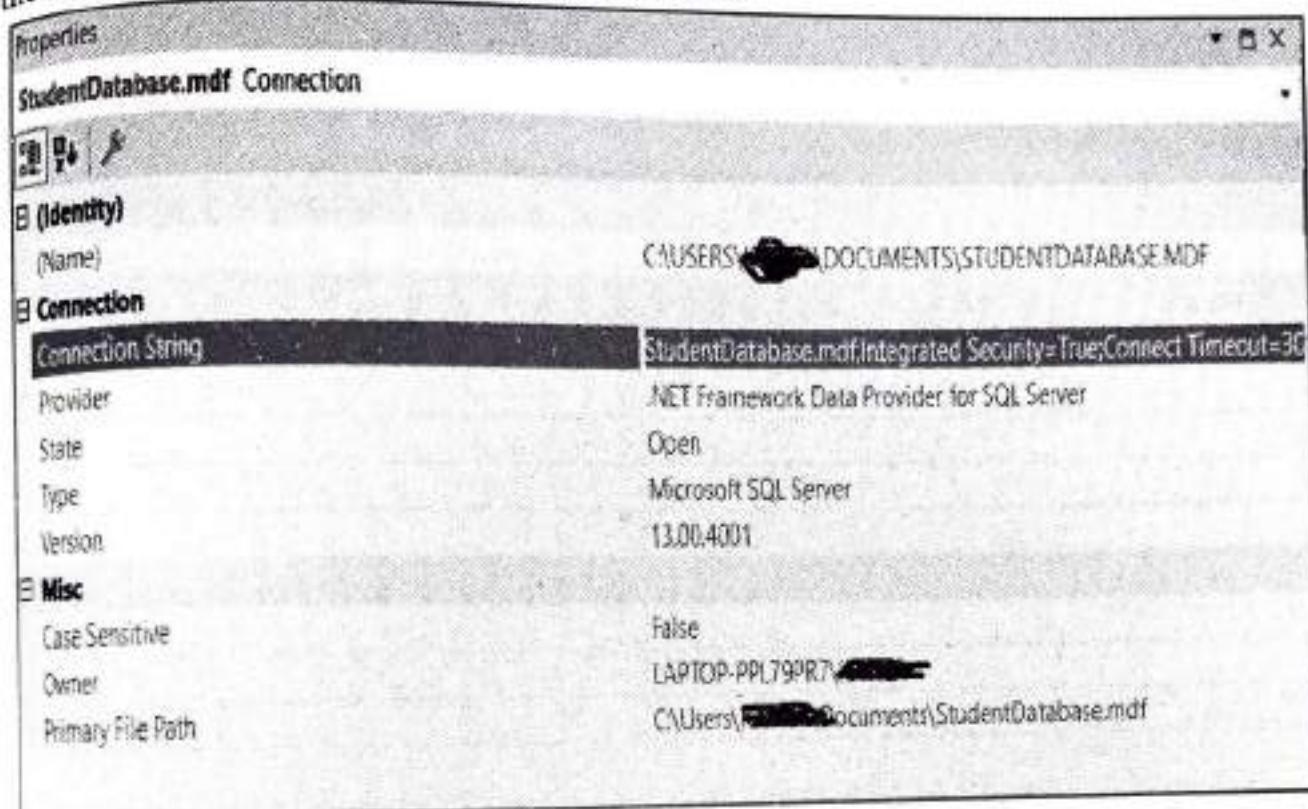
### Database Table Records:

- Further, right-click on the database in Server Explorer and click on the Properties option as shown below.

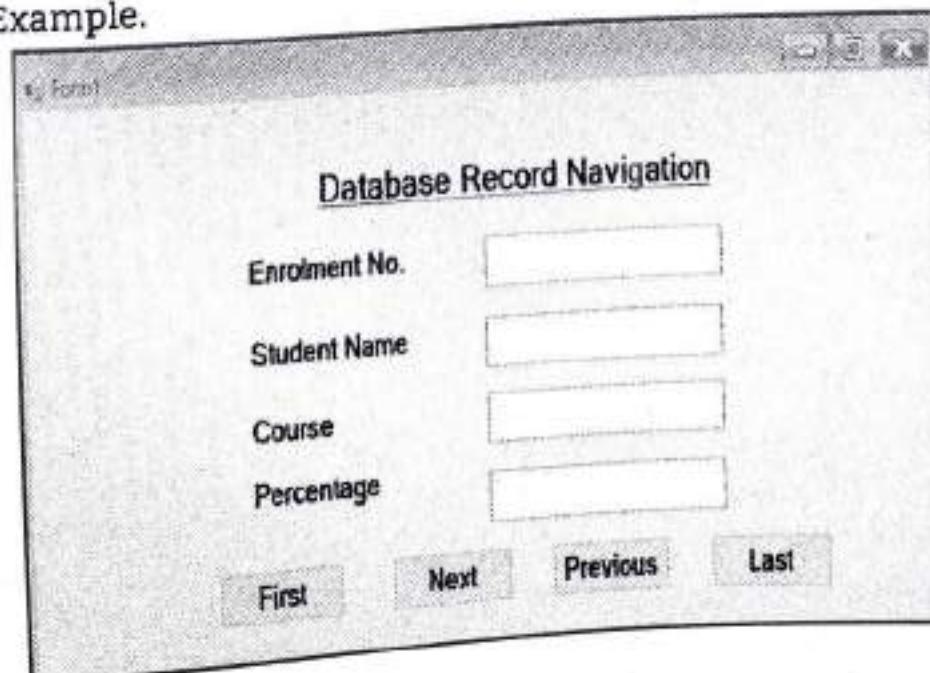


**Select Properties Option:**

- Now select the value of the Connection String property and copy it. When we create a Connection object, we need to specify the connection string. Therefore, we will use the value of the connection string in the code.

**Connection String for the Database:**

- Design a Form for Database Record Navigation Example.
- Once our database is ready, we can develop the application. Therefore, we can start with the Form Design. The following figure shows the Form for Database Record Navigation Example.



### Example 13: Form for Database Record Navigation

#### Complete Code

- After that, click on the form and write the code for connecting with the database and the corresponding buttons.

```
imports System.Data
imports System.Data.SqlClient
public Class Form1
 Dim con As SqlConnection
 Dim da As SqlDataAdapter
 Dim ds As DataSet
 Dim i As Integer = 0
 Dim n As Integer
 Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
 con = New SqlConnection("Data
Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename=C:\Users\KAVITA\Docu
ments\StudentDatabase.mdf;Integrated Security=True;Connect
Timeout=30")
 con.Open()
 da = New SqlDataAdapter("select * from Student", con)
 ds = New DataSet()
 da.Fill(ds, "Student")
 n = ds.Tables("Student").Rows.Count
 Dim dr As DataRow
 dr = ds.Tables("Student").Rows(0)
 TextBox1.Text = dr(0).ToString
 TextBox2.Text = dr(1).ToString
 TextBox3.Text = dr(2).ToString
 TextBox4.Text = dr(3).ToString
 End Sub
 Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
 Dim dr As DataRow
 dr = ds.Tables("Student").Rows(0)
 TextBox1.Text = dr(0).ToString
 TextBox2.Text = dr(1).ToString
 TextBox3.Text = dr(2).ToString
 TextBox4.Text = dr(3).ToString
 End Sub
```

```
private Sub Button4_Click(sender As Object, e As EventArgs) Handles
Button4.Click
 Dim dr As DataRow
 dr = ds.Tables("Student").Rows(n - 1)
 TextBox1.Text = dr(0).ToString
 TextBox2.Text = dr(1).ToString
 TextBox3.Text = dr(2).ToString
 TextBox4.Text = dr(3).ToString
End Sub
Private Sub Button2_Click(sender As Object, e As EventArgs) Handles
Button2.Click
 Dim dr As DataRow
 If (i = n - 1) Then
 Button4.PerformClick()
 Else
 i = i + 1
 dr = ds.Tables("Student").Rows(i)
 TextBox1.Text = dr(0).ToString
 TextBox2.Text = dr(1).ToString
 TextBox3.Text = dr(2).ToString
 TextBox4.Text = dr(3).ToString
 End If
End Sub
Private Sub Button3_Click(sender As Object, e As EventArgs) Handles
Button3.Click
 Dim dr As DataRow
 If (i = 0) Then
 Button1.PerformClick()
 Else
 i = i - 1
 dr = ds.Tables("Student").Rows(i)
 TextBox1.Text = dr(0).ToString
 TextBox2.Text = dr(1).ToString
 TextBox3.Text = dr(2).ToString
 TextBox4.Text = dr(3).ToString
 End If
End Sub
End Class
```

**Output:**

Database Record Navigation

Enrolment No.	101
Student Name	Ankit
Course	MCA
Percentage	78

First    Next    Previous    Last

**Summary****ADO.NET**

- ADO.NET is a set of classes (a framework) to interact with data sources such as databases and XML files. ADO is the acronym for ActiveX Data Objects. It allows us to connect to underlying data or databases. It has classes and methods to retrieve and manipulate data.
- The following are a few of the .NET applications that use ADO.NET to connect to a database, execute commands and retrieve data from the database.
  - ASP.NET Web Applications
  - Console Applications
  - Windows Applications.

**Various Connection Architectures**

- There are the following two types of connection architectures:
  1. **Connected architecture:** The application remains connected with the database throughout the processing.
  2. **Disconnected architecture:** The application automatically connects/disconnects during the processing. The application uses temporary data on the application side called a DataSet.

**Understanding ADO.NET and its class library**

- We can see that there are various types of applications (Web Application, Console Application, Windows Application and so on) that use ADO.NET to connect to databases (SQL Server, Oracle, OleDb, ODBC, XML files and so on).

**Important Classes in ADO.NET**

- We can also observe various classes in the preceding diagram. They are:
  1. Connection Class
  2. Command Class
  3. DataReader Class
  4. DataAdaptor Class
  5. DataSet Class

[S-23]

- Connection Class:** In ADO.NET, we use these connection classes to connect to the database. These connection classes also manage transactions and connection pooling.
- Command Class:** The Command class provides methods for storing and executing SQL statements and Stored Procedures. The following are the various commands that are executed by the Command Class.
  - **ExecuteReader:** Returns data to the client as rows. This would typically be an SQL select statement or a Stored Procedure that contains one or more select statements. This method returns a DataReader object that can be used to fill a DataTable object or used directly for printing reports and so forth.
  - **ExecuteNonQuery:** Executes a command that changes the data in the database, such as an update, delete, or insert statement, or a Stored Procedure that contains one or more of these statements. This method returns an integer that is the number of rows affected by the query.
  - **ExecuteScalar:** This method only returns a single value. This kind of query returns a count of rows or a calculated value.
  - **ExecuteXMLReader:** (SqlClient classes only) Obtains data from an SQL Server 2000 database using an XML stream. Returns an XML Reader object.
- DataReader Class:** The DataReader is used to retrieve data. It is used in conjunction with the Command class to execute an SQL Select statement and then access the returned rows.
- DataAdapter Class:** The DataAdapter is used to connect DataSets to databases. The DataAdapter is most useful when using data-bound controls in Windows Forms, but it can also be used to provide an easy way to manage the connection between your application and the underlying database tables, views and Stored Procedures.
- DataSet Class:** The DataSet is the heart of ADO.NET. The DataSet is essentially a collection of DataTable objects. In turn each object contains a collection of DataColumn and DataRow objects. The DataSet also contains a Relations collection that can be used to define relations among Data Table Objects.

### How to Connect to a Database using ADO.NET?

- To create a connection, you must be familiar with connection strings. A connection string is required as a parameter to SqlConnection. A ConnectionString is a string variable (not case sensitive).
- This contains key and value pairs, like provider, server, database, userid and word as in the following:

Server="name of the server or IP Address of the server"

Database="name of the database"

userid="user name who has permission to work with database"

word="the word of userid"

### Example

#### SQL Authentication

```
String constr="server=.;database=institute;
 user id=rakesh;word=abc@123";
```

Or

```
String constr="data source=.;initial catalog=institute;
uid=rakesh;pwd=abc@213";
```

### Windows Authentication

```
String constr="server=.;database=institute;trusted_connection=true"
```

Or

```
String constr="server=.;initial catalog=institute;
integrated security=true"
```

### How to retrieve and display data from a database?

1. Create a SqlConnection object using a connection string.
2. Handle exceptions.
3. Open the connection.
4. Create a SqlCommand. To represent a SqlCommand like (select \* from studentdetails) and attach the existing connection to it. Specify the type of SqlCommand (text/storedprocedure).
5. Execute the command (use executereader).
6. Get the Result (use SqlDataReader). This is a forwardonly/readonly dataobject.
7. Close the connection
8. Process the result
9. Display the Result

You must use the System.Data.SqlClient namespace to connect to a SQL Database. In the preceding code we are using the SqlConnection class, SqlCommand class and SqlDataReader class because our application is talking to SQL Server. SQL Server only understands SQL.

### Code for connecting to Oracle Database

- If you want to connect to an Oracle database, all you need to do is to change the connection class name from SqlConnection to **OracleConnection** Command class name from SqlCommand to OracleCommand and **SqlDataReader** to **OracleDataReader** and also in the beginning use the namespace **System.Data.OracleClient**. This works for OleDb and ODBC also.

### Practice Questions

1. What is mean by ADO.NET?
2. Write Steps to connect to database using ADO.NET.
3. What are the Classes in ADO.NET? Explain in detail.
4. Explain Connection object and Command object.
5. Write the properties and methods of DataTable class.
6. Explain DataAdapter Object in detail.
7. What is DataGridView data binding?
8. Explain the Navigation in ADO.NET using data source.
9. Write the properties of DataSourceView class.
10. Write a program to create a Form for Database Record Navigation.
11. How to retrieve and display data from a database?



**Solved Question Papers**  
**Summer 2022**

Time: 2<sup>1/2</sup>

Max. Marks: 70

**Q.1 Attempt any Eight of the following:**

[8 × 2 = 16]

- (a) What is the use of CLR?

Ans. Refer to Section 1.2.

- (b) What is CTS?

Ans. Refer to Section 1.2.

- (c) Enlist any two operator in vB.net?

Ans. Refer to Section 2.2.1.

- (d) Explain following Function?

(i) MessageBox()

(ii) InputBox()

Ans. Refer to Section 2.3.4.

- (e) Explain 'this' keyword in C#?

Ans. Refer to Section 3.2.1.

- (f) Explain constructor and destructors In C#?

Ans. Refer to Section 3.2.3.

- (g) Explain server object?

Ans. Refer to Section 4.6.

- (h) Explain the type of menu control?

Ans. Refer to Section 4.10.4.2.

- (i) Explain connected and disconnected Architecture in ADO .net?

Ans. Refer to Section 5.1.3.

- (j) Explain Timer control in vb.net?

Ans. Refer to Section 2.3.1.

**Q.2 Attempt any Four of the following:**

[4 × 4 = 16]

- (a) Explain Architecture of .net fromework?

Ans. Refer to Section 1.2.

- (b) What are HTML control?

Ans. Refer to Section 4.4.

- (c) Explain Asp.net basic control?

Ans. Refer to Section 4.7.

- (d) What is connection object in ADO.net?

Ans. Refer to Section 5.1.1.

- (e) Explain DataReader in ADO.net?

Ans. Refer to Section 5.1.5.

**Q.3 Attempt any Four of the following:****[4 × 4 = 16]**

- (a) Write a vb .net program for find max number among Entered two number.

**Ans.** Refer to Section 2.2.1.

- (b) Write a vb .net program to check whether Enter string is palindrome or not.

**Ans.** Refer to Section 2.2.3.

- (c) Write a vb .net program to accept a number from user through input box and display its multiplication table into list box?

**Ans.** Refer to Sections 2.3.1 and 2.3.2.

- (d) Write a program in C# .net for sum of two number.

**Ans.** Refer to Section 3.1.1.

- (e) Write a program in C# .net to reverse given number.

**Ans.** Refer to Section 3.1.1.

**Q.4 Attempt any Four of the following:****[4 × 4 = 16]**

- (a) What is command object?

**Ans.** Refer to Section 5.1.2.

- (b) Explain Event handing in ASP .net?

**Ans.** Refer to Section 4.9.1.

- (c) Write a C# program to swap two number.

**Ans.** Refer to Section 3.1.

- (d) Write a vb .net program to move the text "Pune university" continuously from left to right.

**Ans.** Refer to Example 10, Chapter 2.

- (e) Write a C# program for multiplication of matrix.

**Ans.** Refer to Section 3.1.3.

**Q.5 Write a short note on Any Two of the following:****[2 × 3 = 6]**

- (a) Method overloading in C#.

**Ans.** Refer to Section 3.2.7.

- (b) Validation Control in ASP .net.

**Ans.** Refer to Section 4.10.2.

- (c) Explain Data type in vb .net.

**Ans.** Refer to Section 2.2.2.

■■■

**Winter 2022****Time: 2 $\frac{1}{2}$** **Max. Marks: 70****Q.1 Attempt any Eight of the following (out of Ten):****[8 × 2 = 16]**

- (a) What is CTS?

**Ans.** Refer to Section 1.2.

- (b) State any two advantages of .Net.  
**Ans.** Refer to Section 1.1.
- (c) What is Event Driven Programming?  
**Ans.** Refer to Section 4.9.2.
- (d) Explain difference between menu and popup menu in .Net.  
**Ans.** Refer to Section 2.3.2.
- (e) List any two properties of Radio Button Control.  
**Ans.** Refer to Section 2.3.1.
- (f) What do you mean by value type and Reference type?  
**Ans.** Refer to Section 3.1.2.
- (g) What is boxing in C#?  
**Ans.** Refer to Section 3.1.2.
- (h) What is mean by ADO.Net?  
**Ans.** Refer to Section 5.1.
- (i) Enlist any four data types used in .Net?  
**Ans.** Refer to Section 2.2.2.
- (j) What is use of 'this' keyword in C#?  
**Ans.** Refer to Section 3.2.1.

**Q.2 Attempt any Four of the following (out of Five):****[4 × 4 = 16]**

- (a) Explain ASP .NET page life cycle in detail.  
**Ans.** Refer to Section 4.2.
- (b) What are the classes in ADO.Net. Explain in detail.  
**Ans.** Refer to Section 5.1.
- (c) How to create menus in VB.Net?  
**Ans.** Refer to Section 2.3.2.
- (d) Enlist and explain various objectives of .Net frameworks.  
**Ans.** Refer to Section 1.1.
- (e) State and explain various statements used in VB.Net.  
**Ans.** Refer to Section 2.2.3.

**Q.3 Attempt any Four of the following (out of Five):****[4 × 4 = 16]**

- (a) Write a program in C# for multiplication of two numbers.  
**Ans.** Refer to Section 3.1.1.
- (b) Write a program in C# to calculate area of a square.  
**Ans.** Refer to Section 3.1.
- (c) Write a VB .NET program to check given number is palindrome or not.  
**Ans.** Refer to Section 2.2.3.
- (d) Write a VB.NET program to print Yesterday's date on screen.  
**Ans.** Refer to Example 9, Chapter 2.

- (e)** Write a VB.NET program to display Studentid, Student Name and student course and student fees.

**Ans.** Refer to Section 2.2.3.

**Q.4 Attempt any Four of the following (out of Five):**

[ $4 \times 4 = 16$ ]

- (a) What are the HTML controls? Explain.

**Ans.** Refer to Section 4.4.

- (b) Write steps to connect to database using ADO.Net.

**Ans.** Refer to Section 5.1.1.

- (c) Explain Common Type Systems (CTS) and Common Language Specification (CLS).

**Ans.** Refer to Section 1.2.

- (d) Write a program to show list of doctors visiting to "Sahyadri Multispecialty Hospital".

**Ans.** Refer to Section 5.1.5.

- (e) Write a program to find prime numbers between 2 to 20.

**Ans.** Refer to Section 3.1.1.

**Q.5 Write a short note on Any Two of the following. (out of Three):**

[ $2 \times 3 = 6$ ]

- (a) Inheritance.

**Ans.** Refer to Section 3.2.4.

- (b) ASP.Net server controls.

**Ans.** Refer to Section 4.6.

- (c) Constructor and Destructor.

**Ans.** Refer to Section 3.2.3.

■ ■ ■

### Summer 2023

**Time:** 2 $\frac{1}{2}$

**Max. Marks:** 70

**Q.1 Attempt any Eight of the following:**

[ $8 \times 2 = 16$ ]

- (a) Enlist concatenation operators in vb.Net.

**Ans.** Refer to Section 2.2.

- (b) List properties of Array in C#.

**Ans.** Refer to Section 3.1.3.

- (c) What do you mean by constructor?

**Ans.** Refer to Section 3.2.3.

- (d) What is ADO.Net Dataset?

**Ans.** Refer to Section 5.1.3.

- (e) Write any two string functions in C#.

**Ans.** Refer to Section 3.1.4.

- (f) Enlist any four data types used in Vb.Net.  
 Ans. Refer to Section 2.2.2.
- (g) What is use of virtual keyword?  
 Ans. Refer to Section 3.2.7.
- (h) List any four common web controls.  
 Ans. Refer to Section 4.10.1.
- (i) What is use of site Mappath control.  
 Ans. Refer to Section 4.10.4.2.3.
- (j) List any four properties of combobox control.  
 Ans. Refer to Section 2.3.1.

**[4 × 4 = 16]****Q.2 Attempt any Four of the following:**

- (a) Explain data Gridview control.  
 Ans. Refer to Section 2.3.1.
- (b) Explain the architecture of ASP.Net.  
 Ans. Refer to Section 4.3.
- (c) Explain classes in ADO.Net.  
 Ans. Refer to Summary, Chapter 5.
- (d) What are the properties and methods of server object?  
 Ans. Refer to Section 4.6.
- (e) Explain message Box function in detail.  
 Ans. Refer to Section 2.3.4.

**[4 × 4 = 16]****Q.3 Attempt any Four of the following:**

- (a) Write Vb.Net program to accept character from user and check whether it is vowel or not.  
 Ans. Refer to Example 2, Chapter 2.
- (b) Write program in C# to create function to find factorial of given number.  
 Ans. Refer to Section 3.1.4.
- (c) Write Vb.Net program to accept the details of Employee (Eno, Ename, salary).  
 Ans. Refer to Program 5, Chapter 2.
- (d) Write C# program to find armstrong number of given number.  
 Ans. Refer to Section 3.1.1.
- (e) Write Vb.Net program to display today's date on the screen.  
 Ans. Refer to Section 2.3.1.

**Q.4 Attempt any Four of the following:**

- (a) Explain advantages of Dot.Net.  
 Ans. Refer to Section 1.1.

**[4 × 4 = 16]**

(b) Explain any four properties of text Box control.

**Ans.** Refer to Section 2.3.1.

(c) Explain pop menus in VB.Net.

**Ans.** Refer to Section 2.3.2.

(d) Write a c# program to find length of string.

**Ans.** Refer to Program 1, Chapter 3.

(e) Write a c# program to calculate area of circle.

**Ans.** Refer to Example from Section 3.1.

[ $2 \times 3 = 6$ ]

**Q.5** Write a short note on any Two of the following:

(a) Event Driven programming

**Ans.** Refer to Section 1.4.

(b) JIT compilers

**Ans.** Refer to Section 1.2.4.

(c) Method overloading

**Ans.** Refer to Section 3.2.7.



**OUR MOST RECOMMENDED  
TEXT BOOKS FOR T.Y. B.B.A. : SEMESTER-VI**

- Essentials of E-Commerce : Dr. Gautam Bapat
- Management Information System : Dr. Jayant Oke
- Business Project Management : Dr. Sheetal Randhir, Dr. Kabeer Kharade
- Management of Innovations and Sustainability : Dr. Leena Modi (Gandhi)
- International Brand Management : Dr. Shaila Bootwala, Dr. Sadia Merchant
- Cases in Marketing Management + Project : Dr. Shaila Bootwala, Uzma Shaikh, Dr. Raisa Shaikh, Dr. Farzana Shaikh
- Financial Management : Archana Vechalekar, Dr. Smita Wadaskar, Mrs. Rekha Kankariya
- Cases in Finance + Project : Dr. Ameya Anil Patil, Dr. Swami
- Global Human Resource Management : Dr. Leena Modi (Gandhi), Dr. Shalaka Parker, Mrs. Viral Ahire
- Recent Trends and HR Accounting + Project : Ankita Bhatt, Karan Randive, Dr. Anamika Ghosh

**OUR MOST RECOMMENDED  
TEXT BOOKS FOR T.Y. B.B.A. (I.B.) : SEMESTER-VI**

- New Venture Creation & Start-Ups : Arati Oturkar, Dr. Jyoti Joshi
- International Project Management : Dr. Devangi Deore, Dr. Nutan Thoke
- Decision Making and Risk Management : Dr. Ameya Anil Patil
- Management of Agribusiness and Agri Export : Dr. Roopali Sheth, Asmita Kulkarni, Dr. Nutan Thoke
- International Service Management : Diya Tanmay Devare
- International HRM : Rutuja Purohit, Pasmina Doshi

**OUR MOST RECOMMENDED  
TEXT BOOKS FOR T.Y. B.B.A. (C.A) : SEMESTER-VI**

- Recent Trends in IT : Dr. Pallavi Bulakh, Mrs. Meenal Jabde
- Software Testing : Dr. Aruna A. Deoskar, Dr. Jyoti J. Malhotra
- Advanced Java : Dr. Ms. Manisha Bharambe
- Android Programming : Kamil Ajmal Khan
- DOT NET Framework : Dr. Anjali P. Kalkar, Prof. Nutan P. Joshi, Prof. Piyush Dixit

**BOOKS AVAILABLE AT**

**PRAGATI BOOK CENTER - Email:** pbc@pragationline.com

- 157 Budhwar Peth, Opp. Ratan Talkies, Next To Balaji Mandir, Pune 411002 • Mobile : 9657703148
- 676/B Budhwar Peth, Opp. Jogeshwari Mandir, Pune 411002 Tel : (020) 2448 7459 • Mobile : 9657703147 / 9657703149
- 152 Budhwar Peth, Near Jogeshwari Mandir, Pune 411002 Mobile : 8087881795

**PRAGATI BOOK CORNER - Email:** niralimumbai@pragationline.com

- Apurva Building, Shop No. 1, Bhavani Shankar Road, Opp. Shardashram Society, Dadar (W), Mumbai 400028. Tel: (022) 2422 3526/6662 5254 • Mobile : 9819935759



niralipune@pragationline.com | www.pragationline.com

Also find us on [www.facebook.com/niralibooks](https://www.facebook.com/niralibooks)

@nirali.prakashan