M.C.A.
(TWO YEARS PATTERN)
SEMESTER - II (CBCS)

ADVANCED WEB TECHNOLOGIES LAB

SUBJECT CODE : MCAL24

Prof. Suhas Pednekar
Vice Chancellor
University of Mumbai, Mumbai.

Prof. Ravindra D. Kulkarni
Pro Vice-Chancellor,
University of Mumbai.

Prof. Prakash Mahanwar
Director
IDOL, University of Mumbai.

Programme Co-ordinator : **Prof. Mandar Bhanushe**
Head, Faculty of Science & Technology,
IDOL, University of Mumbai - 400 098.

Course Co-ordinator : **Mr. Shyam Mohan T**
Dep. of MCA IDOL,
University of Mumbai, Mumbai.

Course Writers : **Dr. Ghayathri J**
Associate Professor,
Kongu Arts And Science College (Autonomous).

: **Mr. Sandeep Kamble**
Assistant Professor,
Cosmopolitan's Valia College.

: **Mrs. Asha Hulsure**
Lecturer,
V.V.P Polytechnic, Solapur.

: **Ms. Priti U. Bharanuke**
Assistant Professor, M.Sc.(I.T.),
Institute of Distance and Open Learning,
The University of Mumbai .

: **Mrs. Kavita Chouk**
Assistant Professor,
Satish Pradhan Dnyanasadhana College.

May 2022, Print I

Published by
Director
Institute of Distance and Open learning ,
University of Mumbai,Vidyanagari, Mumbai - 400 098.

DTP COMPOSED AND PRINTED BY
Mumbai University Press
Vidyanagari, Santacruz (E), Mumbai - 400098.

CONTENT

*****

Syllabus

Course Code	Course Name
MCAL24	**Advanced Web Technologies**

Module	Detail Content	HRS
1	**Module: Basics of C#** Windows Forms Application, Classes and Objects, UI Controls, Inheritance, Interfaces, Abstract Classes **Self-Learning Topics**: Indexers and Strings Manipulations	4
2	**Module: Introduction to ASP.NET** Design Simple web pages(Data types, variables, operators,ASP.net Objects), Basic Server side controls,Working with CrossPage, Postback And Autopostback ,Advanced Web server controls (validation, Calendar, AdRotator, Navigation, File upload),Build an Applications using Angular JS,JQuery and NodeJS, Websites using Master Pages (creating master and content pages) **Self-Learning Topics**: Themes and skins	12
3	**Module: Database Programming in ASP.NET** Connected and disconnected Architecture of ADO.NET , Commands, Datasets, Data Readers, Data Adapters, Working with Stored Procedures,Data bound controls (DataList, DetailsView, FormView, GridView, ListView, Repeater), LINQ with ASP.NET,LINQ Introduction, Mapping data model to an Object model, Introducing query syntax, Entity Framework **Self-Learning Topics**: Charts and Data Pagers	10
4	**Module: Session Management and AJAX** Client Side State Management - View State, Query String, Cookie, Hidden Fields ,Server Side State Management Various State Management Techniques- Profiles, Session State, Application State, cache ,ASP.NET Applications with AJAX , AJAX Controls, Testing an ASP.NET Ajax application, Global.asax and Web Config,Caching **Self-Learning Topics**: Web Parts	8
5	**Module: Web Services and WCF**Creating and Consuming a XML Web Service-Simple and Database ,Creating and Consuming a WCF service – Simple and Database**Self-Learning Topics**: Caching Web service responses	6
6	**Module: ASP.NET MVC** Designing MVC	12

	application, Creating a Simple Data-Entry Application with validations, Using Automatically Implemented Properties, Using Object and Collection Initializers, Using Extension Methods, Using Lambda Expressions, Programs based on MVC Pattern, FORMS AND HTML HELPERS, Define and access the model type ,Reduce duplication in views, Specify a default layout, Pass data values to the view from the controller , Generate different content based on data values, Add a namespace to a view **Self-Learning Topics**: Xamarin application	

*****

MODULE I

1

WINDOWS FORMS APPLICATION, CLASSES AND OBJECTS, UI

Unit Structure

1.0 OBJECTIVES

This chapter would make the learners to understand:

- The basics of data types, decision making statements and keywords of C# language.
- The practical implementation of web form application.
- The concepts of defining methods and objects.
- The usage of various User Interface controls in web forms.
- The Inheritance and interfacing concepts to be applied in C# programs.

1.1 INTRODUCTION

C# is Microsoft's premier language for .NET development. It leverages time-tested features with cutting-edge innovations and provides a highly usable, efficient way to write programs for the modern enterprise computing environment. It is, by any measure, one of the most important languages of the twenty-first century.

Features of C#:

It is simple: The symbols like -> and :: in C++ are removed in this.

Object oriented: It is truly object oriented and supports Encapsulation, Inheritance

and polymorphism concepts of OOPs.

The entire C# class model is built on top of the Virtual Object System of the

.NET Framework In C#, everything is an object. There are no more global functions, variables and constants.

Type safe: Type-safety promotes robust programs. C# incorporates a number of type-safe measures:

- All dynamically allocated objects and arrays are initialized to zero.
- Use of any uninitialized variables produces an error message by the compiler.
- Access to arrays is range-checked and warned if it goes out-of-bounds.
- C# does not permit unsafe casts.
- C# enforces overflow checking in arithmetic operations.
- Reference parameters that are passed are type-safe.
- C# supports automatic garbage collection.

It is versonable*:* Making new versions of software modules work with the existing applications is known as versioning. C# provides support for versioning with the help of new and override keywords.

It is compatible: C# enforces the NET common language specifications and therefore allows inter-operation with other .NET languages.

It is Interoperable and Flexible: Although C# does not support pointers, we may declare certain classes and methods as 'unsafe' and then use pointers to manipulate them. However, these codes will not be type-safe.

Name space: It is having namespaces; a namespace to define a declarative region

Applications of C#:

It can be used for a variety of applications that are supported by the .NET platform:

- Console applications
- Windows applications
- Developing Windows controls

- Developing ASP.NET projects:
- Creating Web controls
- Providing Web services
- Developing NET component library

1.1.3 C# and .NET:

The .NET framework is responsible for executing the C# program in the system. Framework is the combination of common language runtime and a set of class libraries. The implementation of Common Language Infrastructure (CLI) is done through Common Language Runtime (CLR).

Microsoft .NET framework consists of the following main components:

- Common Language Runtime
- Base class library
- User Program interfaces

It also supports Common Type System (CTS) which sstandardizes the data types of all programming languages using .NET under the umbrella of .NET to a common data type for easy and smooth communication among these .NET languages

1.2 AN OVERVIEW ABOUT C#

The elements of Visual Studio .NET applications are mentioned below:

1) The Start Page
2) Solution Explorer window
3) Output Window
4) Class View Window
5) Code Editor Window
6) Error List Window
7) Standard ToolBar

1.2.1 A Simple program:

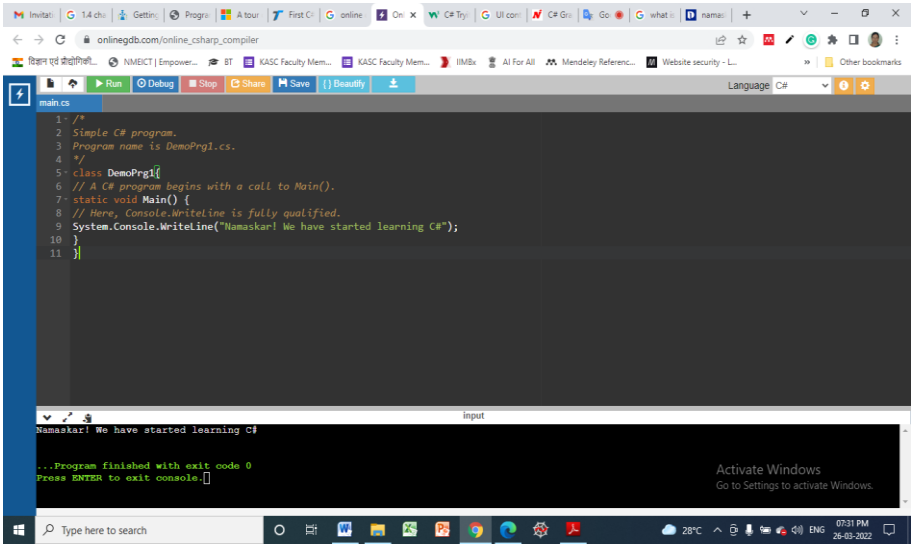


Fig 1.2 Simple C# Program

1.2.2 C# Program Structure:

```
/* Structure of C# Program  */

using System;
using System.Collections.Generic;

namespace Structure
{
    class DemoPrg2
    {
        static void Main(string[] args)
        {
            string msgs = "Structure of C# Program";

            Console.WriteLine(msgs);
        }
    }
}
```

```
Structure of C# Program


...Program finished with exit code 0
Press ENTER to exit console.
```

1. **Reference of.NET framework namespaces:** Every .NET application takes the reference of the necessary .NET framework namespaces that it is planning to use with the using (Keyword) , Ex., using System;

2. **namespace:** (Keyword) Declares the namespace for the current class Ex. Structure

3. **class:** (Keyword) Declares a class: Ex. DemoPrg2

4. **Method:** Main() is a method of DemoPrg2 class which is the entry point of the console application.

5. **String:** It is a data type.

6. **variable:** msgs is a variable that holds the value of the specified (here it is string) data type.
7. **value:** "Structure of C# Program" is the value of the message variable 'msgs'.
8. **static method:** The Console.WriteLine() is used to display a text on the console.
9. Ouput screen shows the output of the program.
10. Every line or statement in C# must end with a semicolon (;).

Compile and Run C# Program:

To see the output of the above C# program, we have to compile it and run it by pressing Ctrl + F5 or clicking the Run button or by clicking the "Debug" menu and clicking "Start Without Debugging".

Data Types:

Variable: It is used to store the data value in a storage location and takes different values during the execution time based on the input to be given.

These variables are represented through the defined data types. C# has two main categories of data types such as i) value types and ii) reference types. The other types are coming under these categories as given in Table 1.1.

S.No.	***Data Types***			***Examples datatype <variable_name> = value;***
1	**Value Types**	*Predefined Types (Simple Types)*	**1. Integers**	int a = 99;
			2. Real Numbers	float b=99.9f;
			3. Booleans	bool c = true;
			4. Characters	char d = 'E';
		User-defined Types	**1. Enumerations**	enum fruits{ apple, orange, grapes}

			2. Structure	struct Coordinate
				{
				public int x;
				public int y;
				}

2	**Reference Types**	*Predefined Types*	**1. Objects**	public class Car{
				public model { get; set; }
				public int year { get; set; }
				public string color{ get; set; }
				}

			2. Strings	string msg="Happy";

		User-defined Types	**1. Classes**	public class MyClass { }
			2. Arrays	var num = new int[]{ 1,2,3,4,5};
			3. Delegates	public delegate void MyDelegate(string msg);

			4. Interfaces	interface IFile
				{
				void ReadFile();
				void WriteFile(string text);
				}

3	**Pointers (type *var-name;)**			int x = 4000; int* y = &x;

Key words:

C# language is defined by its keywords which determine the features built into the language.

In general there are two types of keywords: reserved and contextual.

The reserved keywords shown in Table 1.2 cannot be used as names for variables, classes, or methods.

abstract	as	base	bool	break
byte	case	catch	char	checked
class	const	continue	decimal	default
delegate	do	double	else	enum
event	explicit	extern	false	finally
fixed	float	for	foreach	goto
if	implicit	in	int	interface
internal	is	lock	long	namespace
new	null	object	operator	out
override	params	private	protected	public
readonly	ref	return	sbyte	sealed
short	sizeof	stackalloc	static	string
struct	switch	this	throw	true
try	typeof	uint	ulong	unchecked
unsafe	ushort	using	virtual	volatile
void	while			

Table 1.2 Reserved Keywords

add	dynamic	from	get	global
group	into	join	let	orderby
partial	remove	select	set	value
var	where	yield		

Table 1.3 Contextual keywords

C# has some contextual keywords, shown in Table 1.3. They act as keywords in a special meaning in certain contexts. But they are not technically reserved. Apart from those contexts, they can be used as names for other program elements, such as variable names. It is a best practice to avoid using contextual keywords for other purposes.

Decision Making Statements:

if Statement:

If (condition) statement;
else statement;
	if (10<i)
	{
	Console.WriteLine("You are Correct");
	}

- Single statement targets
- else clause is optional

if(condition)
{
statement sequence
}
else
{
statement sequence
}
	if (10<i)
	{
	Console.WriteLine("You are Correct");
	}
	else
	{
	Console.WriteLine("Check the value of i");
	}

if-else-if Ladder:

If (condition)
statement;
else if (condition)
statement;
else if (condition)
statement;
...
else
statement;
	if (10<i)
	{
	Console.WriteLine("You are Correct");
	elseif (10<j)
	Console.WriteLine("You are almost Correct");
	else
	Console.WriteLine("Check the value of I and j")
	}

The for Loop:

The general form of the for loop for repeating a single statement is

for (initialization; condition; iteration)
{
statement sequence
}
	for(int n = 10; i < 25; i++)
	{
	Console.WriteLine("Value of n is:", n);
	}

The while Loop

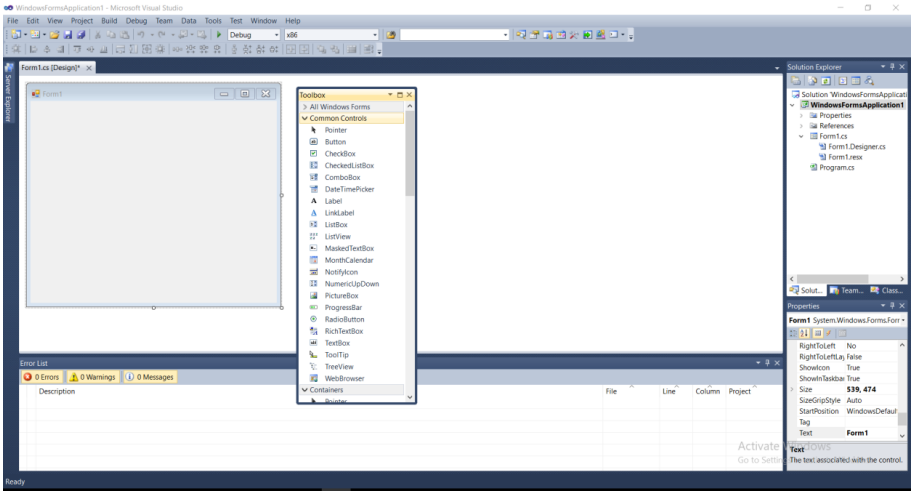
Another of C#'s loops is the while. The general form of the while loop is

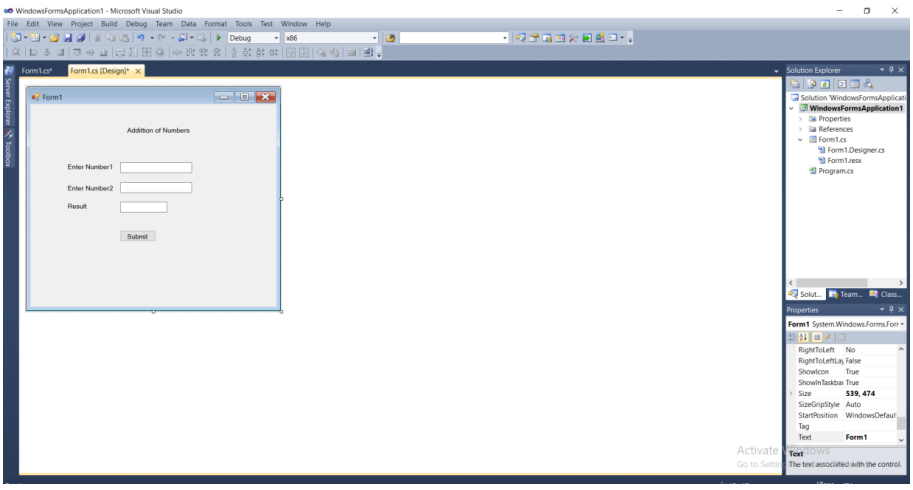
while(condition) statement	while (n<=10){
	Console.WriteLine("Value of n is :", n);
	n=n+2;
	}

do {
statements;
} while (condition);
	do{
	sum= sum+n;
	Console.WriteLine);
	n=n+2;
	} while (n <= 20);

1.3 WEB FORMS APPLICATION

Design UI based applications using basic Windows forms Controls:

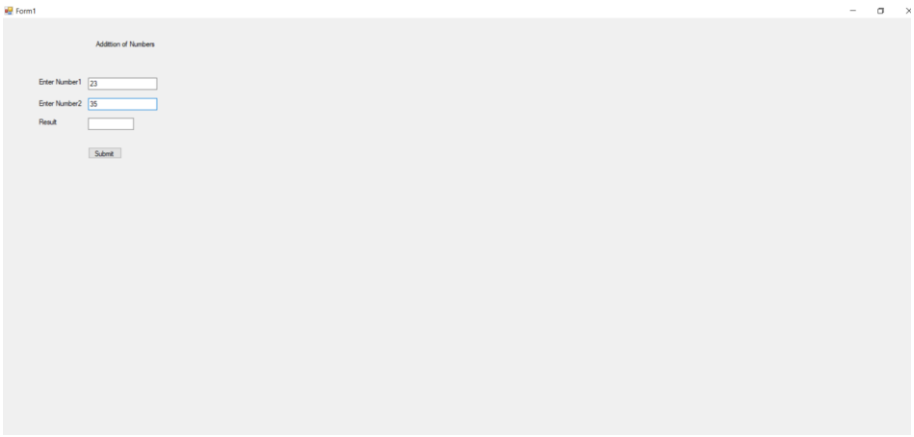


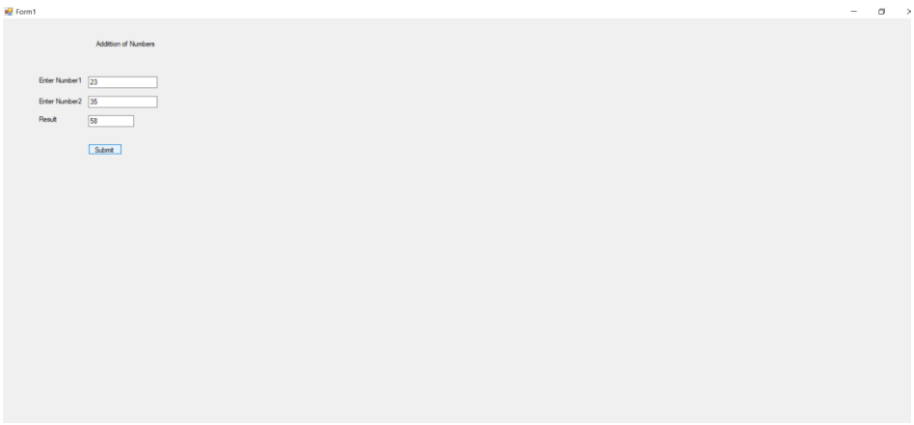


PROGRAM TO ADD TWO NUMBERS:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace WindowsFormsApplication1
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
      }

      private void button1_Click(object sender, EventArgs e)
      {
         int a, b, c;
         a = Convert.ToInt32(textBox1.Text);
         b = Convert.ToInt32(textBox2.Text);
         c = a + b;

         textBox3.Text = c.ToString();
      }
    private void label2_Click(object sender, EventArgs e)
      {
      }
   }
}
```

Output:





1.4 CLASSES AND OBJECTS

General structure of Class:

To define a class, declare the data that it contains and the code that operates on it. Simple classes contain only code or only data, most real-world classes contain both.

Data is contained in data members defined by the class, and code is contained in function members.

C# defines different kind of data and function members. For example, data members (also called fields) include instance variables and static variables and function members.

Function members include methods, constructors, destructors, indexers, events, operators, and properties.

A class is created by use of the keyword class.

General form of a simple class definition:

class classname

{

```
// declare instance variables

access type var1;

access type var2;

// declare methods

access ret-type method1(parameters)

{

// body of method

}

}
```

Defining a class:

Buildings have the information about houses, stores, offices, and so on. This class is named as Building, and it will store three items of information about a building: the number of floors, the total area, and the number of occupants.

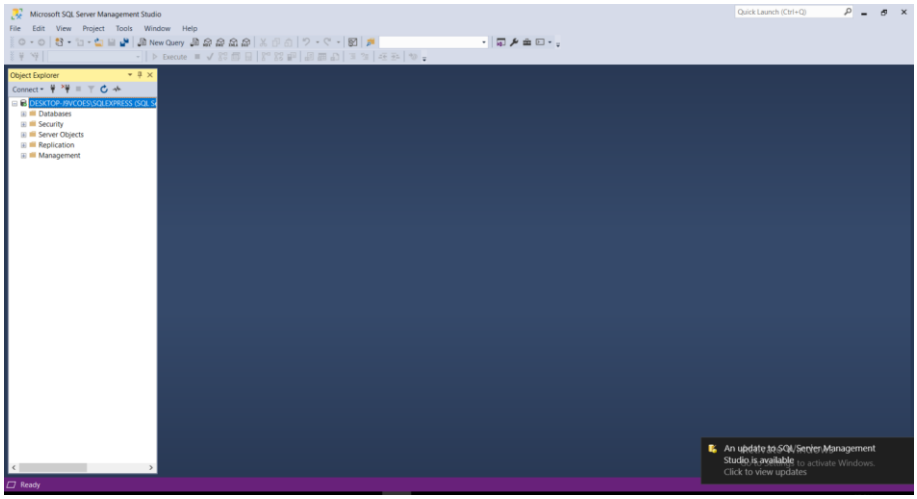
Demo program 2: Design Applications using Classes and Object

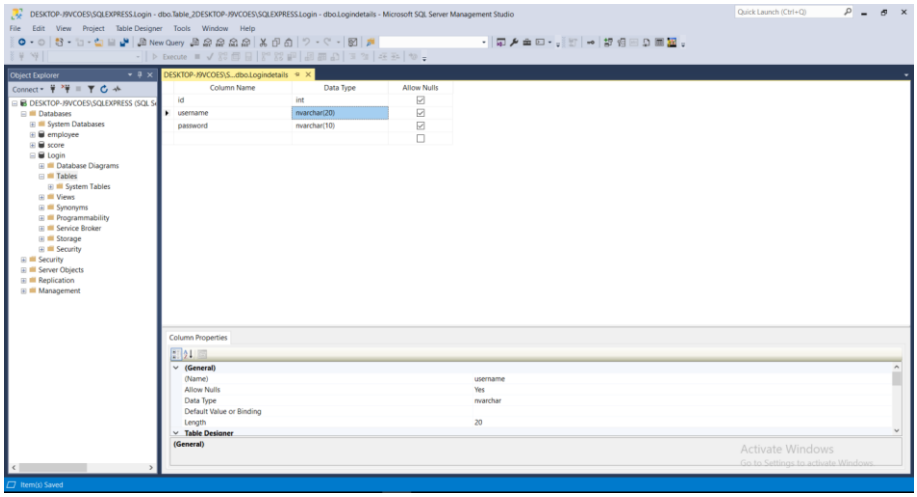
```
using System;

namespace MCAClassObjectsDemo

{

   class Program

   {

      static void Main(string[] args)

      {

         //Creating object

         Calculator calObject = new Calculator();

         Console.Write("Enter number1 : ");

         int n1 = Convert.ToInt32(Console.ReadLine());

         Console.Write("Enter number2 : ");

         int n2 = Convert.ToInt32(Console.ReadLine());

         //Accessing Calculator class member using Calculator class object

         int result1 = calObject.CalculateSum(n1, n2);

         int result2 = calObject.CalculateMinus(n1, n2);

         int result3 = calObject.CalculateMul(n1, n2);
```

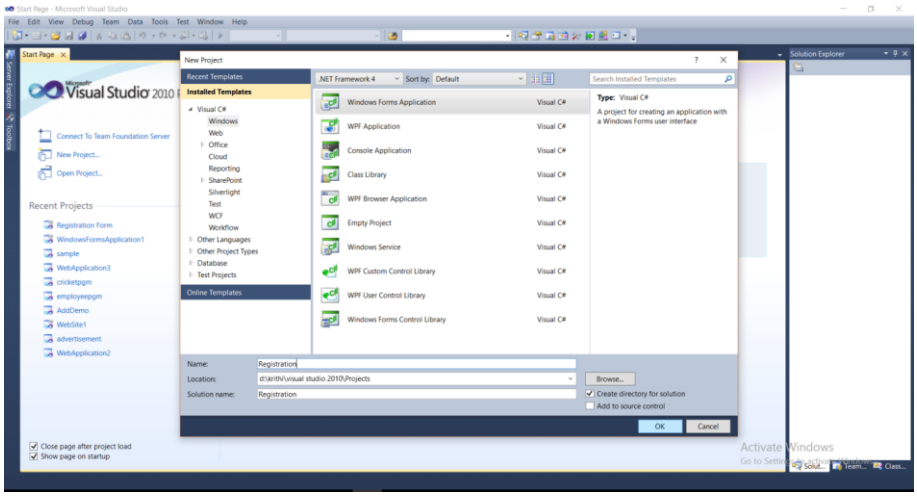
```
Console.WriteLine("The Sum of Two numbers is  : " + result1);
Console.WriteLine("The Subtraction of Two numbers is  : " + result2);
Console.WriteLine("The Multiplication of Two numbers is  : " + result3);
        Console.ReadKey();
    }
}
```

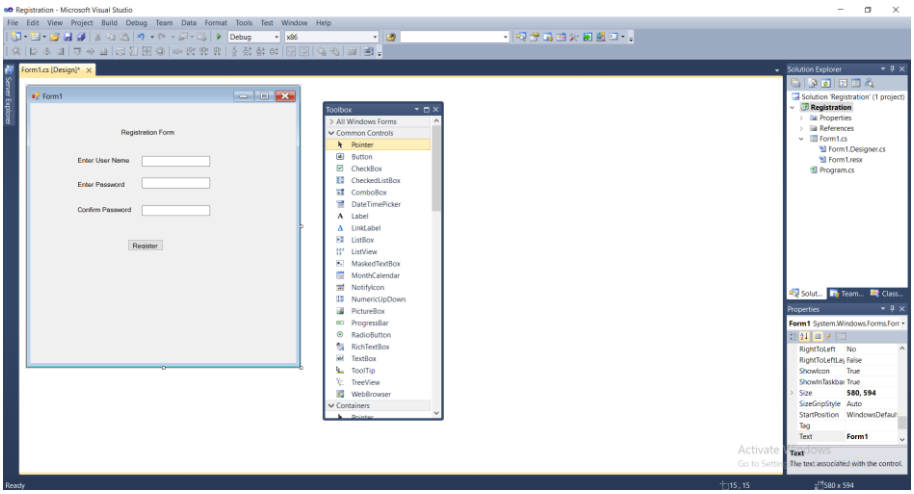
1.5 UI CONTROLS

Design a Web Application for an Organization with Registration forms and advanced controls:







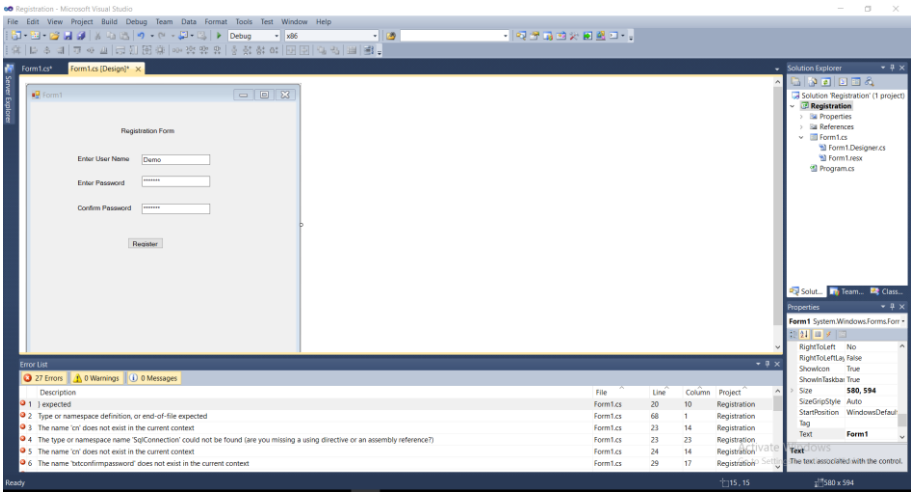


Coding:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace Registration
{
    public partial class Form1 : Form
    {
        public Form1()
        {
```

```
        InitializeComponent();
    }
    private void Form1_Load(object sender, EventArgs e)
    {
        private void Registration_Load(object sender, EventArgs e)
      {
cn = new SqlConnection(@"Data Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename=H:\Website\RegistrationAndLogin\Database.mdf;Integrated Security=True");
         cn.Open();
        }
    private void button1_Click(object sender, EventArgs e)
    {
if (txtconfirmpassword.Text != string.Empty || txtpassword.Text != string.Empty || txtusername.Text != string.Empty)
        {
if (txtpassword.Text == txtconfirmpassword.Text)
            {
cmd = new SqlCommand("select * from LoginTable where username='" + txtusername.Text + "'", cn);
                dr = cmd.ExecuteReader();
                if (dr.Read())
                {
                   dr.Close();
MessageBox.Show("Username Already exist please try another ", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
                }
                else
                {
                   dr.Close();
cmd = new SqlCommand("insert into LoginTable values(@username,@password)", cn);
cmd.Parameters.AddWithValue("username", txtusername.Text);
cmd.Parameters.AddWithValue("password", txtpassword.Text);
```

```
cmd.ExecuteNonQuery();

MessageBox.Show("Your Account is created . Please login now.", "Done", MessageBoxButtons.OK, MessageBoxIcon.Information);
                }
            }
            else
            {

MessageBox.Show("Please enter both password same ", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }
        else
        {

MessageBox.Show("Please enter value in all field.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }
    }
  }
}
```



1.6 INHERITANCE

1.6.1 Inheritance Concepts and working principles:

C# supports inheritance by allowing one class to incorporate another class into its declaration. This is done by specifying a base class when a derived class is declared.

The following class called TwoDShape stores the width and height of a twodimensional object, such as a square, rectangle, triangle, and so on.

TwoDShape can be used as a base class (that is, as a starting point) for classes that describe specific types of two-dimensional objects. For example, the following program uses TwoDShape to derive a class called Triangle. Pay close attention to the way that Triangle is declared.

Design Applications using Inheritance and Abstract Classes:

```
// A simple class hierarchy.

using System;

// A class for two-dimensional objects.

class TwoDShape

{

public double Width;

public double Height;

public void ShowDim()

{

Console.WriteLine("Width and height are " + Width + " and " + Height);

}

}

// Triangle is derived from TwoDShape.

class Triangle : TwoDShape

{

public string Style; // style of triangle

// Return area of triangle.

public double Area()

{

return Width * Height / 2;

}

// Display a triangle's style.

public void ShowStyle()

 {

Console.WriteLine("Triangle is " + Style);
```

```
}
}
class Shapes
{
static void Main()
{
        Triangle t1 = new Triangle();
        Triangle t2 = new Triangle();
        t1.Width = 4.0;
        t1.Height = 4.0;
        t1.Style = "isosceles";
        t2.Width = 8.0;
        t2.Height = 12.0;
        t2.Style = "right";
        Console.WriteLine("Info for t1: ");
        t1.ShowStyle();
        t1.ShowDim();
        Console.WriteLine("Area is " + t1.Area());
        Console.WriteLine();
        Console.WriteLine("Info for t2: ");
        t2.ShowStyle();
        t2.ShowDim();
Console.WriteLine("Area is " + t2.Area());
    }
}
```

The output from this program is shown here:

```
Info for t1:
Triangle is isosceles
Width and height are 4 and 4
```

Area is 8

Info for t2:

Triangle is right

Width and height are 8 and 12

Area is 48

1.7 INTERFACES

In object-oriented programming it is sometimes helpful to define what a class must do, but not how it will do it. You have already seen an example of this: the abstract method. An abstract method declares the return type and signature for a method, but provides no implementation. A derived class must provide its own implementation of each abstract method defined by its base class. Thus, an abstract method specifies the interface to the method, but not the implementation. Although abstract classes and methods are useful, it is possible to take this concept a step further. In C#, you can fully separate a class' interface from its implementation by using the keyword interface. Interfaces are syntactically similar to abstract classes. However, in an interface, no method can include a body. That is, an interface provides no implementation whatsoever. It specifies what must be done, but not how. Once an interface is defined, any number of classes can implement it. Also, one class can implement any number of interfaces.

To implement an interface, a class must provide bodies for the methods described by the interface. Each class is free to determine the details of its own implementation. Thus, two classes might implement the same interface in different ways, but each class still supports the same set of methods. Therefore, code that has knowledge of the interface can use objects of either class since the interface to those objects is the same. By providing the interface, C# allows you to fully utilize the "one interface, multiple methods" aspect of polymorphism. An interface offers an alternative to an abstract class for creating contracts among classes and their clients. These contracts are made manifest using the interface keyword, which declares a reference type that encapsulates the contract. An interface is like a class that has only abstract methods.

Interfaces are declared by using the interface keyword. Here is a simplified form of an interface declaration:

interface interface-name

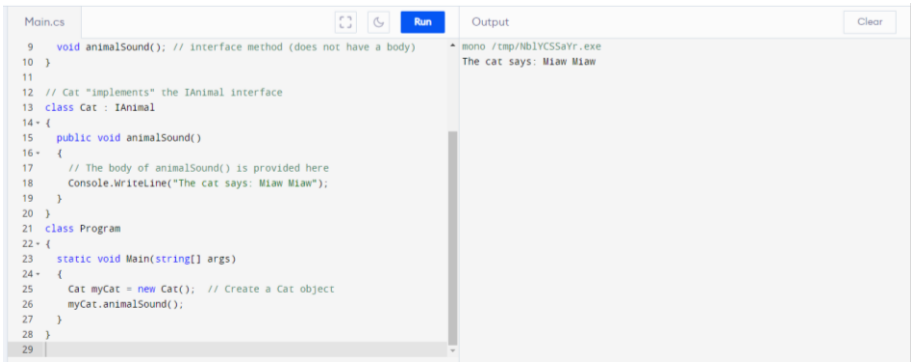
{

return-type method-name(list of parameters);

}

The interface-name of the interface is specified by name. Methods are declared using only their return type and signature. They are, essentially, abstract methods.

```
// Interface
interface IAnimal
{
  void animalSound(); // interface method (does not have a body)
}

// Cat "implements" the IAnimal interface
class Cat : IAnimal
{
  public void animalSound()
  {
    // The body of animalSound() is provided here
    Console.WriteLine("The cat says: Miaw Miaw");
   }
}
class Program
{
  static void Main(string[] args)
  {
    Cat myCat = new Cat();  // Create a Cat object
    myCat.animalSound();
  }
}
```



1.8 ABSTRACT CLASSES

In the hierarchical inheritance applications, an application can have a single base class and number of different derived classes. The base class acts as base for all the other but only the required information is visible to the derived class. The information to be hidden from the users of the derived classes can be implemented using Abstract classes. These classes contain abstract methods that are inherited by other classes that provide more functionality. The features of Abstract Classes are:

- The keyword abstract is used to create abstract class.
- Some of the methods of the abstract class can also contain the keyword abstract.
- No object can be created of the abstract class i.e.it cannot be instantiated.
- The abstract methods in the abstract class are implemented actually only in the derived classes.

Design Applications using Inheritance and Abstract Classes:

```
using System;
namespace AbstractionDemo
{
  abstract class Shape
  {
    public abstract double area();
  }

  class Circle: Shape
  {
    private double radius;
    public Circle( double r)
    {
      radius = r;
    }

    public override double area ()
    {
      return (3.14*radius*radius);
    }
  }
```

```
class Square: Shape
{
  private double side;
  public Square( double s)
  {
    side = s;
  }
  public override double area ()
  {
    return (side*side);
  }
}
 class Triangle: Shape
{
  private double tbase;
  private double theight;
  public Triangle( double b, double h)
  {
    tbase = b;
    theight = h;
  }
  public override double area ()
  {
    return (0.5*tbase*theight);
  }
}
class Test
{
  static void Main(string[] args)
  {
    Circle c = new Circle(5.0);
    Console.WriteLine("Area of Circle = {0}", c.area());
    Square s = new Square(2.5);
    Console.WriteLine("Area of Square = {0}", s.area());
```

```
        Triangle t = new Triangle(2.0, 5.0);
        Console.WriteLine("Area of Triangle = {0}", t.area());
    }
  }
}
```

The output of the above program is as follows:

Area of Circle = 78.5

Area of Square = 6.25

Area of Triangle = 5

- The abstract class is shape and it contains the abstract method area().
- The class Circle is inherited from Shape. It has a private data member radius. The parameterized constructor assigns a value to radius. The function area() calculates the area of the circle and returns that value
- The class Square inherits from Shape. It has a private data member side. The parameterized constructor assigns a value to side. The function area() calculates the area of the square and returns that value.
- The class Square inherits from Shape. It has a private data member side. The parameterized constructor assigns a value to side. The function area() calculates the area of the square and returns that value.
- The class Triangle inherits from Shape. It has private data members tbase and theight. The parameterized constructor assigns a value to tbase and theight. The function area() calculates the area of the triangle and returns that value.
- The function main() contains the objects c, s and t for classes Circle, Square and Triangle respectively. Then the areas of the circle, square and triangle are printed using the function area().

1.9 SUMMARY

This chapter describes about the C# language which is one of the language in the .NET framework family. This language supports CLS and CTS. It is very simple object oriented language which supports interoperability.

C# consists of reserved keywords to use while writing the codes. To do the decision making if, if.. else, if.. else…if ladder , for loop, while loop, do—while loop are used with following the defined syntax.

Classes can be complex representations and abstractions of things in the real world.The structure is of class contains data and function members The individual instances of class are known as objects. A class has both properties and behaviors.

The different types of UI controls like textbox, buttons are useful to create the GUI based environment.

Inheritance concept is used in C# to derive the properties of the base class to the another class called derived class.

An interface is a contract that guarantees to a client how a class or struct will behave. When a class implements an interface, it supports the methods, properties, events, and indexers of the named interface.

An abstract method has no implementation. It creates a method name and signature that must be implemented in all derived classes.

1.10 UNIT END EXERCISES

1. State the features and applications of C#.
2. Write a program in C sharp to find odd and even numbers
3. Design a web form for arithmetic operations
4. Write an application to design employee details web form.

1.11 LIST OF REFERENCES

1. Spaanjaars, Imar. Beginning ASP. NET 4.5. 1: in C# and VB. John Wiley & Sons, 2014. ISBN: 1861009038
2. Schildt, Herbert. C# 4.0: the complete reference. Tata McGraw-Hill Education, 2010
3. Balagurusamy E, Programming in C#, Tata McGraw-Hill Education
4. https://www.wideskills.com/csharp/overview-csharp
5. https://www.onlinegdb.com/online_csharp_compiler

*****

MODULE II

2

INTRODUCTION TO ASP.NET

Unit Structure

2.0 OBJECTIVES

.NET Framework is a technology that supports building and running Windows apps and web services. .NET Framework is designed to fulfill the following objectives:

- Provide a consistent, object-oriented programming environment whether object code is stored and executed locally, executed locally but web-distributed, or executed remotely.
- Provide a code-execution environment that:
  - Minimizes software deployment and versioning conflicts.
  - Promotes safe execution of code, including code created by an unknown or semi-trusted third party.
  - Eliminates the performance problems of scripted or interpreted environments.
- Make the developer experience consistent across widely varying types of apps, such as Windows-based apps and Web-based apps.
- Build all communication on industry standards to ensure that code based on .NET Framework integrates with any other code.

2.1 INTRODUCTION

.NET Framework consists of the common language runtime (CLR) and the .NET Framework class library. The common language runtime is the foundation of .NET Framework. Think of the runtime as an agent that manages code at execution time, providing core services such as memory management, thread management, and remoting, while also enforcing strict type safety and other forms of code accuracy that promote security and robustness. In fact, the concept of code management is a fundamental principle of the runtime. Code that targets the runtime is known as managed code, while code that doesn't target the runtime is known as unmanaged code. The class library is a comprehensive, object-oriented

collection of reusable types that you use to develop apps ranging from traditional command-line or graphical user interface (GUI) apps to apps based on the latest innovations provided by ASP.NET, such as Web Forms and XML web services.

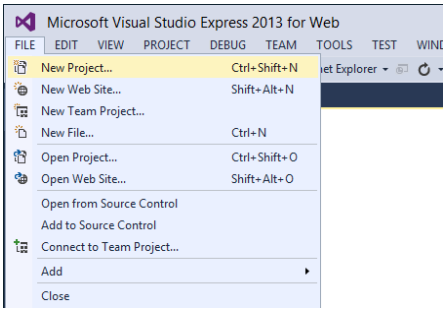
.NET Framework can be hosted by unmanaged components that load the common language runtime into their processes and initiate the execution of managed code, thereby creating a software environment that exploits both managed and unmanaged features. .NET Framework not only provides several runtime hosts but also supports the development of third-party runtime hosts.

Creating a Web application project and a Page:

In this part of the walkthrough, you will create a Web application project and add a new page to it. You will also add HTML text and run the page in your browser.

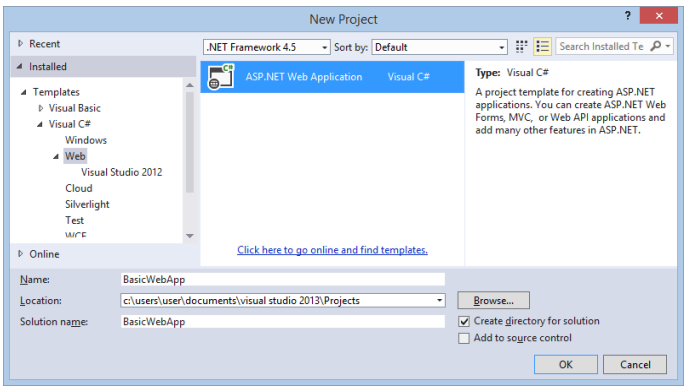
To create a Web application project:

1. Open Microsoft Visual Studio.
2. On the **File** menu, select **New Project**.

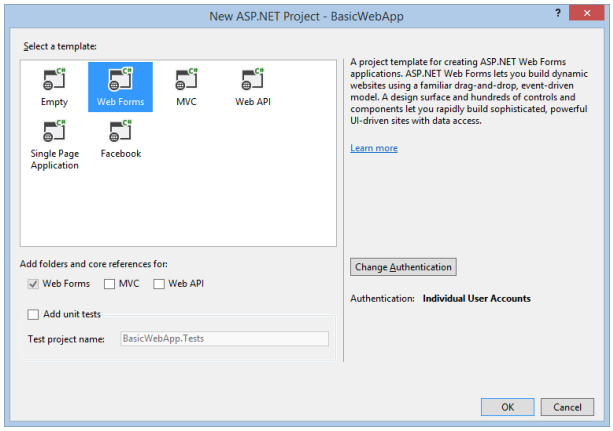


The **New Project** dialog box appears.

3. Select the **Templates** -> **Visual C#** -> **Web** templates group on the left.
4. Choose the **ASP.NET Web Application** template in the center column.
5. Name your project **BasicWebApp** and click the **OK** button.



6. Next, select the **Web Forms** template and click the **OK** button to create the project.

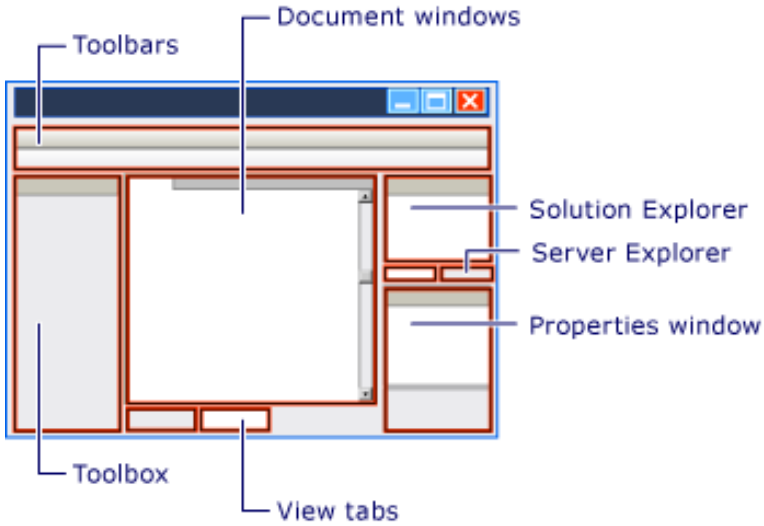


Visual Studio creates a new project that includes prebuilt functionality based on the Web Forms template. It not only provides you with a Home.aspx page, an About.aspx page, a Contact.aspx page, but also includes membership functionality that registers users and saves their credentials so that they can log in to your website. When a new page is created, by default Visual Studio displays the page in **Source** view, where you can see the page's HTML elements. The following illustration shows what you would see in **Source** view if you created a new Web page named BasicWebApp.aspx.

A Tour of the Visual Studio Web Development Environment:

Before you proceed by modifying the page, it is useful to familiarize yourself with the Visual Studio development environment. The following illustration shows you the windows and tools that are available in Visual Studio and Visual Studio Express for Web.

The Visual Studio environment:



Familiarize yourself with the Web designer:

Examine the above illustration and match the text to the following list, which describes the most commonly used windows and tools. (Not all windows and tools that you see are listed here, only those marked in the preceding illustration.)

- Toolbars. Provide commands for formatting text, finding text, and so on. Some toolbars are available only when you are working in **Design** view.
- **Solution Explorer** window. Displays the files and folders in your Web application.
- Document window. Displays the documents you are working on in tabbed windows. You can switch between documents by clicking tabs.
- **Properties** window. Allows you to change settings for the page, HTML elements, controls, and other objects.
- View tabs. Present you with different views of the same document. Design view is a near-WYSIWYG editing surface. **Source** view is the HTML editor for the page. **Split** view displays both the **Design** view and the **Source** view for the document. You will work with the **Design** and **Source** views later in this walkthrough. If you prefer to open Web pages in **Design** view, on the **Tools** menu, click **Options**, select the **HTML Designer** node, and change the **Start Pages In** option.
- **ToolBox**. Provides controls and HTML elements that you can drag onto your page. **Toolbox** elements are grouped by common function.
- **Server Explorer**. Displays database connections. If Server Explorer is not visible, on the View menu, click Server Explorer.

2.2 SUMMARY

Creating a new ASP.NET Web Forms Page:

When you create a new Web Forms application using the **ASP.NET Web Application** project template, Visual Studio adds an ASP.NET page (Web Forms page) named Default.aspx, as well as several other files and folders. You can use the Default.aspx page as the home page for your Web application. However, for this walkthrough, you will create and work with a new page.

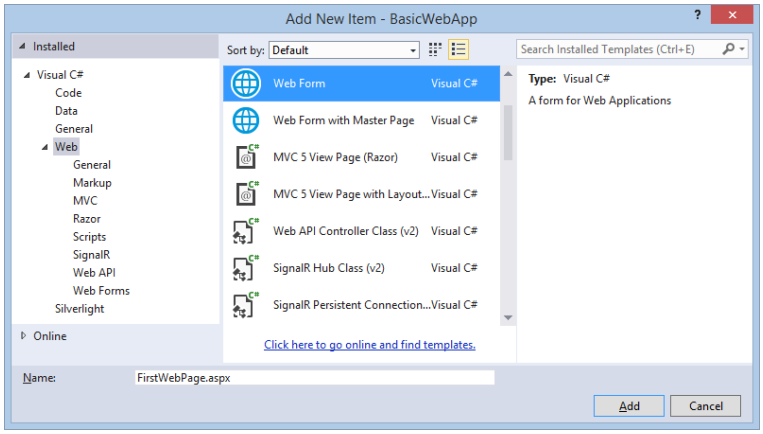
To add a page to the Web application:

1. Close the Default.aspx page. To do this, click the tab that displays the file name and then click the close option.

2. In **Solution Explorer**, right-click the Web application name (in this tutorial the application name is **BasicWebSite**), and then click **Add -> New Item**.

   The Add New Item dialog box is displayed.

3. Select the **Visual C# -> Web** templates group on the left. Then, select **Web Form** from the middle list and name it FirstWebPage.aspx.



4. Click **Add** to add the web page to your project.

   Visual Studio creates the new page and opens it.

Adding HTML to the Page:

In this part of the walkthrough, you will add some static text to the page.

To add text to the page:

1. At the bottom of the document window, click the **Design** tab to switch to **Design** view.

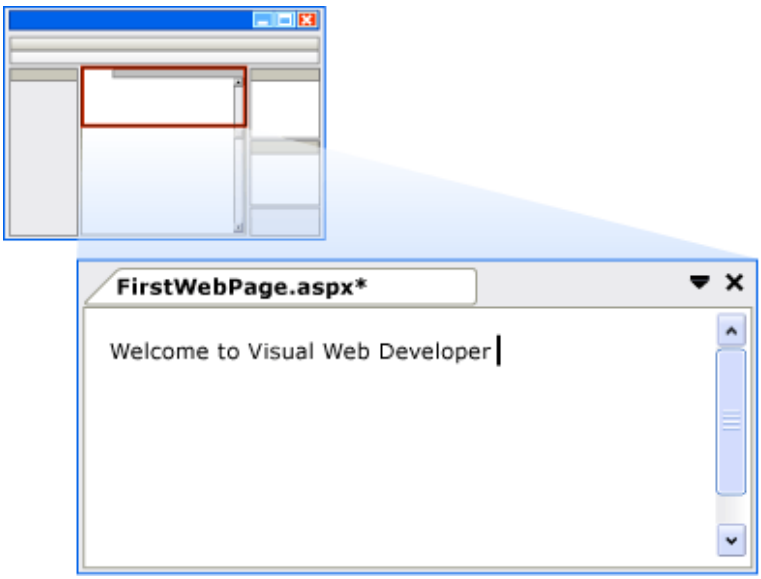
   Design view displays the current page in a WYSIWYG-like way. At this point, you do not have any text or controls on the page, so the

page is blank except for a dashed line that outlines a rectangle. This rectangle represents a **div** element on the page.

2. Click inside the rectangle that is outlined by a dashed line.

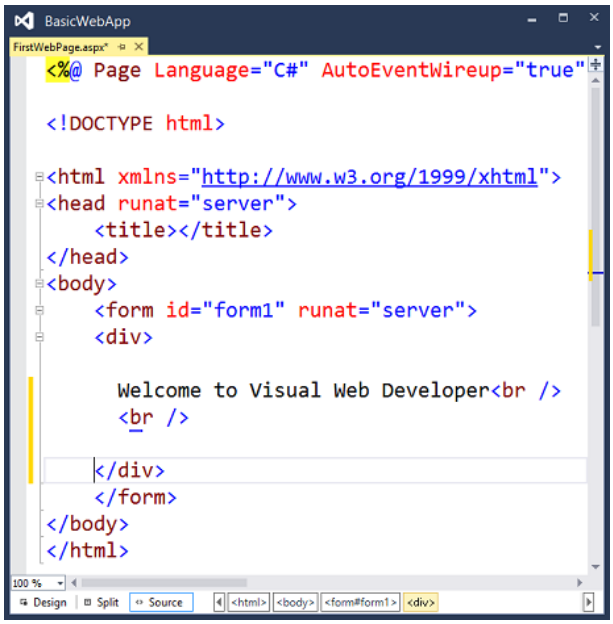
3. Type Welcome to **Visual Web Developer** and press **ENTER** twice.

   The following illustration shows the text you typed in **Design** view.



4. Switch to **Source** view.

   You can see the HTML in **Source** view that you created when you typed in **Design** view.



Running the Page:

Before you proceed by adding controls to the page, you can first run it.

To run the page:

1. In **Solution Explorer**, right-click FirstWebPage.aspx and select **Set as Start Page**.

2. Press **CTRL+F5** to run the page.

   The page is displayed in the browser. Although the page you created has a file-name extension of .aspx, it currently runs like any HTML page.

   To display a page in the browser you can also right-click the page in **Solution Explorer** and select **View in Browser**.

3. Close the browser to stop the Web application.

Adding and Programming Controls:

You will now add server controls to the page. Server controls, such as buttons, labels, text boxes, and other familiar controls, provide typical form-processing capabilities for your Web Forms pages. However, you can program the controls with code that runs on the server, rather than the client.

You will add a Button control, a TextBox control, and a Label control to the page and write code to handle the Click event for the Button control.

To add controls to the page:

1. Click the **Design** tab to switch to **Design** view.

2. Put the insertion point at the end of the Welcome to **Visual Web Developer** text and press **ENTER** five or more times to make some room in the **div** element box.

3. In the **Toolbox**, expand the **Standard** group if it is not already expanded.

   Note that you may need to expand the **Toolbox** window on the left to view it.

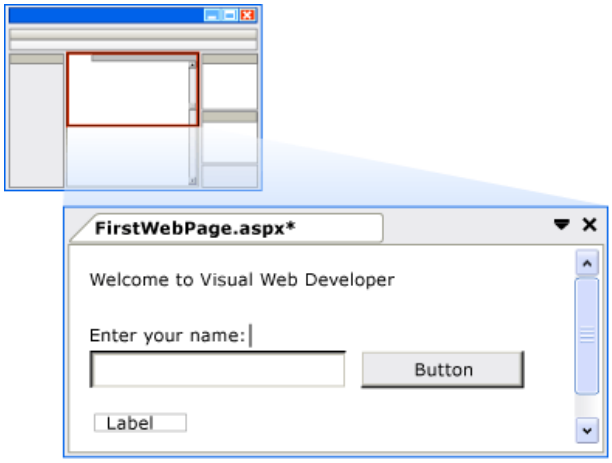
4. Drag a TextBox control onto the page and drop it in the middle of the **div** element box that has Welcome to **Visual Web Developer** in the first line.

5. Drag a Button control onto the page and drop it to the right of the TextBox control.

6. Drag a Label control onto the page and drop it on a separate line below the Button control.

7. Put the insertion point above the TextBox control, and then type **Enter your name:**

This static HTML text is the caption for the TextBox control. You can mix static HTML and server controls on the same page. The following illustration shows how the three controls appear in **Design** view.



Setting Control Properties:

Visual Studio offers you various ways to set the properties of controls on the page. In this part of the walkthrough, you will set properties in both **Design** view and **Source** view.

To set control properties:

1. First, display the **Properties** windows by selecting from the **View** menu -> **Other Windows** -> **Properties Window**. You could alternatively select **F4** to display the Properties window.

2. Select the Button control, and then in the **Properties** window, set the value of **Text to Display Name**. The text you entered appears on the button in the designer, as shown in the following illustration.

3. Switch to **Source** view.

   Source view displays the HTML for the page, including the elements that Visual Studio has created for the server controls. Controls are declared using HTML-like syntax, except that the tags use the prefix **asp:** and include the attribute **runat="server"**.

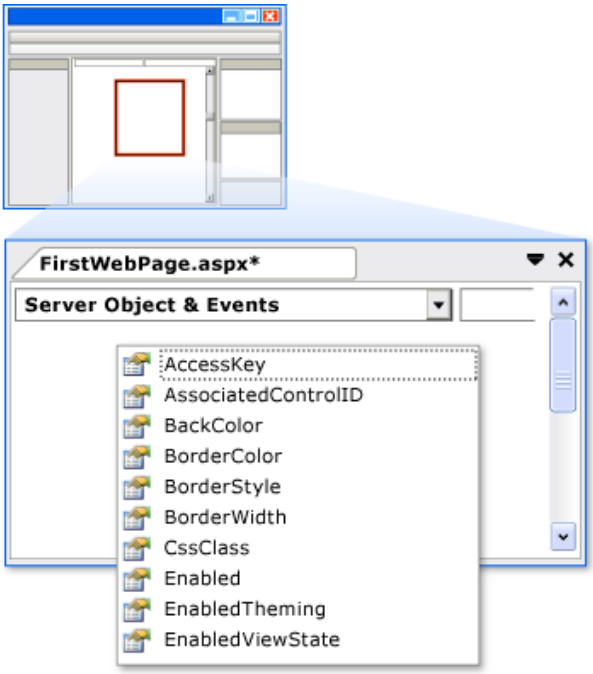
   Control properties are declared as attributes. For example, when you set the **Text** property for the Button control, in step 1, you were actually setting the Text attribute in the control's markup.

   Note: All the controls are inside a **form** element, which also has the attribute **runat="server"**. The **runat="server"** attribute and the **asp:** prefix for control tags mark the controls so that they are processed by

ASP.NET on the server when the page runs. Code outside of **<form runat="server">** and **<script runat="server">** elements is sent unchanged to the browser, which is why the ASP.NET code must be inside an element whose opening tag contains the **runat="server"** attribute.

4. Next, you will add an additional property to the Label control. Put the insertion point directly after **asp:Label** in the **<asp:Label>** tag, and then press **SPACEBAR**.

   A drop-down list appears that displays the list of available properties you can set for a Label control. This feature, referred to as **IntelliSense**, helps you in **Source** view with the syntax of server controls, HTML elements, and other items on the page. The following illustration shows the **IntelliSense** drop-down list for the Label control.



5. Select **ForeColor** and then type an equal sign.

   IntelliSense displays a list of colors.

Note:

You can display an **IntelliSense** drop-down list at any time by pressing **CTRL+J** when viewing code.

6. Select a color for the **Label** control's text. Make sure you select a color that is dark enough to read against a white background.

   The **ForeColor** attribute is completed with the color that you have selected, including the closing quotation mark.

Programming the Button Control:

For this walkthrough, you will write code that reads the name that the user enters into the text box and then displays the name in the Label control.

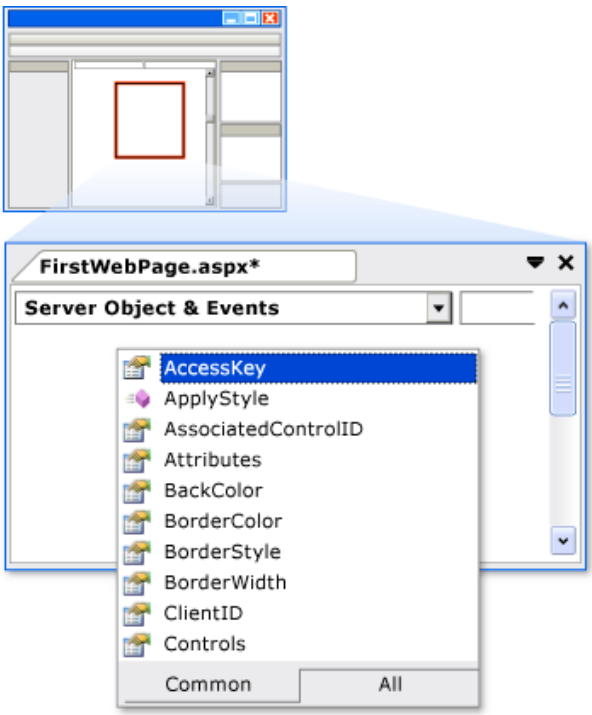
Add a default button event handler:

1. Switch to **Design** view.
2. Double-click the Button control.

   By default, Visual Studio switches to a code-behind file and creates a skeleton event handler for the Button control's default event, the Click event. The code-behind file separates your UI markup (such as HTML) from your server code (such as C#).

   The cursor is positioned to added code for this event handler.
3. Inside the **Button1_Click** event handler, type **Label1** followed by a period (**.**).

   When you type the period after the **ID** of the label **(Label1)**, Visual Studio displays a list of available members for the Label control, as shown in the following illustration. A member commonly a property, method, or event.



4. Finish the **Click** event handler for the button so that it reads as shown in the following code example.

   C#Copy

   protected void Button1_Click(object sender, System.EventArgs e)

   {

```
Label1.Text = TextBox1.Text + ", welcome to Visual Studio!";
}
```

VBCopy

```
Protected Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
Label1.Text = Textbox1.Text & ", welcome to Visual Studio!"
End Sub
```

5. Switch back to viewing the **Source** view of your HTML markup by right-clicking FirstWebPage.aspx in the **Solution Explorer** and selecting **View Markup**.

6. Scroll to the **<asp:Button>** element. Note that the **<asp:Button>** element now has the attribute **onclick="Button1_Click"**.

   This attribute binds the button's Click event to the handler method you coded in the previous step.

   Event handler methods can have any name; the name you see is the default name created by Visual Studio. The important point is that the name used for the **OnClick** attribute in the HTML must match the name of a method defined in the code-behind.

Running the Page:

You can now test the server controls on the page.

To run the page:

1. Press **CTRL+F5** to run the page in the browser. If an error occurs, recheck the steps above.

2. Enter a name into the text box and click the **Display Name** button.

   The name you entered is displayed in the Label control. Note that when you click the button, the page is posted to the Web server. ASP.NET then recreates the page, runs your code (in this case, the Button control's Click event handler runs), and then sends the new page to the browser. If you watch the status bar in the browser, you can see that the page is making a round trip to the Web server each time you click the button.

3. In the browser, view the source of the page you are running by right-clicking on the page and selecting **View source**.

   In the page source code, you see HTML without any server code. Specifically, you do not see the **<asp:>** elements that you were working with in **Source** view. When the page runs, ASP.NET processes the server controls and renders HTML elements to the page that perform the functions that represent the control. For example, the

<asp:Button> control is rendered as the HTML **<input type="submit">** element.

4. Close the browser.

2.3 UNIT END EXERCISE

- Write a program to add a Calendar control which displays dates and month at a time.

2.4 LIST OF REFERENCES

1] Practical ASP.NET Web API

2] ASP.NET Core in Action by Andrew Lock

3] Building RESTful Web Services with .NET Core by Gaurav Aroraa, Tadit Dash

*****

3

VARIABLES, DATA TYPES, OPERATORS AND OBJECTS IN ASP.NET

Unit Structure

3.0 OBJECTIVES

A web page whose main purpose is to display regular text and images to the browser is suited for HTML tags only. To make the page more interesting, you and your users can exchange information, as illustrated in the previous lesson. For example, you may want your users to submit values to the server, let the server either store those values or analyze them and send some results to the user. This interact usually means that, in the beginning, you would not know the type of value(s) that one visitor would submit as opposed to another user. Based of this, you must prepare storage areas on the server's computer memory to temporarily hold the values used during a visitor's interaction with your web page. This type of storage area is called a variable.

As there are different types of values used in an application, there are also different requirements to store those values. Fortunately, to store a value in a memory, you must provide only two pieces of information: the name of the variable and the type of information that would be stored in the memory reserved for that variable.

3.1 INTRODUCTION

The Name of a Variable:

To specify the name of a variable, there are rules you must follow:

The name of a variable must start with either a letter or an underscore. A name can have only one letter but if you start it with an underscore, then the underscore must be followed by a letter

After the first character as an underscore or a letter, the name can contain letters, digits, or underscores in any combination

The name must not contain space but it can be made of various words as long as these words are concatenated (added together to produce one word)

The name must not contain any symbol other than letters, digits, or underscores

After respecting these rules, you can adopt a naming convention that suits you.

Names in C#:

To name the variables of your program, you must follow strict rules. In fact, everything else in your application must have a name. C# uses a series of words, called keywords, for its internal use. This means that you must avoid naming your objects using one of these keywords. They are:

abstract	const	extern	int	out	short	typeof
As	continue	false	interface	override	sizeof	Uint
base	decimal	finally	internal	params	stackalloc	ulong
bool	default	fixed	is	private	static	unchecked
break	delegate	float	lock	protected	string	unsafe
byte	do	for	long	public	struct	ushort
case	double	foreach	namespace	readonly	switch	using
catch	else	goto	new	ref	this	virtual
char	enum	if	null	return	throw	Void
checked	event	implicit	object	sbyte	true	volatile
class	explicit	in	operator	sealed	try	while

Once you avoid these words, there are rules you must follow when naming your objects. On this site, here are the rules we will follow:

The name must start with a letter or an underscore

After the first letter or underscore, the name can have letters, digits, and/or underscores

The name must not have any special characters other than the underscore

The name cannot have a space

Besides these rules, you can also create your own but that abide by the above. C# is case-sensitive. This means that the names Case, case, and CASE are completely different.

3.2 SUMMARY

Data Types:

Â

Characters

Escape Sequence	**Name**	**Description**
\a	Bell (alert)	Makes a sound from the computer
\b	Backspace	Takes the cursor back
\t	Horizontal Tab	Takes the cursor to the next tab stop

\n	New line	Takes the cursor to the beginning of the next line
\v	Vertical Tab	Performs a vertical tab
\f	Form feed	Â
\r	Carriage return	Causes a carriage return
\"	Double Quote	Displays a quotation mark (")
\'	Apostrophe	Displays an apostrophe (')
\?	Question mark	Displays a question mark
\\	Backslash	Displays a backslash (\)
\0	Null	Displays a null character

In the English alphabet, a character is one of the following symbols: a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, and Z. Besides these readable characters, the following symbols are called digits and they are used to represent numbers: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. In addition, some symbols on the (US QWERTY) keyboard are also called characters or symbols. They are ` ~ ! @ # $ % ^ & * ( ) - _ = + [{] } \ | ; : ' < ? . / , > "

Besides the English language, other languages use other or additional characters that represent verbal or written expressions.

C# recognizes that everything that can be displayed as a symbol is called a character. To declare a variable whose value would be a character, use the char keyword. Here is an example:

```
char Gender = 'M';
```

Escape Sequences:

An escape sequence is a special character that displays non-visibly. For example, you can use this type of character to indicate the end of line, that is, to ask the program to continue on the next line. An escape sequence is represented by a backslash character, \, followed by another character or symbol. For example, the escape sequence that moves to the next line is \n.

An escape can be included in single-quotes as in '\n'. It can also be provided in double-quotes as "\n".

The C# language recognizes other escape sequences.

Â To use an escape sequence, you can also first declare a char variable and initialize it with the desired escape sequence in single-quotes.

The Byte Data Type:

A byte is a number whose value can range from 0 to 255 and therefore can be stored in one byte. You can use it when you know a variable would hold a relatively small value such as people's age, the number of children of one mother, etc. To declare a variable that would hold a small natural number, use the byte keyword. Here is an example:

byte Age;

You can initialize a byte variable when declaring it or afterwards. Here is an example that uses the byte data type:

Byte Age = 14;

Make sure you do not use a value that is higher than 255 for a byte variable, you would receive an error. When in doubt, or when you think that there is a possibility a variable would hold a bigger value, don't use the byte data type as it doesn't like exceeding the 255 value limit.

Signed Byte:

A byte number is referred to as signed if it can hold a negative of a positive value that ranges from -128 to 127, which can therefore fit in a byte. To declare a variable for that kind of value, use the sbyte keyword. Here is an example:

sbyte RoomTemperature = -88;

Short Integers:

A natural number is also called an integer. To use a variable that would hold such a number, you can declare it using the short keyword followed by a name. A variable declared as short can hold a value that ranges from â€“32768 to 32767. Here are two examples:

short NumberOfPages;

short Temperature;

NumberOfPages = 842;

TemperatureÂ Â = -1544;

Because a short integer handles numbers that are larger than the signed byte, any variable you can declare for a signed byte can also be declared for a short variable.

Unsigned Short Integers:

If a variable must hold positive and relatively small numbers, it is referred as an unsigned short integer. Such a variable can be declared using the ushort keyword. An unsigned short integer can hold numbers that range from 0 to 65535 and therefore can fit in 16 bits. Here is an example:

// These variables must hold only positive integers

ushort NumberOfTracks;

ushort MusicCategory;

NumberOfTracks = 16;

MusicCategoryÂ = 2;

To use a variable that would hold quite large numbers, declare it using the int keyword. A variable declared as int can store values between â€“2,147,483,648 and 2,147,484,647 negative or positive. Here is an example:

int CoordX;

int CoordY;

Â Â Â

CoordX = 12;

CoordY = -8;

Unsigned Integers:

If the variable must hold only positive natural numbers, you can declared it using the uint keyword. The uint keyword is used for a variable that can hold positive numbers whose value would range from 0 to 2,147,484,647. Here are examples:

uint DayOfBirth;

uint MonthOfBirth;

uint YearOfBirth;

Â Â Â

DayOfBirthÂ Â = 8;

MonthOfBirth = 11;

YearOfBirthÂ = 1996;

Long Integers:

To use a variable that can hold very large numbers, declare it using the long keyword.

Here is an example that uses the long data type:

long CountryArea;

Â Â Â Â Â Â

CountryArea = 5638648;

Although the long data type is used for large number, it mainly indicates the amount of space available but you don't have to use the whole space. For example, you can use the long keyword to declare a variable that would hold the same range of numbers as the short, the int, or the uint data

types. If you declare a variable as long but use it for small numbers, the compiler would allocate the appropriate amount of space to accommodate the values of the variable. Consequently, the amount of space made available may not be as large as required. If you insist and want the compiler to reserve the whole space required for a long variable, when assigning a value to the variable, add an L suffix to it. Here is an example:

long CountryArea;

Â Â Â Â Â Â

CountryArea = 5638648L;

Keep in mind that an int, a uint, a short, or a ushort can fit in a long variable.

Unsigned Long Integers:

In some cases, you will need a variable to hold only positive, though large, numbers. To declare such a variable, you can use the ulong data type. A variable declared as ulong can handle very large positive numbers that range from 0 to 18,446,744,073,709,551,615.

Introduction to Real Numbers:

A real number is a number that displays a decimal part. This means that the number can be made of two sections separated by a symbol that is referred to as the Decimal Separator or Decimal Symbol. This symbol is different by language, country, group of languages, or group of countries. In US English, this symbol is the period as can be verified from the Regional (and Language) Settings of the Control Panel. On both sides of the Decimal Symbol, digits are used to specify the value of the number. The number of digits on the right side of the symbol determines how much precision the number offers.

Floating-Point Numbers With Single Precision:

The integers we have used so far have the main limitation of not allowing decimal values. C# provides floating values that would solve this problem. The most fundamental floating variable is declared with the float keyword. A variable declared a float can store real numbers that range from Â±1.5 Ã— 10â^'45 to Â±3.4 Ã— 1038 with a precision of 7 digits in 32 bits. Here is an example:

float Distance;

Floating-Point Numbers With Double Precision:

When a variable is larger than the float can handle and requires more precision, you should declare it using the double keyword. A variable declared as double can store very large numbers ranging from Â±5.0 Ã— 10â^'324 to Â±1.7 Ã— 10308 with a precision of 15 or 16 digits.

Here is an example:

double StudentAge;

Â Â Â Â Â Â

StudentAge = 14.50;

Because the double data type provides a better result with a better precision than the float, whenever you declare a variable as float and assign it a value as we did earlier, the compiler reserves very large memory space that can store the values of the variable. If you insist on the variable being treated as float, when assigning it a value, add an F suffix to the value. Here is an example:

float Distance;

Â Â Â Â Â Â

Distance = 248.38F;

Decimal:

The decimal data type can be used to declare a variable that would hold significantly large values. You declare such a variable using the decimal keyword. The values stored in a decimal variable can range from Â±1.0 Ã— 10â^’28 to Â±7.9 Ã— 1028 with a precision of 28 to 29 digits. Because of this high level of precision, the decimal data type is suitable for currency values.

After declaring a decimal variable, you can initialize it with a natural number. To indicate that the variable holds a decimal value, when initializing it, add an M suffix to its value. Here is an example:

decimal HourlySalary;

Â Â Â Â Â Â

HourlySalary = 24;

Accessory Data Types

Â

Strings:

A string is an empty space, a character, a word, or a group of words that you want the compiler to consider "as is", that is, not to pay too much attention to what the string is made of, unless you explicitly ask it to. This means that, in the strict sense, you can put in a string anything you want.

Primarily, the value of a string starts with a double quote and ends with a double-quote. An example of a string is "Welcome to ASP.NET development".

Sometimes, you will need to use a string whose value is not known in advance. Therefore, you can first declare a string variable. To do this, use the string keyword followed by a name for the variable. The name will follow the rules we defined above. An example of a string declaration is:

string Msg;

After declaring a string, you can give it a primary value by assigning it an empty space, a character, a symbol, a word, or a group of words. The value given to a string must be included in double-quotes. Here are examples of string variables declared and initialized:

string EmptyÂ = "";

string Gender = "F";

string FNameÂ = "Nelson Mandela";

string MsgÂ Â Â = "Welcome to the World of C# Programming! ";

Dates and Times:

A date is a unit that measures the number of years, months, or days elapsed in a specific period. A time is a unit that counts the number of seconds that have elapsed since midnight of the day considered. Although dates and times are large subjects that would require a detailed study, at this time, we will consider them in simple terms.

To declare a variable that would hold date or time values, use the DateTime data type. Here is an example:

DateTime DateHired;

The .NET Framework sets its starting periodic date to January 1, 0001 at midnight (12:00:00 or 0:00 AM). If not assigned a specific value, the variable is initialized to 1/1/0001 at midnight.

OBJECTS:

The object data type is used to declare a variable whose type is not primarily defined and can be any of the other data types we have introduced. Here is an example:

object Whatever;

After creating an object variable, you can use its value as you see fit.

Constants

Custom Constants:

Suppose you intend to use a number such as 39.37 over and over again. If you use this 39.37 many times in your program, at one time, you may make a mistake and type it as 3937 or 3.937 or else. Because of mistakes in the way to represent the number, the same calculation produces different results. To make sure that this is unlikely, you can instead use a variable that holds the value. Then, when you need that value, you can access the variable instead of the value itself. A number such as 39.37 is called a constant.

A constant is a value that never changes such as 244, "ASEC Mimosa", 39.37, or True. These are constant values you can use in your program any time. You can also declare a variable and make it a constant; that is, use it so that its value is always the same.

To create a constant, type the const keyword to its left. When declaring a constant, you must initialize it with an appropriate value. Here is an example:

```
const double ConversionFactor = 39.37;
```

Once a constant has been created and it has been appropriately initialized, you can use its name where the desired constant would be used. To initialize a constant variable, the value on the right side of the assignment operator "=" must be a constant or a value that the compiler can determine as constant. Instead of using a known constant, you can also assign it another variable that has already been declared as constant.

Built-in Constants:

There are two main categories of constants you will use in your programs. You can create your own constant as we saw above. The C# language also provides various constants. Some constants are part of the C# language. Some other constants are part of the .NET Framework. Before using a constant, of course, you must first know that it exists. Second, you must know how to access it. A constant that is part of the C# language can be accessed anywhere in your code. Those constant are normally defined in the System namespace. Other constant are defined in various appropriate namespaces.

null: The null keyword is a constant used to indicate that a variable doesn't hold a known value

PI: PI is a constant used as the ratio of the circumference of a circle to its diameter. PI is defined in Math. To use it, you would type Math.PI.

3.3 UNIT END EXERCISE

- Write a code to show how to declare a variable, assign a value to it using an Assign<T> activity, and then display its value to the console using a WriteLine activity.

3.4 LIST OF REFERENCE

1] Practical ASP.NET Web API

2] ASP.NET Core in Action by Andrew Lock

3] Building RESTful Web Services with .NET Core by Gaurav Aroraa, Tadit Dash

*****

4

BASIC SERVER-SIDE CONTROL

Unit Structure

4.0 OBJECTIVES

Controls are small building blocks of the graphical user interface, which include text boxes, buttons, check boxes, list boxes, labels, and numerous other tools. Using these tools, the users can enter data, make selections and indicate their preferences.

Controls are also used for structural jobs, like validation, data access, security, creating master pages, and data manipulation.

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

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

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

- **Validation controls:** These are used to validate user input and they work by running client-side script.
- **Data source controls:** These controls provides data binding to different data sources.
- **Data view controls:** These are various lists and tables, which can bind to data from data sources for displaying.
- **Personalization controls:** These are used for personalization of a page according to the user preferences, based on user information.
- **Login and security controls:** These controls provide user authentication.

- **Master pages:** These controls provide consistent layout and interface throughout the application.
- **Navigation controls:** These controls help in navigation. For example, menus, tree view etc.
- **Rich controls:** These controls implement special features. For example, AdRotator, FileUpload, and Calendar control.

The syntax for using server controls is:

```
<asp:controlType  ID ="ControlID" runat="server"
Property1=value1  [Property2=va
```

The Microsoft.NET Framework provides a rich set of server-side controls for developing Web applications. You can add these controls to WebForms pages just as you add Windows controls to a form. Server-side controls are often called server controls or Web Forms controls. There are four types of Server controls: HTML server controls. Web server controls, validation control, and user controls.

4.1 INTRODUCTION

HTML Server controls:

HTML developers must be familiar with old HTML controls, which they use to write GUI applications in HTML. These controls are the same HTML controls; you can run these controls on the server by defining the runat ="server" attribute. These control names start with Html. Below table defines some of these controls.

Table: HTML server Controls

CONTROL	DESCRIPTION
HtmlForm	Create an HTML form control, used as a place holder of other controls
HtmlInputText	Creates an input text box control used to get input from user
HtmltextArea	Creates multiline text box control
HtmlAnchor	Creates a Web navigation
HtmlButton	Creates a button control
HtmlImage	Creates an image control, which is used to display an image
HtmlInputCheckBox	Creates a check box control
HtmlInputRadioButton	Creates a radio button control
HtmlTable	Creates a table control
HtmlTableRow	Creates a row within a table
HtmlTableCell	Creates a cell with in a row

Web Server Controls:

Web server controls are more powerful than HTML controls because they provide more functionality and are easier to use. Besides some of the basic controls such as button, text box, label, and checkbox, ASP.NET provides some more powerful controls such as DataGrid, DataList, and Calendar. I'll use these controls throughout this article. Below table describes some of these controls.

Table: Web Server controls:

CONTROL	DESCRIPTION
Label	Represents a label control
ListBox	Represents a list box control
CheckBox	Represents a Check box control
Calendar	Represents a calendar control
ImageButton	Represents an image button control
TableCell	Represents a table cell
Panel	Represents a panel control
DataList	Represents a data list control
TextBox	Represents a text box control
Image	Represents an image control
CheckBoxList	Represents a list box with check boxes
Button	Represents a button control
HyperLink	Represents a hyperlink control
TableRow	Represents a row of a table
RadioButtonList	Represents a list box with radio button controls
DataGrid	Represents a data grid control
DropDownList	Represents a drop-down list control
AdRotator	Represents an ad rotator control
RadioButton	Represents a radio button control
LinkButton	Represents a link button control
Table	Represents a table control
Repeater	Represents a repeater control

Validation Controls:

Validating user input is one of the important needs for Web applications. These controls provide features to validate user input. Using these controls, you can check as required field, a value, a range, a pattern of characters, and so on below table describes validation controls.

Table: Validation Controls

CONTROL	DESCRIPTION
RequiredFieldValidator	Makes sure that the user doesn't skip an entry
CompareValidator	Compares user input with a value using a comparison operator such as less than, greater than, and so on

RangeValidator	Checks if the user's input falls with in a certain range
RegularExpressionValidator	Checks if the user's input matches a defined pattern
CustomValidator	Create your own customer logic

User Controls:

Besides HTML serve controls, web server controls, and validation controls, you can also create your own controls by embedding Web Forms controls. These controls are called custom controls. You create custom controls when the available controls can't provide the functionality you need. For example, if you want to create a data grid control with check boxes, combo boxes, calendars, and date controls, you can create a custom control derived from the available controls and the then write the additional functionality.

Server Controls and the .NET Framework Library:

The .NET Framework library provides the System.Web and its 15 supporting namespaces to define Web classes. These namespace reside in the System.web.dll assembly. Before you use any Web namespaces, though, you need to add a reference to the System.web.dll assembly and include the required namespace in the application. Some major namespace of the Web series are System.Web, System.Web.UI, System.Web.UI.HtmlControls, System.Web.UI.WebControl, and System.Web.Services.

The System.Web Namespace:

The System.Web namespace contains browser-and server-related classes and interfaces. For example, the HTTPRequest and HTTPResponse classes provide functionality to make requests for HTTP to retrieve and post data on the server through a browser. The HttpApplication class defines the functionality of an ASP.NET application. This namespace also contains the HttpCookie and HttpCookieCollection classes for manipulating cookies. The HttpFileCollection class provides access to and organizes file uploaded by client. You can use the HttpWriter class to write to the server through HttpResponse.

The System.Web.UI Namespace:

The System.Web.UI namespace contains classes and interfaces that enable you to develop Web-based GUI applications similar to Windows GUI applications. This namespace provide classes to create Web Forms pages and controls. The control is the mother of all Web control classes and provides methods and properties for HTML, web or user controls. The page class represents Web page requested by the server in an ASP.NET application. It also has classes for data binding with the data-bound controls such as DataGrid and DataList. You'll see these classes in the

examples in this article. In addition to these classes, it also includes state management, templates, and validation-related classes.

The System.Web.UI.HtmlControls Namespace:

This namespace contains HTML control classes, which I've discussed in the "HTML Server Controls" section of this article. Some of these namespace classes are HtmlButton, HtmlControl, HtmlForm, HtmlImage, HtmlInputText, HtmlTable, and so on.

The System.Web.UI.WebControls Namespace:

This namespace contains classes related to server controls and their supporting classes, as discussed in the "Web Server Controls" section of this article. Some of the classes are AddRotator, Button, Calendar, CheckBox, DataGrid, DataList, DropDownList, Hyperlink, Image, Label, ListBox, ListControl, Panel, Table, TableRow, and TextBox. Besides control classes, it also contains control helper classes. For example, the DataGridItem, DataGridColumn, and TableRow, TableCell, TableCellCollection, TableHeaderCell, and TableItemsStyle are helper classes of the table control.

The System.Web.Services Namespace:

A web Service is an application that sits and runs on the Web server. System.Web.Service and its three helper namespaces System.Web.Service.Description, System.Web.Services.Discovery, and System.Web.Service.Protocol – provides classes to build Web services.

Why are Web Forms Controls called server-side controls?:

Microsoft .NET Framework consists powerful Web controls. By using these Web controls you write powerful Web GUI applications similar to desktop applications. You can either write code for these controls manually or by using VS.NET, which supports the drag-and-drop design-time feature. In other worlds, you can drag and drop Web Forms controls onto a Web form, set properties by right-clicking on a control, and even write handlers by double-clicking on the control as you'd do in windows GUI applications such as Visual Basic.

When a client (Web browser) makes a call for Web control such as a Button or a DataGrid, The runat ="Server" (Discussed later in more detail) tells the Web server that the controls will be executes on the server and they'll send HTML data to the client at run-time after execution. Because the execution of these control events, methods, and attributes happens on the server, these controls are server-side Web controls. The main functionality of these controls includes rendering data from the server to the client and event handling. (The controls fire events and handle those events.)

Adding server side controls to a Web Form:

You have two ways to add server controls to a Web Form (also referred as a web page). You can either use the VS .NET IDE to add server Controls or you can add controls manually by typing code using the <asp:> syntax.

Adding server control using VS .NET:

Adding server controls using VS.NET is pretty simple. As you have seen in the "Developing your first ASP .NET Web Application" section of this article, you create a new ASP.NET Application project, open the toolbox, drag and drop controls from the toolbox, set properties, and write event handlers for the control.

Adding Server Controls Using ASP .NET Syntax:

The other method of adding server controls to an application is that you write the code manually. VS.NET writes the code in the background foe you when you drop a control from the toolbox to Web form.

To add server controls manually, you create a text file and save it with an .aspx extension .NET utilizes XML tags to write server controls. A tag should a low XML syntax. Every ASP.NET control starts with asp: and a control name. For example, the following line a text box control:

<asp: textbox id=TextBox1 runat = "Server" Text =" " text =" "> </asp:textbox>

In this line, I created a text box server control. Event control has unique ID. In this sample the ID is TextBox1. The runat ="Server" attribute represents that the control will run on the server.

The following code shows that you can write the same code without the closing tag:

<asp:textbox id = Textbox1 runat ="server" />

Below shows the ASP.NET version of your first ASP.NET application . You can see the ASP.NET version using the HTML mode of the designer.

As you can see below, asp:Button, asp:TextBox, and asp:ListBox are three ever controls added using ASP.NET. In listing 7-4, you'll see some unfamiliar items such as <%Page, Language, and codebehind. The <%Page language is a page directive, which defines the language you're using in the page you can use any .NET supported language, such as C# or VB.NET. You use the codebehind directive to separate the code from the ASP.NET page. You can define a C# or VB.NET page as codebehind, which will host the code foe ASP.NET controls for the page. WebForm1.aspx.cs is a C# class, which hosts code for the WebForm1 page.

ASP.NET version of my first ASP.NET application:

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeBehind="Default.aspx.cs" Inherits="firstaspnet._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:Button ID="Button1" runat="server" Text="Button"
BackColor="#3366FF" Font-Bold="True"
         ForeColor="Yellow" /><br />
      <br />
      <asp:TextBox ID="TextBox1" runat="server"
BorderColor="#66CCFF" BorderStyle="Groove"
         Height="32px" Width="198px"></asp:TextBox>
      <br />
      <asp:ListBox ID="ListBox1" runat="server" BackColor="#CCFF99"
Height="209px" Width="310px">
      </asp:ListBox>
      <font size="5"><strong>My First ASP.NET
Application</strong></font>
      <p>
         <font><strong>Click Add button to add contents of text box to the
list box</strong></font>
      </p>
    </div>
    </form>
</body>
</html>
```

4.2 SUMMARY

The HTML server controls are HTML elements that include a runat=server attribute. The HTML server controls have the same HTML output and the same properties as their corresponding HTML tags. In addition, HTML server controls provide automatic state management and server-side events. HTML server controls offer the following advantages:

- The HTML server controls map one to one with their corresponding HTML tags.
- When the ASP.NET application is compiled, the HTML server controls with the runat=server attribute are compiled into the assembly.

- Most controls include an OnServerEvent for the most commonly used event for the control. For example, the <input type=button> control has an OnServerClick event.
- The HTML tags that are not implemented as specific HTML server controls can still be used on the server side; however, they are added to the assembly as HtmlGenericControl.
- When the ASP.NET page is reposted, the HTML server controls keep their values.

The System.Web.UI.HtmlControls.HtmlControl base class contains all of the common properties. HTML server controls derive from this class.

To use an HTML server control, use the following syntax (which uses the HtmlInputText control as an example):

ASP.NET (C#)Copy

```
<input type="text" value="hello world" runat=server />
```

Web server controls:

Web controls are similar to the HTML server controls such as Button, TextBox, and Hyperlink, except that Web controls have a standardized set of property names. Web server controls offer the following advantages:

- Make it easier for manufacturers and developers to build tools or applications that automatically generate the user interface.
- Simplify the process of creating interactive Web forms, which requires less knowledge of how HTML controls work and make the task of using them less prone to errors.

The System.Web.UI.WebControls.WebControl base class contains all of the common properties. Most of the Web server controls derive from this class.

To use a Web server control, use the following syntax (which uses the TextBox control as an example):

ASP.NET (C#)Copy

```
<asp:textbox text="hello world" runat=server />
```

Web server controls can be divided into four categories:

- Basic Web Controls
- Validation Controls
- List Controls
- Rich Controls

Basic web controls:

Basic Web controls provide the same functionality as their HTML server control counterparts. However, basic Web controls include additional methods, events, and properties against which you can program.

Validation controls

Validation controls are used to validate the values that are entered into other controls of the page. Validation controls perform client-side validation, server-side validation, or both, depending on the capabilities of the browser in which the page is displayed. Validation controls offer the following advantages:

- You can associate one or more validation controls with each control that you want to validate.
- The validation is performed when the page form is submitted.
- You can specify programmatically whether validation should occur, which is useful if you want to provide a cancel button so that the user can exit without having to fill valid data in all of the fields.
- The validation controls automatically detect whether validation should be performed on the client side or the server side.

List controls

List controls are special Web server controls that support binding to collections. You can use list controls to display rows of data in a customized, template's format. All list controls expose DataSource and DataMember properties, which are used to bind to collections.

List controls can bind only to collections that support the IEnumerable, ICollection, or IListSource interfaces. For example, a Visual C# .NET sample page appears as follows:

ASP.NET (C#)Copy

```
<%@ Page Language="C#" %>
<script runat="server">
   Public void Page_Load()
   {
      String[] myStringArray = new String[] {"one","two","three"};
      rptr.DataSource = myStringArray;
      rptr.DataBind();
   }
</script>
<html>
```

```
<body>
  <asp:repeater id=rptr runat="server">
    <itemtemplate><%# Container.DataItem %><br></itemtemplate>
  </asp:repeater>
</body>
</html>
```

A Visual Basic .NET sample page appears as follows:

ASP.NET (VB)Copy

```
<%@ Page Language="vb" %>
<script runat="server">
  public sub Page_Load()
    Dim myStringArray as String()
    myStringArray = new String() {"one","two","three"}
    rptr.DataSource = myStringArray
    rptr.DataBind()
  end sub
</script>
<html>
  <body>
    <asp:repeater id=rptr runat="server">
      <itemtemplate><%# Container.DataItem %><br></itemtemplate>
    </asp:repeater>
  </body>
</html>
```

The output appears as follows:

- one
- two
- three

4.3 UNIT END EXERCISE

- Write a program containing the following controls:
  - A ListBox
  - A Button
  - An Image
  - A Label

- The listbox is used to list items available in a store. When the user clicks on an item in the listbox, its image is displayed in the image control. When the user clicks the button, the cost of the selected item is displayed in the control.

4.4 LIST OF REFERENCE

1] Practical ASP.NET Web API

2] ASP.NET Core in Action by Andrew Lock

3] Building RESTful Web Services with .NET Core by Gaurav Aroraa, Tadit Dash

*****

MODULE III

5

DATABASE PROGRAMMING IN ASP.NET

Unit Structure

Self-Learning Topics: Charts and Data Pagers

5.0 OBJECTIVES

What You Will Learn in This Chapter:

- Learn the basic classes in ADO.NET and its architecture.
- Learn the different ADO.NET Data Providers.
- Learn about the Connection and Command components.
- Learn about the Working with Stored Procedures.
- Learn about the DataAdapter, DataReader and DataSet components.

- Learn LINQ with ASP.NET.
- Discuss about the query syntax in LINQ.
- Learn the Entity Framework Architecture.

5.1 AN OVERVIEW

- ADO.NET is a set of classes that expose data access services to the Microsoft .NET programmer. ADO.NET provides a rich set of components for creating distributed, data-sharing applications.
- ADO.NET is an integral part of the Microsoft .NET Framework, providing access to relational, XML, and application data.
- All ADO.NET classes are located at the System.Data namespace with two files named System.Data.dll and System.Xml.dll. When compiling code that uses the System.Data namespace, reference both System.Data.dll and System.Xml.dll.
- Basically speaking, ADO.NET provides a set of classes to support you to develop database applications and enable you to connect to a data source to retrieve, manipulate and update data with your database.

5.2 INTRODUCTION OF THE ADO.NET

ADO.NET is a collection of managed libraries used by .NET applications for data source communication using a driver or provider:

Enterprise applications handle a large amount of data. This data is primarily stored in relational databases, such as Oracle, SQL Server, and Access and so on. These databases use Structured Query Language (SQL) for retrieval of data.

To access enterprise data from a .NET application, an interface was needed. This interface acts as a bridge between an RDBMS system and a .NET application. ADO.NET is such an interface that is created to connect .NET applications to RDBMS systems.

In the .NET framework, Microsoft introduced a new version of Active X Data Objects (ADO) called ADO.NET. Any .NET application, either Windows based or web based, can interact with the database using a rich set of classes of the ADO.NET library. Data can be accessed from any database using connected or disconnected architecture.

ADO.NET provides mainly the following two types of architectures:

- **ConnectedArchitecture:**

The ADO.NET Connected architecture considers mainly three types of objects:

- Connection con

- Command cmd
- DataReaderdr

- **DisconnectedArchitecture:**

The ADO.NET Disconnected architecture considers primarily the following types of objects:

- DataSet ds
- DataAdapter da
- Connection con

5.3 ARCHITECTURE OF ADO.NET

Ado.net is a data access technology that allows interaction between applications and databases.

Types of Architecture:

- Connected Architecture
- Disconnected Architecture

Ado.net is both connection-oriented as well as disconnection oriented. Depending upon the functionality of an application, we can make it connection-oriented or disconnection oriented. We can even use both the modes together in a single application.

5.3.1 Connected Architecture:

- As the name suggests, connected architecture refers to the fact that the connection is established for the full time between the database and application. For e.g. we make a program in C# that is connected with the database for the full time, so that will be connected architecture.
- Connected architecture is forward only and read-only. This means the connected mode will work only in one particular direction i.e. forward and that too for read-only purpose. Application issues query then read back results and process them.
- For connected architecture, we mainly use the object of the DataReader class.
- DataReader is used to retrieve the data from the database and it also ensures that the connection is maintained for the complete interval of time.
- In connected architecture, the application is directly linked with the Database.

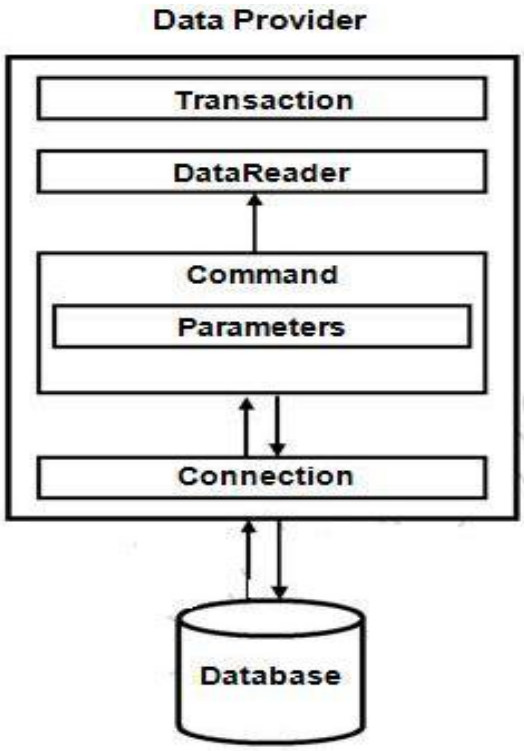


Fig 5.1 Connected Architecture in ADO.NET

5.3.2 Disconnected Architecture:

- Disconnected architecture refers to the mode of architecture in Ado.net where the connectivity between the database and application is not maintained for the full time. Connectivity within this mode is established only to read the data from the database and finally to update the data within the database.

- This means during the processing of the application, we need data so that data is fetched from the database and kept in temporary tables. After that whenever data is required, it is fetched from the temporary tables. And finally, when the operations were completed, the connection was established to update the data within the database from the temporary tables.

- In this mode, application issues query then retrieves and store results for processing. For this purpose, we use objects of DataAdapter and DataSet classes.

- In disconnected architecture, a Dataset is used for retrieving data from the database. This way there is no need to establish a connection for the full time because DataSet acts as temporary storage.

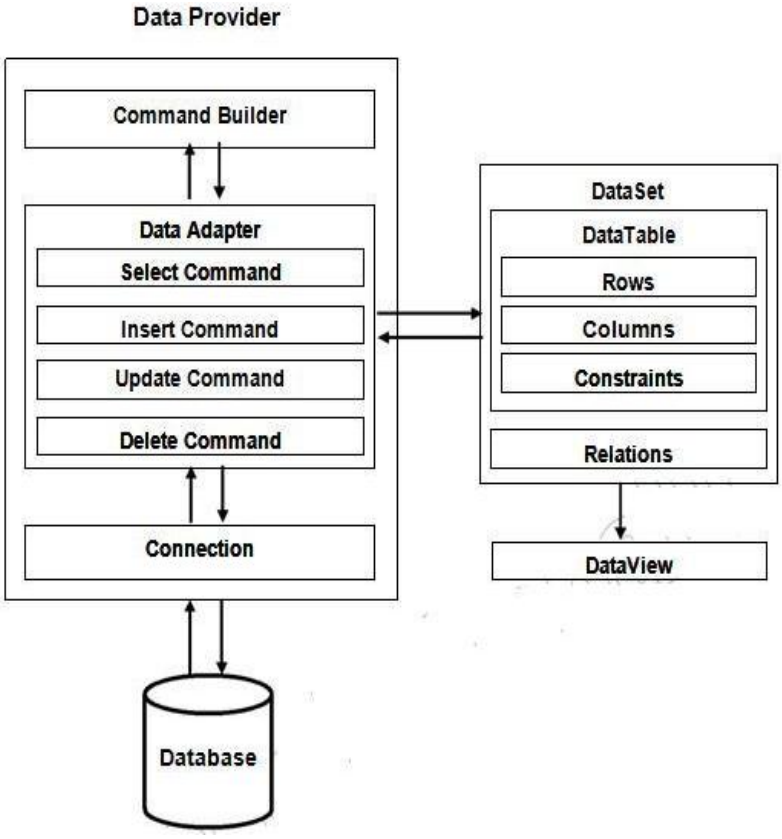


Fig 5.2 Disconnected Architecture in ADO.NET

Difference between Connected and Disconnected:

Connected	Disconnected
It is Connection Oriented	It is Dis-Connection Oriented
Connected get high in speed and performance.	Disconnected get low in speed and performance.
Connected you need to use a read only forward only data reader	Disconnected you cannot
Data Reader can't persist the data	Data Set can persist the data
It is Read only, we can't update the data	We can update data

5.4 COMPONENTS OF ADO.NET

Components are designed for data manipulation and faster data access. **Connection, Command, DataReader, DataAdapter, DataSet,** and **DataView** are the components of ADO.NET that are used to perform database operations. ADO.NET has two main components that are used for accessing and manipulating data. They are as follows:

1. **Data provider and**
2. **DataSet**

5.4.1 Data Provider:

The Data Provider can also be called a data driver and it can be used as a major component for your data-driven applications.The functionalities of the Data Provider are to:

1. **Connect to the Database**
2. **Prepare an SQL Command**
3. **Execute the Command**
4. **Retrieve the results and display them in the application**

The Data Provider is physically composed of a binary library file and this library is in the DLL file format. Sometimes this DLL file depends on other DLL files, so in fact a Data Provider can be made up of several DLL files. Based on the different kinds of databases, Data Provider can have several versions and each version is matched to each kind of database.

There are following different types of data providers included in ADO.Net

Data Provider	Description
Data Provider for SQL Server (SQL Server.NET)	It provides data access for Microsoft SQL Server. The **System.Data.SqlClient** namespace is used
Data Provider for OLEDB (OLEDB.NET)	It is used for data sources using OLE DB. The **System.Data.OleDb** namespace is used
Data Provider for ODBC (ODBC.NET)	It is used for data sources using ODBC. The **System.Data.Odbc** namespace is used
Data Provider for Oracle (Oracle.NET)	It is used Oracle data sources. The **System.Data.OracleClient** namespace is used

The different data providers are located at the different namespaces, and these namespaces hold the various data classes that you must import into your code in order to use those classes in your project.

Table contains the most popular namespaces used by the different data providers and used by the DataSet.

DataProvider Properties and Methods:

Objects	Description
Connection	This component is used to set up a connection with a data source.
Command	A command is a SQL statement or a stored procedure used to retrieve, insert, delete or modify data in a data source.
DataReader	Data reader is used to retrieve data from a data source in a read-only and forward-only mode
DataAdapter	This is integral to the working of ADO.Net since data is transferred

Objects	Description
	to and from a database through a data adapter. It retrieves data from a database into a dataset and updates the database. When changes are made to the dataset, the changes in the database are actually done by the data adapter.

Advantages of using Data Providers in ADO.NET:

1) It provides high performance of client-server access over RDBMS
2) It contains trigger providing powerful means for maintaining business rules at database level
3) It provides referential integrity that supports primary/foreign key definition
4) It provides data access through table or server side control
5) It is easy to convert the .NET framework to other database engines
6) It contains full server based transaction to eliminate database corruption
7) It has database security functionality and encryption support

5.4.2 Connection:

Data Provider contains four sub-classes and the Connection component is one of them. This class provides a connection between your applications and the database you selected to connect to your project. To use this class to setup a connection between your application and the desired database, you need first to create an **instance** or an **object** based on this class.

The **Connection** object you want to use depends on the type of the data source you selected. Data Provider provides four different Connection classes and each one is matched to one different database.

These popular Connection classes used for the different data sources:

The Connection classes and databases

Connection Class	Associated Database
OdbcConnection	ODBC Data Source
OleDbConnection	OLE DB Database
SqlConnection	SQL Server Database
OracleConnection	Oracle Database

The **connection string** is a property of the **Connection** class and it provides all necessary information to connect to your data source. Regularly this connection string contains a quite few parameters to define a connection, but only five of them are popularly utilized for most data-driven applications:

- **Provider**
- **Data Source**
- **Database**
- **User ID**
- **Password**

A typical data connection instance with a general connection string can be expressed by the following codes:

Connection = New xxxConnection("Provider = **MyProvider**;" & _

"Data Source = **MyServer**;" & _

"Database = **MyDatabase**;" & _

"User ID = **MyUserID**;" & _

"Password = **MyPassWord**;")

wherexxx should be replaced by the selected Data Provider in your real application, such as **OleDb, Sql** or **Oracle**. You need to use the real parameter values implemented in your applications to replace those nominal values such as **MyServer, MyDatabase, MyUserID** and **MyPassWord** in your application.

Some typical Connection instances are listed below:

OLE DB Data Provider for Microsoft Access Database:

```
Connection = New
OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" & _
        "Data Source=C:\database\CSE_DEPT.mdb;" & _
        "User ID=MyUserID;" & _
        "Password=MyPassWord;")
```

SQL Server Data Provider for SQL Server Database:

```
Connection = New SqlConnection("Server=localhost;" + _
        "Data Source=Susan\SQLEXPRESS;" + _
 "Database=CSE_DEPT;" + _
        "Integrated Security=SSPI")
```

Oracle Data Provider for Oracle Database:

```
Connection = New OracleConnection("Data Source=XE;" + _
                                  "User ID=system;" + _
                                "Password=reback")
```

Example for SQL Server Database:

```
using System;
using System.Data;
using System.Data.SqlClient;
```

```
public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
    }
    string conString = "Data Source=(local);Initial
Catalog=Employee;Integrated Security=True";

    protected void Button1_Click(object sender, EventArgs e)
    {
        // Creating the object of SqlConnection class
        SqlConnectionconObject = new SqlConnection(conString);

        try
        {
            conObject.Open();
            if (conObject.State == ConnectionState.Open)
                Response.Write("Database Connection is Open");
        }
        catch(SqlExceptionsqlexception)
        {
            Response.Write("ERROR ::" + sqlexception.Message);
        }
        catch(Exception ex)
        {
            Response.Write("ERROR ::"+ex.Message);
        }
        finally
        {
            conObject.Close();
        }
    }
}
```

In connection string, we can write or Server or Address or Network Address in place of Data Source attribute. To specify the database we can use Database or the Initial Catalog attribute.

You can provide yes, no, true, SSPI and false value to Integrated Security attribute of connection string. If Integrated Security = false then User ID, and Password must be specified in the connection string and if true then current Windows account credentials are used for authentication. SSPI is equivalent to value True.

Connection String Parameter Name	Description
Data Source	Identify the server.Local Machine/Machine Domain Name/Ip Address
Initial Catalog	Database Name
Integrated Security	Set to SSIP to make Connection with user's

Connection String Parameter Name	Description
	Window Login
UserId	Name of user configured in SQL Server
Password	Password match Sql Server UserId

Some of the properties for the connection object are as shown below:

Property	Description
CanRaiseEvents	It gets a value indicating the component can raise an event
ConnectionString	It gets or sets string used to open the connection
ConnectionTimeout	It gets the time to wait for establishing the connection
Database	It gets the name of the current database after the connection is opened
DataSource	It gets the name of the database server to which the user can connect
Site	It gets or sets the ISite of the component
ServerVersion	It gets a string representing the version of the server
State	It gets a string that describes the state of the connection

Some of the methods for the DbConnection object are as shown below:

Methods	Description
BeginDbTransaction	It starts the database transaction
ChangeDatabase	It changes the current database for an open connection
Close	It closes the connection to the database
CreateCommand	It creates and returns the DbCommand object
Dispose()	It releases the resources used by the component
GetHashCode	It serves as a default hash function
GetSchema()	It returns the schema information for the data source

Open	It opens the database connection with the settings specified by the ConnectionString object
ToString	It returns the string containing the name of the component

Open() and Close() are two important methods of Connection object.

Program:

```
using System;
using System.Data;
using System.Data.SqlClient;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            string conString = "Data Source=(local);Initial
Catalog=Employee;Integrated Security=True";
            SqlConnectionconObject = new SqlConnection(conString);
            conObject.Open();
            Console.WriteLine("Connection String ::
"+conObject.ConnectionString);
            Console.WriteLine("DataBase :: "+conObject.Database);
            Console.WriteLine("DataSource :: " + conObject.DataSource);
            Console.WriteLine("ConnectionTimeout :: " +
conObject.ConnectionTimeout);
            Console.WriteLine("Connection State :: " + conObject.State);
            Console.WriteLine("ServerVersion :: " +
conObject.ServerVersion);
            Console.ReadLine();
        }
    }
}
```

Output:

```
Connection String :: Data Source=(local);Initial
Catalog=Employee;Integrated Security=True
DataBase :: Employee
DataSource :: (local)
ConnectionTimeout :: 15
```

Connection State :: Open
ServerVersion :: 10.50.1600

Storing connection string in configuration file

If you did not write connection string in config file, then it may create problem. Suppose that, after deploying the project, you have to change the connection string, then you have to change the connection string in every page.

It is a good practice to store the connection string for your application in a config file rather than in every page.

In this scenario, if you have written the connection string in config file, then the required change will be done in single place (config file) and the change will be reflect automatically in every page.

```
<configuration>
  <connectionStrings>
    <add name="conString" connectionString="Data Source=(local);Initial Catalog=Employee;Integrated Security=True"/>
  </connectionStrings>
</configuration>
```

Acceccing the connecting string from config file

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Configuration;
protected void Button1_Click(object sender, EventArgs e)
{
    string conn = ConfigurationManager.ConnectionStrings["conString"].ConnectionString;
    // Creating the object of SqlConnection class
    SqlConnection  conObject = new SqlConnection(conn);
    conObject.Open();
    if (conObject.State == ConnectionState.Open)
        Response.Write("Database Connection is Open");
}
```

By default, a connection string is enabled with connection pooling.

By default, the maximum number of pools is 100, minimum is 0.

Connection pooling is the ability of re-use your connection to the Database. This means if you enable Connection pooling in the connection object, actually you enable the re-use of the connection to more than one user.

5.4.2.1 Connection Pooling:

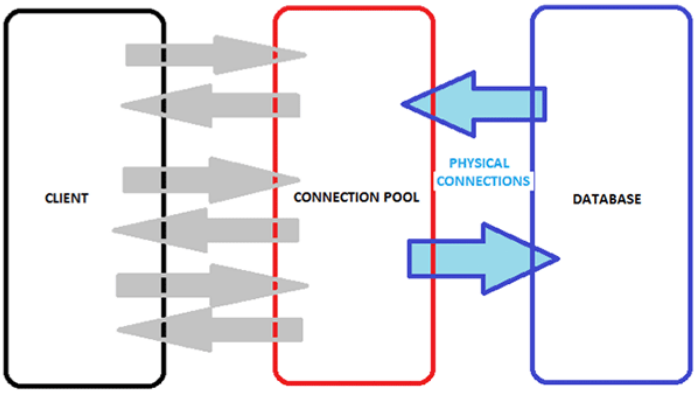


Fig. 5.3 Connection Pooling

Connection pooling enables you to re-use your connection to the Database. It is an optimization technique that will increase the performance of database programming. By default connection pooling feature is enabled with connection string. The maximum number of pools is 100, and minimum is zero.

Connecting to database is time consuming and resource intensive task. Whenever user wants to connect database, connection string is parsed and it checks whether given credentials in connection stings are correct or not. Therefore it takes lots of time to open and close the connection.

To reduce the cost of opening and closing the same connection repeatedly and increasing the performance ADO.NET uses connection pooling.

A Connection String in the Web.Config file with connection pooling option:

```
<connectionStrings>
<clear/>
<add name="sqlConnectionString" connectionString="Data
Source=mySQLServer;Initial Catalog=myDatabase;Integrated
Security=True;Connection Timeout=15;Connection Lifetime=0;Min Pool
Size=0;Max Pool Size=100;Pooling=true;"/>
</connectionStrings>
```

SQL Server connection string pooling attributes:

- **Connection Lifetime:**
  - Length of time in seconds after creation after which a connection is destroyed. The default is 0, indicating that connection will have the maximum timeout.

- **Connection Reset:**
  - Specifies whether the connection is reset when removed from the pool. The default is true.
- **Enlist:**
  - Specifies whether the connection is automatically enlisted in the current transaction context of the creation thread if that transaction context exists. The default is true.
- **Load Balance Timeout:**
  - Length of time in seconds that a connection can remain idle in a connection pool before being removed.
- **Max Pool Size:**
  - Maximum number of connections allowed in the pool. The default is 100.
- **Min Pool Size:**
  - Minimum number of connections maintained in the pool. The default is 0.
- **Pooling:**
  - When true, the connection is drawn from the appropriate pool, or if necessary, created and added to the appropriate pool. The default is true.

5.4.3 Command:

The Command object works with the Connection object and used to execute SQL queries or Stored Procedures against the data source. You can perform insert, update, delete, and select operations with this object. It requires an instance of a Connection Object for executing the SQL statements as example.

SqlCommandcmd = new SqlCommand("SQL query or stored procedure", conn);

The SQL statements can be SELECT, INSERT, UPDATE, or DELETE. It uses a connection object to perform these actions on the database.

SELECT

```
cmd =new SqlCommand("select * from Employee", con);
cmd =new OracleCommand("select * from Employee", con);
```

INSERT

```
cmd = new SqlCommand("INSERT INTO Employee(Emp_ID,Emp_Name)VALUES ('" + aa + "','" + bb +"')",con);
cmd = new OracleCommand("INSERT INTO Employee(Emp_ID , Emp_Name) VALUES('" + aa + "','" + bb + "' )",con);
```

UPDATE

```
cmd =new SqlCommand("UPDATE Employee SET Emp_ID ='" + aa + "' , Emp_Name ='" + bb + "' WHERE Emp_ID ='" + aa + "'", con);
cmd =new OracleCommand("UPDATE Employee SET Emp_ID ='" + aa + "' , Emp_Name ='" + bb + "' WHERE Emp_ID ='" + aa + "'", con);
```

DELETE

```
cmd =new SqlCommand("DELETE FROM Employee where Emp_ID='" + aa + "'", con);
cmd =new OracleCommand("DELETE FROM Employee where Emp_ID='" + aa + "'", con);
```

Steps for executing SQL query with command object:

- Create a Connection Object and initialize with connection string.
- Create the command object and initialize with SQL query and connection object.
- Open the connection.
- Execute query with the help of any one method of command object.
- Store the result in DataReader object (In case of select SQL query)
- Close the connection.

Some of the Command properties are as listed below:

Property	Description
CommandTimeout	Contains the length of the timeout of a query, in seconds
CommandText	It gets or sets the text command against the data source
CommandType	**CommandType** is an important property of Command class. This property is used to determine which type of command being executed. **CommandType.StoredProcedure:** It informs the Command object that the stored procedure

	will be used in place of simple SQL statements. **CommandType.Text:** It informs the Command object that the CommandText value contains a SQL query. **CommandType.TableDirect:** It indicates that the CommandText property contains the name of a table
Connection	It gets or sets the DbConnection used by the DbCommand
Parameters	It gets the collection of DbParameter objects
Site	It gets or sets the ISite of the component
Transaction	It gets or sets the DbTransaction which the DbCommand object executes
UpdatedRowSource	It gets or sets the command results applied to the DataRow

Some of the methods for the Command objects are as shown below:

Methods	Description
Cancel()	It attempts to cancel the execution of the DbCommand
CreateDbParameter	It creates a new instance of DbParameter object
Dispose()	It releases all the resources used by the component
ExecuteDbDataReader	It executes the command text against the connection
ExecuteNonQuery	It executes SQL statement against the connection object If you are using Insert, Update or Delete SQL statement then use this method. Its return type is Integer (The number of affected records).

ExecuteReader()	It executes the CommandText against the Connection object This method works on select SQL query. It returns the DataReader object. Use DataReader read () method to retrieve the rows.
ExecuteScalar()	Executes the CommandText property and returns data in a DataReader object. This method returns single value. Its return type is Object. When you want single value (First column of first row), then use ExecuteScalar () method. Extra columns or rows are ignored. ExecuteScalar method gives better performance if SQL statements have aggregate function.
GetService	It returns an object representing a service object for the instance
ExecuteXMLReader	It returns an instance of XmlReader class. This method is used to return the result set in the form of an XML document.

Let's take one example that will use command object. The table structure is as follows.

Program:

```
using System;
using System.Data;
using System.Data.SqlClient;
public partial class _Default : System.Web.UI.Page
{
   SqlDataReader reader;
   SqlConnection con;
   protected void Page_Load(object sender, EventArgs e)
   {
      con = new SqlConnection("Data Source=(local);Initial
Catalog=Employee;Integrated Security=True");
      SqlCommandcmd = new SqlCommand();
      cmd.Connection = con;
      cmd.CommandText = "select * from tblemps";
      cmd.CommandType = CommandType.Text;
      try
```

```
        {
            con.Open();
            reader = cmd.ExecuteReader();
            GridView1.DataSource = reader;
            GridView1.DataBind();
        }
        catch (Exception ex)
        {
            Label1.Text = "ERROR :: " + ex.Message;
        }
        finally
        {
            reader.Close();
            con.Close();
        }
    }
}
```

5.4.3.1 Working With Stored Procedures With Command Object:

First we understand that what is stored procedure and then we will use the stored procedure in our database program.

Stored Procedure:

A stored procedure in SQL Server is a group of one or more SQL statements. Stored procedure are created and stored in the database.

When you execute the application, the SQL statements are parsed and execution plan is created. This happened every time whenever you execute the application.

It is not happened in case of stored procedure. Stored procedure compiles only once and stores in database with its execution plan. If we use stored procedure, then we can avoid recompilation of the query. It increases the performance of application. Another advantage is that it can be tested independent of the application. Stored procedures are easy to maintain. If any changes occur, then just update the stored procedure. The change will reflect in all pages automatically. It is a big advantage that rather than change SQL queries in all the pages just we need to update a single stored procedure.

Creating Stored Procedure:

Open Microsoft SQL Server, select your database, right click and select create new stored procedure tab.

Here we have created a simple stored procedure named "**GetAllEmployees**". It selects all the record from table tblEmps.

```
CREATE PROCEDURE [dbo].[GetAllEmployees]
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
SET NOCOUNT ON;
    -- Insert statements for procedure here
SELECT * from tblEmps
END
```

Display data using stored procedure:

Just change the previous code as follows and you will get the same result.

```
SqlConnection  con = new SqlConnection("Data Source=(local);Initial
Catalog=Employee;Integrated Security=True");
SqlCommandcmd = new SqlCommand();
cmd.Connection = con;
cmd.CommandType = CommandType.StoredProcedure;
cmd.CommandText = "GetAllEmployees";
```

Insert Record using stored procedure:

First create a simple input screen to insert the data into database as given below.

Create a Stored Procedure with Input Parameter in the database.

```
CREATE PROCEDURE InsertEmployees
    @EmpID integer,
    @EmpNamevarchar(100),
    @Gender varchar(50),
    @Salary money,
    @Address varchar(200),
    @DepID integer
AS
BEGIN
    -- Insert statements for procedure here
    insert into tblEmps values(@EmpID,@EmpName ,@Gender
,@Salary ,@Address ,@DepID )
END
GO
```

Write down your connection string in web.config file.

```
<configuration>
   <connectionStrings>
      <add name="conString" connectionString="Data
Source=(local);Initial Catalog=Employee;Integrated Security=True"/>
   </connectionStrings>
</configuration>
```

```
using System;
using System.Configuration;
using System.Data;
using System.Data.SqlClient;
public partial class Default2 : System.Web.UI.Page
{
   SqlConnectionconObject;
   SqlCommandcmd;
   protected void Page_Load(object sender, EventArgs e)
   {
   }
   protected void btnAdd_Click(object sender, EventArgs e)
   {
      string con
=ConfigurationManager.ConnectionStrings["conString"].ConnectionStrin
g;
      conObject = new SqlConnection(con);
      cmd = new SqlCommand();
      cmd.Connection = conObject;
      cmd.CommandType = CommandType.StoredProcedure;
      cmd.CommandText = "InsertEmployees";

cmd.Parameters.Add("@EmpID",SqlDbType.Int).Value=txtEmpID.Text;

cmd.Parameters.Add("@EmpName",SqlDbType.VarChar).Value=txtNam
e.Text;
      cmd.Parameters.Add("@Gender",
SqlDbType.VarChar).Value=txtGender.Text;
      cmd.Parameters.Add("@Salary",
SqlDbType.Money).Value=txtSalary.Text;
      cmd.Parameters.Add("@Address",
SqlDbType.VarChar).Value=txtAddress.Text;
```

```
        cmd.Parameters.Add("@DepID",
SqlDbType.Int).Value=txtDepID.Text;
        try
        {
           conObject.Open();
           int i=cmd.ExecuteNonQuery();
           if(i>0)
           {
              Label1.Text = "Record Added Successfully !!";
           }
        }
        catch(Exception ex)
        {
           Label1.Text="ERROR ::"+ex.Message;
        }
        finally
        {
           conObject.Close();
        }
    }
}
```

5.4.4 Dataset:

The DataSet class is used for representing a subset of the database. It is defined in the **System.Data** namespace. It is used for disconnected representation of results sets from the Data Source. It provides more flexibility while dealing with the result sets.

DataSet object behaves like a small database. The DataSet contains rows, columns, primary keys, constraints, and relation with Data Table objects. It has a collection of DataTable objects and can be related to each other with Data Relation objects.

It can contain more than one table and also create the relationship between tables. Again DataTable is the collection of DataRow and DataColomn objects.

One of the advantages of dataset is that it can work with different types of databases simultaneously. Suppose that one table is available within SQL server and another table is available in Oracle. You can easily query each database server and create a single DataSet with both the table.

The list of properties for DataSet are as shown below:

Property	Description
Container	It gets the container for the component
CaseSensitive	It gets or sets value indicating whether string comparisons with Data Table objects
DataSetName	It gets or sets the name of the current DataSet
DefaultViewManager	It gets a custom view of the data contained into the DataSet to allow filtering, searching and navigating data using a custom DataViewManager
DesignMode	It gets a value indicating the component is currently in design mode
Events	It gets a list of event handlers that are attached to this component
ExtendedProperties	It gets a collection of customized user information associated with Data Set
HasErrors	It gets a value indicating the errors in any DataTable objects within the DataSet
Tables	It gets the collection of tables contained in the DataSet
Relations	It gets a collection of relation that links tables and allow navigation from parent tables to child tables
IsInitialized	It gets a value that indicates whether the DataSet is initialized

The list of methods of DataSet are as shown below:

Methods	Description
AcceptChanges	It commits all the changes made in the DataSet when it was loaded or when the last changes were accepted

BeginInit	It begins the initialization of a DataSet used on a form or by other component
Clear	It clears the DataSet of any data by removing all rows in the table
Clone	It copies the structure of the DataSet including Data Table schemas, relations and constraints
Copy	It copies the structure and data from the DataSet
CreateDataReader()	It returns the DataTableReader with one result set per Data Table
Dispose()	It release all the resources used by the MarshalByValueComponent
EndInit	It ends the initialization od a DataSet that is used on a form
Finalize	It allows the object to try to free resources and perform cleanup operations before the garbage collection
GetChanges()	It gets a copy of the DataSet containing all changes made in the last load
GetDataSetSchema	It gets a copy of XmlSchemaSet for the DataSet
GetHashCode	It serves as a default hash function
HasChanges()	It gets a value indicating whether DataSet has changes, including new, deleted or modified rows
InferXmlSchema(String, String[])	It applies the XML Schema from the specified Stream into DataSer
InferXmlSchema(TextReader, String[])	It applies the XML Schema from the specified XmlReader to the DataSet
IsBinarySerialized	It inspects the format of the serialized representation of the DataSet
Load(IDataReader, LoadOption, String[])	It fills a DataSet with values from a data source using supplied IDataReader using array of strings
MemberwiseClone	It creates a shallow copy of the current Object

Merge(DataRow[])	It merges an array of DataRow objects into the current DataSet
Merge(DataSet, Boolean)	It merges a DataSet and its schema into current DataSet
Merge(DataTable, Boolean, MissingSchemaAction)	It merges a DataTable and its schema into current DataSet preserving or removing changes into the DataSet
OnRemoveRelation	It occurs when a DataRelation object is removed from the DataTable
ReadXml(Stream)	It reads XML Schema and data into DataSet using System.IO.Stream
Reset	It clears all tables and removes all relations, contraints, and tables from the DataSet
ShouldSerializeRelations	It gets a value indicating the Relations property should be persisted
ToString	It returns a string containing the name of the component
WriteXml(Stream)	It writes the current data for the DataSet using the System.IO.Stream
WriteXml(TextWriter)	It writes the current data for the DataSet using the TextWriter
WriteXml(TextWriter, XmlWriteMode)	It writes a current data and schema for the DataSet using the TextWriter and XmlWrite mode
WriteXmlSchema(XmlWriter)	It writes the DataSet structure as an XMl schema to the XmlWriter object

The DataSet can be used along with the Data Adapter in an application. The syntax for the creation of DataSet and using in an application is as shown below:

```
DataSet ds = new DataSet();
SqlDataAdapter da = new SqlDataAdapter(sql,connection);
da.Fill(ds);
```

Program:

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Configuration;
```

```
public partial class Default : System.Web.UI.Page
{
   SqlConnection conn;
   SqlDataAdapter adapter;
   DataSet ds;
   protected void Page_Load(object sender, EventArgs e)
   {
      string cs =
ConfigurationManager.ConnectionStrings["conString"].ConnectionString;
      try
      {
         conn = new SqlConnection(cs);
         adapter = new SqlDataAdapter("select * from tblEmps", conn);
         ds = new DataSet();
         adapter.Fill(ds);
         GridView1.DataSource = ds;
         GridView1.DataBind();
      }
      catch(Exception ex)
      {
         Label1.Text = "ERROR :: " + ex.Message;
      }
      finally
      {
         ds.Dispose();
         conn.Dispose();
      }
   }
}
```

5.4.4.1 Datatableand Datarow:

The DataTable class is used when the tables of the database are to be represented. The list of some DataTable properties is as shown below:

Property	Description
CaseSensitive	It indicates that the string comparisons are case – sensitive
ChildRelations	It gets the collection of child relations for this DataTable
Columns	It gets the collection of columns belonging to the table
Constraints	It gets the collection of constraints maintained by the table
DataSet	It gets the DataSet to the corresponding table

DisplayExpression	It gets or sets the expression returning a value to represent the table
Events	It gets the event handlers list attached to the component
Namespace	It gets or sets the namespace for the XML representation of the data store
PrimaryKey	It gets or sets an array of columns functioning as primary key for the table
Rows	It gets the collection of rows belonging to the table
TableName	It gets or sets the name of the DataTable

The list of some of methods of the DataTable are as shown below:

Methods	**Description**
AcceptChanges	It contains all the changes when the last AcceptChanges was called
Clear	It clears the DataTable of all data
Clone	It clones the structure of the DataTable containing constraints and schemas
Copy	It copies the structure and data for the DataTable
Dispose()	It releases all the resources used by the MarshalByValueComponent
EndLoadData	It turns on notifications, constraints, indes maintenance after the data is loaded
Finalize	It allows an object to free resources and perform cleanup operations before garbage collection
GetChanges()	It gets a copy of DataTable where all the changes are made by the user
ImportRow	It copies the DataRow into DataTable
Load(IDataReader)	It fills a DataTable with values from a data source using IDataReader

Merge(DataTable)	It merges the DataTable with the current DataTable
NewRow	It creates a new row with the schema similar to the DataTable
Reset	It resets the DataTable to the original state
Select()	It gets an array of all DataRow objects
WriteXml(String)	It writes the current contents of the DataTable as XML using the specified file

The syntax of creating DataTable in an application is as shown below:

```
DataTabledt = newDataTable();

DataColumn dc = newDataColumn();
dc.ColumnName = “Srno”;
dc.DataType = typeof(int);
dt.Columns.Add(dc);

DataColumn dc1 = newDataColumn();
dc.ColumnName = “Name”;
dc.DataType = typeof(string);
dt.Columns.Add(dc1);

dt.Rows.Add(newobject [ ] { “1”, “John”});
```

Datarow:

The DataRow is used to represent rows in the DataTable. Some of the properties of DataRow are as shown below:

Property	Description
HasErrors	It gets a value indicating that row has errors
Item[DataColumn]	It gets or sets the data stored in the specified DataColumn
Item[String]	It gets or sets the data stored in the column specified by the name
ItemArray	It gets or sets all the values for this row through an array
RowError	It gets or sets the custom error description for a row

Table	It gets the DataTable for DataSet

The syntax for creating DataRow is as shown below:

```
DataRowdr =newdt.NewRow();
dr[“Name”]=”Mark”;
dr[“Course”]=”MBA”;
dt.Rows.Add(dr);
```

5.4.5 Datareader:

DataReader object works in connected mode. DataReader is used in forward only, read only retrieval of data from data sources. They cannot be called directly in the code. It is fast compare to DataSet. DataReader provides the easy way to access the data from database. It can increase the performance of application because it reads records one by one. Use read() method of DataReader to access the record.

- It is used in Connected architecture.
- Provide better performance.
- DataReader Object has Read-only access.
- DataReader Object Supports a single table based on a single SQL query of one database.
- While DataReader Object is Bind to a single control.
- DataReader Object has Faster access to data.
- DataReader Object Must be manually coded.
- We can’t create relation in data reader.
- Data reader communicates with the command object.
- DataReadercan not modify data

For initializing the DataReader object, call the ExecuteReader method of the Command object. It returns an instance of the DataReader.

```
SqlCommandcmd = new SqlCommand("Your SQL query", conn);
SqlDataReaderreaderObj = cmd.ExecuteReader();
```

Once your task completed with the data reader, call the Close() method to close the DataReader.

```
readerObj.Close();
```

DataReader Properties and Methods:

Property	Description
Connection	It gets the Connection associated with DataReader
Depth	Indicates the depth of nesting for row
FieldCount	Returns number of columns in a row
HasRows	It gets a value indicating the DataReader has one or more rows
IsClosed	Indicates whether a data reader is closed
RecordsAffected	Number of row affected after a transaction
Item	Gets the value of a column in native format

Methods	Description
Read()	Reads next record in the data reader.
NextResult()	Advances the data reader to the next result during batch transactions.
Close()	Closes a DataRaeder object.

You can also get the value of particular column of the table by using the data reader object.

```
while(reader.Read())
{
    string ID = reader["EmpID"].ToString();
    string name = reader["Name"].ToString();
    string gender = reader["Gender"].ToString();
    string salary = reader["Salary"].ToString();
}
```

Difference between DataSet and DataReader:

DataSet	DataReader
Works in connected mode.	It provides connection oriented environment.
Provides slow performance compare to DataReader.	Provides the fast execution.
In memory object. You can fetch the data in any order.	It is a forward-only and read only object.
Implicitly open the connection.	It needs explicit open and close the connection.

It can contain more than one table.	At a time, it works on single table.
Dataset objects have XML Support.	It does not fully support XML
It uses fill() method of DataAdapter to populate the dataset.	DataReader object provides the read() method for reading the records.

5.4.6 Dataadapter:

DataAdapter is used as a ADO.NET data provider. It is used in disconnected architecture. It is used as communication between DataSet and DataSource. i.e. It works as a bridge between DataSet and your DataSource. It holds the SQL commands and connection object for reading and writing data. DataSet does not open or close the connection because this task is performed by DataAdapter object. It helps to manage the data in disconnected mode.

DataAdapter performs the following tasks when using with DataSet object:

1. Open connection
2. Fills the DataSet
3. Close connection

It can also perform Select, Insert, Update and Delete SQL operations with the database.

The list of DataAdapter properties is as shown below:

Property	Description
DeleteCommand	It is used for deleting records from the data source
InsertCommand	It is used to insert records in a data source
SelectCommand	It is used to select records from a stored procedure
UpdateCommand	It is used to update records in the data source
TableMappings	It represents the collection of mappings between actual data source table

The list of methods for the DataAdapter is as shown below:

Methods	Description
Fill	It is used to fill data into DataSet using DataAdapter
Update	It is used to update data to the data source
FillSchema	It adds a DataTable to the DataSet

5.5 LINQ WITH ASP.NET

Language integrated query or LINQ enables you to write SQL like syntax in programming language itself. LINQ contains different types of operators, which is used in language itself. In LINQ, the same query can be used in, an SQL database, a DataSet, an array of objects in memory and so many other data types.

5.5.1 LINQ Introduction:

Architecture of LINQ:

In this chapter, we are going to study the **Architecture of LINQ**. The term **LINQ** stands for **Language Integrated Query** and it is pronounced as **LINK**. Nowadays the use of use LINQ increasing rapidly. So, as a developer, you should understand the Linq and its architecture.

What is LINQ?

The LINQ (Language Integrated Query) is a part of a language but not a complete language. It was introduced by Microsoft with .NET Framework 3.5 and C# 3.0 and is available in **System.Linq** namespace.

LINQ provides us common query syntax which allows us to query the data from various data sources. That means using a single query we can get or set the data from various data sources such as SQL Server database, XML documents, ADO.NET Datasets, and any other in-memory objects such as Collections, Generics, etc.

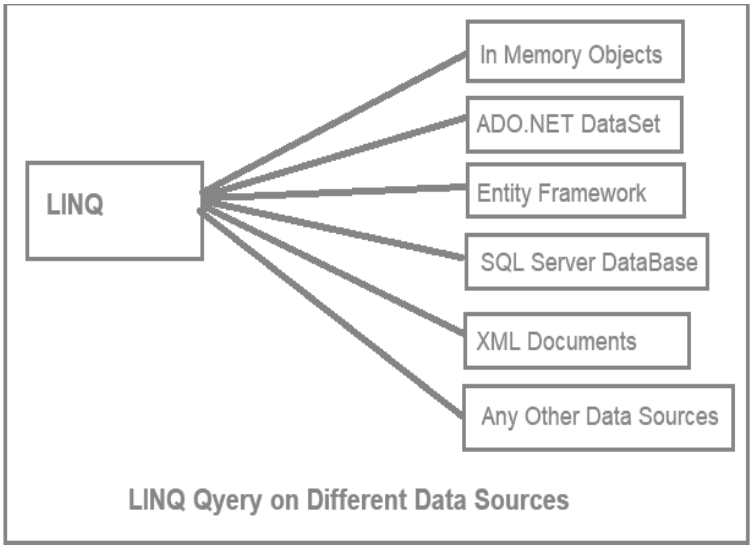


Fig. 5.4 LINQ Query on Different Data Sources

Why should we learn LINQ?

Let us understand why we should learn LINQ with an example.

Suppose we are developing a .NET Application and that application requires data from different data sources. For example

1. The application needs data from SQL Server Database. So as a developer in order to access the data from SQL Server Database, we need to understand ADO.NET and SQL Server-specific syntaxes. If the database is Oracle then we need to learn SQL Syntax specific to Oracle Database.

2. The application also needs data from an XML Document. So as a developer in order to work with XML Document, we need to understand **XPath and XSLT** queries.

3. The application also needs to manipulate the data (objects) in memory such as **List<Products>, List<Orders>**, etc. So as a developer we should also have to understand how to work with in-memory objects.

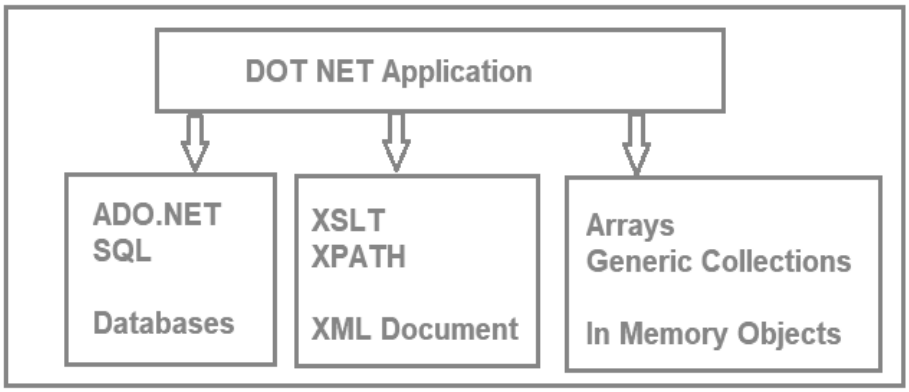


Fig. 5.5 Data from different data sources

LINQ provides a uniform programming model (i.e. common query syntax) which allows us to work with different data sources but using a standard or you can say unified coding style. As a result, we don't require learning different syntaxes to query different data sources.

How LINQ works?

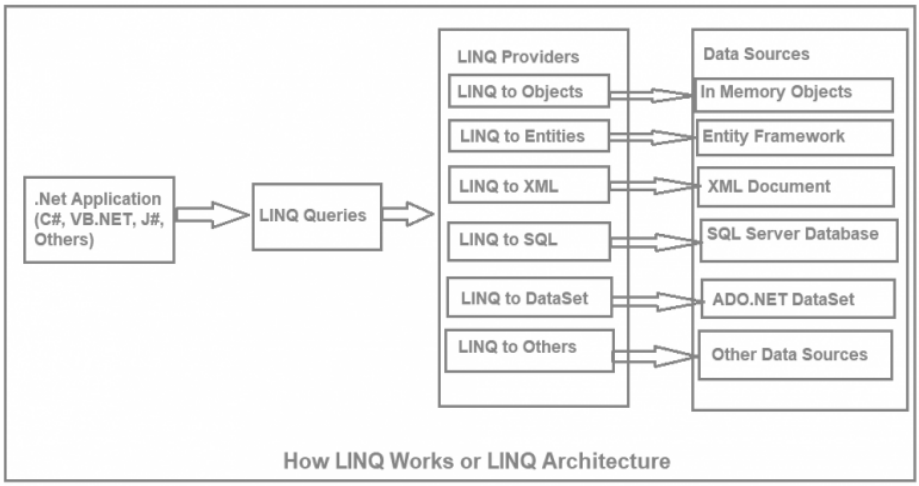


Fig. 5.6 Working of LINQ

As shown in the above diagram, you can write the LINQ queries using any .NET supported programming languages such as C#, VB.NET, J#, etc.

The LINQ provider is a software component that lies between the LINQ queries and the actual data source. The Linq provider will convert the LINQ queries into a format that can be understood by the underlying data source. For example, **LINQ to SQL** provider will convert the LINQ queries to SQL statements which can be understood by the SQL Server database. Similarly, the **LINQ to XML** provider will convert the queries into a format that can be understood by the XML document.

What are LINQ Providers?

A LINQ provider is software that implements the **IQueryProvider** and **IQueryable** interface for a particular data source. In other words, it allows us to write LINQ queries against that data source. If you want to create your custom LINQ provider then it must implement **IQueryProvider** and **IQueryable** interface. Without LINQ Provider we cannot execute our LINQ Queries.

Let us discuss some of the LINQ Providers and how they work internally.

LINQ to Objects:

The LINQ to Objects provider allows us to query an in-memory object such as an array, collection, and generics types. It provides many built-in functions that we can use to perform many useful operations such as filtering, ordering, and grouping with minimum code.

LINQ to SQL (DLINQ):

The LINQ to SQL Provider is designed to work with only the SQL Server database. You can consider this as an object-relational mapping (ORM) framework which allows one to one mapping between the SQL Server database and related .NET Classes. These .NET classes are going to be created automatically by the wizard based on the database table.

LINQ to Datasets:

The LINQ to Datasets provider provides us the flexibility to query data cached in a Dataset in an easy and faster way. It also allows us to do further data manipulation operations such as searching, filtering, sorting, etc. on the Dataset using the LINQ Syntax.

LINQ to Entities:

The LINQ to Entities provider looks like LINQ to SQL. It means it is also an object-relational mapping (ORM) framework that allows one to one, one to many, and many to many mapping between the database tables and .NET Classes. The point that you need to remember is, it is used to query any database such as SQL Server, Oracle, MySQL, DB2, etc. Now, it is called ADO.NET Entity Framework.

LINQ to XML (XLINQ):

The LINQ to XML provider is basically designed to work with an XML document. So, it allows us to perform different operations on XML data sources such as querying or reading, manipulating, modifying, and saving the changes to XML documents. The **System.Xml.Linq** namespace contains the required classes for LINQ to XML.

Parallel LINQ (PLINQ):

The Parallel LINQ or PLINQ was introduced with .NET Framework 4.0. This provider provides the flexibility of parallel implementation of LINQ to Objects. The PLINQ was designed in such a way that it uses the power of parallel programming which targets the Task Parallel Library. So if you want to execute your LINQ query simultaneously or parallel on different processors then you need to write the query using PLINQ.

Advantages of using LINQ?

1. We don’t need to learn new query language syntaxes for different data sources as it provides common query syntax to query different data sources.

2. Less code as compared to the traditional approach. That means using LINQ we can minimize our code.

3. It provides Compile time error checking as well as intelligence support in Visual Studio. This powerful feature helps us to avoid run-time errors.

4. LINQ provides a lot of inbuilt methods that we can use to perform different operations such as filtering, ordering, grouping, etc. which makes our work easy.

5. Its query can be reused.

Disadvantages of using LINQ?

The disadvantages of using LINQ are as follows:

1. Using LINQ it's very difficult to write complex queries like SQL.

2. LINQ doesn't take the full advantage of SQL features like cached execution plan for the stored procedure.

3. We will get the worst performance if we don't write the queries properly.

4. If we do some changes to our queries, then we need to recompile the application and need to redeploy the dll to the server.

5.5.2 Mapping Data Model To An Object Model:

The LINQ to SQL Object Model

In LINQ to SQL, an object model expressed in the programming language of the developer is mapped to the data model of a relational database. Operations on the data are then conducted according to the object model.

In this scenario, you do not issue database commands (for example, INSERT) to the database. Instead, you change values and execute methods within your object model. When you want to query the database or send it changes, LINQ to SQL translates your requests into the correct SQL commands and sends those commands to the database.

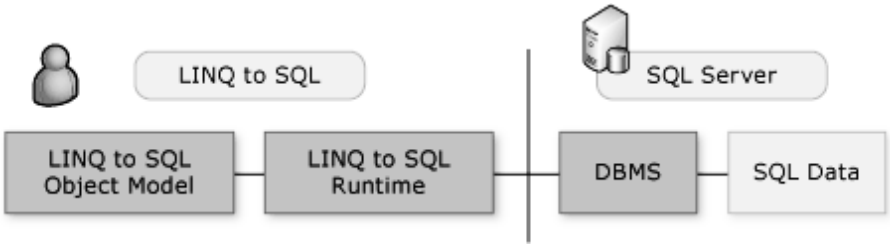


Fig. 5.7 The LINQ to SQL Object Model

The most fundamental elements in the LINQ to SQL object model and their relationship to elements in the relational data model are summarized in the following table:

LINQ to SQL Object Model	Relational Data Model
Entity class	Table
Class member	Column
Association	Foreign-key relationship
Method	Stored Procedure or Function

5.5.3 Introducing Query Syntax:

LinqTo SQL

Link to SQL is Object Relation Mapping (ORM) framework. It creates the strongly typed dot net classes based on database tables. LINQ to SQL enables you to write Select, Insert, Update, Delete query after after strongly typed dot net classes generated. Behind the seen Link to SQL provider converts LINQ query to normal SQL query, which is understands by SQL server database. LINQ to SQL type only supports SQL server database. It also supports Views, StoredProcedures, and Transactions.

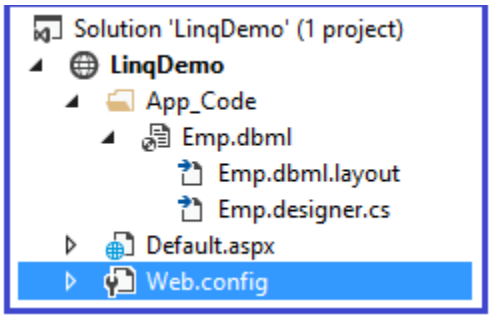
First we will create the database table, name tblEmps in SQL server as:

Column Name	Data Type	Allow Nulls
EmpID	int	☐
Name	nvarchar(50)	☑
Gender	nvarchar(50)	☑
Salary	float	☑
Address	nvarchar(50)	☑
DEPID	varchar(50)	☑

You can create the table according to your need and provide the table name.

Now Right click on the project (or project folder) and select the option Add New Item.

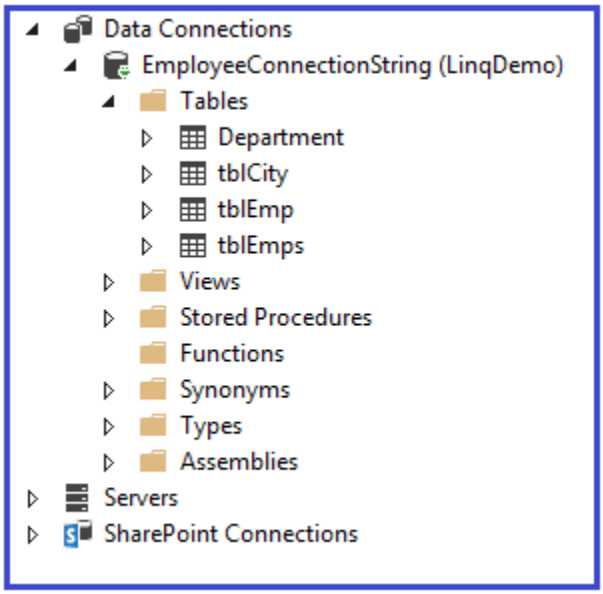
Select LINQ to SQL Classes Template and Give Name as Emp.dbml. You can provide the name according to application requirement. Emp.dbml file will be added under App_Code folder.



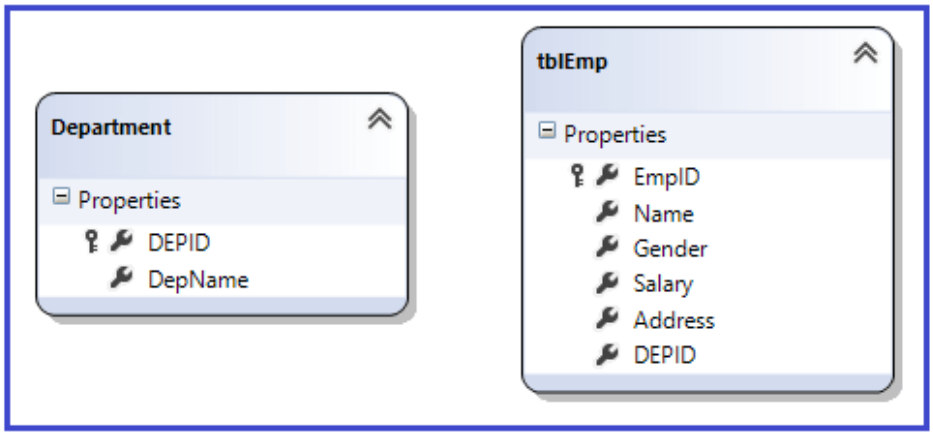
Click on Server Explorer and Right click on the Data Connections and select the option Add Connection.

Add Connection Pop up window will be opened, provide the SQL Server details and click on OK button.

Database will be added under Data Connections as shown below.



Drag the table in the left pane. If primary key & foreign key relations are there then it will automatically displayed. I have not created the foreign key explicitly. Therefore the relation is not shown.



This process generates the Emp.dbml.layout and Emp.designer.cs file under the Emp.dbml

It will also create the connection string automatically in web.config file.

```
<configuration>
  <connectionStrings>
    <add name="EmployeeConnectionString" connectionString="Data
Source=.;Initial Catalog=Employee;Integrated Security=True"
      providerName="System.Data.SqlClient" />
  </connectionStrings>
</configuration>
```

Click on Emp.designer.cs file, you will see EmpDataContext class that is inherited from DataContext class. Remember that we have created the Emp.dbml file and it will generate the EmpDataContext class. Whatever

file name you will provide, it generates the class with class name appended with DataContext as suffix.

```
using System;
using System.Collections.Generic;
using System.Linq;
public partial class _Default : System.Web.UI.Page
{
   protected void Page_Load(object sender, EventArgs e)
   {
      if(! IsPostBack)
      {
         LoadData();
      }
   }
   protected void LoadData()
   {
      EmpDataContextdbContext = new EmpDataContext();
      GridView1.DataSource= from emp in dbContext.tblEmps
                            select emp;
      GridView1.DataBind();
   }
}
```

Output:

EmpID	Name	Gender	Salary	Address	DEPID
2	Ramesh Gupta	Male	20000	Nagpur	1
3	David	Male	20000	USA	2
4	Zeny	Female	42000	Canada	3
5	Nishant	Male	50000	Pune	3
6	Digvijay	Male	45000	New York	3
7	Digambhar	Male	47000	Pune	4

LINQ have rich set of operators. It provides the operator for Filtering, Sorting, Grouping, Aggregation, Projection, Partitioning, Concatenation etc.

You can use any operators with the LINQ to get the result.

You can create a projection that matches the employee object structure. In the above query it selects all the records. If you want that few column should be display, then you can use linq query as

```
fromemp in dbContext.tblEmps
select  new{emp.EmpID , emp.Name};
```

Controlling column ordering and column name using LINQ:

LINQ enables you to change the order of column for display. You can also provide the custom name for column.

```
fromemp in dbContext.tblEmps
select new {
DepartmentID=emp.DEPID,EmployeeID=emp.EmpID,EmpName=emp.N
ame };
```

Query filters:

LINQ provides the where clause to filter the query.

Suppose that you want the record of those employees whose name is starts with letter “R” and salary greater than 20000.

```
fromemp in dbContext.tblEmps
where emp.Name.StartsWith("D") &&emp.Salary>20000
select emp;
```

Insert record using LINQ:

For inserting the record into the database using LINQ, first create the object of EmpDataContext class.

```
EmpDataContextdbContext = new EmpDataContext();
```

In our example the table name is tblEmps. LINQ automatically generates the tblEmps as a class and the table column as properties.

Now create the object of tblEmps class, so you can access the properties of this class.

InsertOnSubmit() method is used to insert the record. SubmitChanges() method commits all the changes into the database.

Please refer the previous code for populate the Gridview using LINQ. In above example we have used LoadData() method for this purpose.

```
protected void btnInsert_Click(object sender, EventArgs e)
  {
     using(EmpDataContextdbContext =new EmpDataContext())
     {
        tblEmpempObj = new tblEmp
        {
           EmpID=Convert.ToInt32( txtEmpID.Text),
           Name=txtName.Text,
           Gender=txtGender.Text,
           Salary=Convert.ToDouble( txtSalary.Text),
           Address=txtAddress.Text,
           DEPID = txtDepID.Text
        };
```

```
            dbContext.tblEmps.InsertOnSubmit(empObj);
            dbContext.SubmitChanges();
        }
        GetData();
    }
```

Update record using LINQ:

For updating the record, first we need a unique ID of employee whose record we want to update. For getting single record, SingleOrDefault is used in LINQ. After that SubmitChanges() method is used to commit the updated record in database.

In the given below code, you can easily see that, the table is converted into the class and the column name is converted into the properties of tblEmps class.

```
protected void btnUpdate_Click(object sender, EventArgs e)
    {
        using (EmpDataContextdbContext = new EmpDataContext())
        {
            tblEmpobj=dbContext.tblEmps.SingleOrDefault(em =>em.EmpID
== Convert.ToInt32(txtEmpID.Text));
                obj.Name=txtName.Text;
                obj.Gender=txtGender.Text;
                obj.Salary=Convert.ToDouble( txtSalary.Text);
                obj.Address=txtAddress.Text;
                obj.DEPID = txtDepID.Text;
                dbContext.SubmitChanges();
        }
        GetData();
    }
```

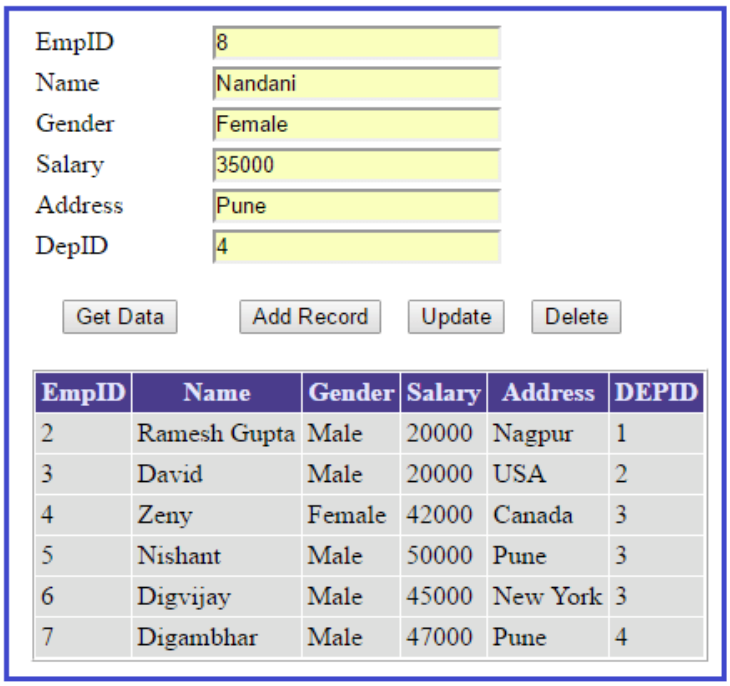
Delete record using LINQ:

For deleting the record, we also need a single record according to unique ID of employee. Here for deletion DeleteOnSubmit() method is used.

```
protected void btnDelete_Click(object sender, EventArgs e)
    {
        using (EmpDataContextdbContext = new EmpDataContext())
        {
            tblEmpobj = dbContext.tblEmps.SingleOrDefault(em
=>em.EmpID == Convert.ToInt32(txtEmpID.Text));
            dbContext.tblEmps.DeleteOnSubmit(obj);
            dbContext.SubmitChanges();
        }
        GetData();
    }
```

In this given example only seven records are available. Here we have entered an eighth record, when you click on AddRecord button,

btnInsert_Click method will be executed and record is added in the database.



EmpID	Name	Gender	Salary	Address	DEPID
2	Ramesh Gupta	Male	20000	Nagpur	1
3	David	Male	20000	USA	2
4	Zeny	Female	42000	Canada	3
5	Nishant	Male	50000	Pune	3
6	Digvijay	Male	45000	New York	3
7	Digambhar	Male	47000	Pune	4

Getting the actual sql query generated by LINQ:

As you know that SQL server can understand only standard SQL queries and objects. Therefore LINQ to SQL convert LINQ queries into standard SQL query format.

You can easily see the sql query, which is generated by LINQ to SQL.

There are several methods to achieve for the same.

```
EmpDataContextdbContext = new EmpDataContext();
varselectQuery = from emp in dbContext.tblEmps
                 select emp;
```

First Method:

You can use the log property of DataContest class. It writes the generated sql query at provided I/O.

```
dbContext.Log = Response.Output;
```

Second Method:

You can directly use the ToString() method as:

```
Response.Write(selectQuery.ToString());
```

Third Method:

GetCommand method of DataContext class provides the information about sql command generated by LINQ to SQL.

stringsqlQuery=dbContext.GetCommand(selectQuery).CommandText;

Response.Write(sqlQuery);

5.5.4 Entity Framework:

What is Entity Framework in .NET Framework?

Entity Framework is an open-source object-relational mapper framework for .NET applications supported by Microsoft. It increases the developer's productivity as it enables developers to work with data using objects of domain-specific classes without focusing on the underlying database tables and columns where this data is stored. It eliminates the need for most of the data-access code which is used to interact with the database that developers usually need to write. It provides an abstract level to the developers to work with a relational table and columns by using the domain-specific object. It also reduces the code size of the data specific applications and also the readability of the code increases by using it. This is a new technology for accessing the data for Microsoft application. The latest version for Entity Framework is 6.0.

The following figure describes where the Entity Framework presents in your application.

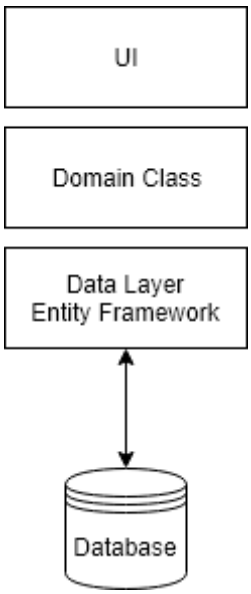


Fig. 5.8 Entity Framework

The above figure represents how an entity framework interacts with the domain class and database. It provides a connection between the business entity and data tables in the database. It saves data stored in the properties of business entities and also retrieves data from the database and converts it to business entities objects automatically. Entity Framework will

execute the relevant query in the database and then materialize results into instances of your domain objects for you to work within your app.

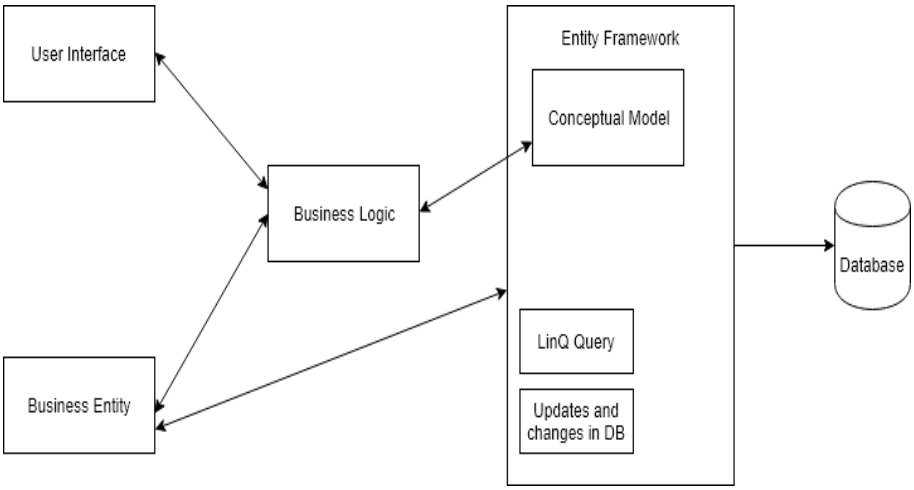


Fig. 5.9 Conceptual Model

When defining the class and features of entity framework first and then entity framework convert it into the conceptual model first and it creates database and objects in the database from the conceptual model this method is called Code First. Now your object directly works with the database to retrieve or make changes.

Features of Entity Framework:

- It is platform independent.
- It uses LinQ queries to manipulate the data in the database instead of SQL queries.
- It keeps the track of values that have been changed of the properties of entities.
- It also save changes which are done insert, delete or update operations.
- It also handles concurrency so the data override by a user and will reflect when another use fetches it.
- It also handles transaction management automatically and also provides the customize options for transaction management.
- It provides caching which means it stores the result of the frequently used queries.
- It also follows certain conventions for programming so it by default configure the EF Model.

- It also allows configuring the EF Model by a fluent API to override the default convention.
- If you made any changes in database schema then you can reflect those changes in EF model by writing migration command in CLI (Command Line Interface).
- It also supports stored procedure.
- It also supports parameterized queries.

5.6 LET US SUM UP

This chapter covered features of ADO.NET. These features are designed to give you the flexibility to handle database processing in a manner never before possible with either of the previous versions of ADO. We studied the components of ADO.NET that is Provider, Connection, Command, Dataset, Datareader and Dataadapter.

The LINQ with ASP.NET,In LINQ to SQL, and an object model expressed in the programming language of the developer is mapped to the data model of a relational database and Query syntax. Entity Framework is an open-source object-relational mapper framework for .NET applications supported by Microsoft.

5.7 UNIT END EXERCISES

1) What is ADO.NET?
2) Explain the advantages of ADO.NET.
3) Explain ADO.NET data architecture.
4) Explain different data provider.
5) Explain connection and command with example
6) Explain data adapter and dataset.
7) Explain data grid with example.
8) Explain data reader with example.
9) Explain the architecture of LINQ.
10) What is The LINQ to SQL Object Model?
11) Explain the working of LINQ.
12) Explain the features of Entity Framework.

5.8 LIST OF REFERENCES

Reference No	Reference Name
1	https://www.tutorialspoint.com/asp.net/asp.net_database_access.htm
2	https://www.guru99.com/insert-update-delete-asp-net.html
3	http://asp.net-informations.com/dbpro/asp-dbpro.htm
4	https://www.w3schools.com/asp/webpages_database.asp
5	https://www.geeksforgeeks.org/what-is-entity-framework-in-net-framework/
6	https://www.tutorialride.com/asp-net/linq-in-asp-net.htm

*****

6

DATABOUND CONTROLS IN ASP.NET

Unit Structure

Self-Learning Topics: Charts and Data Pagers

6.0 OBJECTIVES

What You Will Learn in This Chapter:

- Learn about how to bind data to data bind control
  - DataList
  - DetailsView
  - FormView
  - GridView
  - ListView
  - Repeater

6.1 AN OVERVIEW

Databound controls are used to display data to the end-user within the web applications and using databound controls allows you to manipulate the data within the web applications very easily.

Databound controls are bound to the DataSource property. Databound controls are composite controls that combine other ASP.NET Controls like Text boxes, Radio buttons, Buttons and so on.

6.2 SQLDATASOURCE CONTROL

The SqlDataSource control is used to access data located in the relational database. It uses the ADO.NET classes to interact with any databases that are supported by ADO.NET. The providers that can be used are oracle, SQL server, ODBC and OleDb. Using the control user can access and manipulate data in the ASP.NET page.

To use the SqlDataSource control, set the **ProviderName** and **ConnectionString** property. The **ProviderName** is used for the type of the database used and **ConnectionString** for the connection to the database.

The data of the SqlDataSource control can be cached and retrieved in the application. The caching can be enabled by setting the **EnableCaching** property to true. The filter of the SqlDataSource control supports a **FilterExpression** property. User can add the selection criteria for filtering of data.

The SqlDataSource control code sample is as shown below:

```
<asp:SqlDataSource id=”sldatasource1” runat=”server”

DataSourceMode=”DataReader”
ConnectionString=”<%ConnectionStrings:EmployeeData%>”

SelectCommand=”Select name from Employee” >

</asp:SqlDataSource>
```

6.3 DATABOUND CONTROLS

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

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

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

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

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

- BulletedList

- CheckBoxList
- DropDownList
- ListBox
- RadioButtonList

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

- DetailsView
- FormView
- GridView
- RecordList

6.3.1 Data List Control:

The DataList control is used to display a repeated list of items that are bound to the control. There are different templates using which user can design the layout of the control. The template property are mentioned below:

Template Property	Description
ItemTemplate	It contains the HTML elements and controls to render for each row in the data source
AlternatingItemTemplate	It contains the HTML elements and controls to render once for every other row in the data source
SelectedItemTemplate	It contains the elements to render when the user selects an item in the DataList control
EditItemTemplate	It specifies the layout of an item when the edit mode is working
HeaderTemplate	It contains all the text and controls to be rendered at the beginning of the list
FooterTemplate	It contains all the text and controls to be rendered at the end of the list
SeperatorTemplate	It contains all elements to render between each item

A sample code to demonstrate the DataList control is as shown below:

```
<%@Page Language=”C#” %>
<! DOCTYPE html>
<script runat=”server”>
</script>
<html xmlns=”http://www.w3.org/1999/xhtml”>
```

```
 <head runat=”server”>
  <title></title>
 </head>
 <body>
  <form id=”form1” runat=”server”>
   <div>
   <asp:DataList ID=”DataList1” runat=”server”
DataKeyField=”Id” DataSourceID=”SqlDataSource1”
Height=”285px” RepeatColumns=”3”
RepeatDirection=”Horizontal” Width=”134px”>
   <ItemTemplate>
   Id:
   <asp:Label ID=”IdLabel” runat=”server” Text=’<%# Eval ( “Id”
)%>’ />
   <br/>
   name:
   <asp:Label ID=”nameLabel” runat=”server” Text=’<%# Eval (
“name” )%>’ />
   <br/>
   Income:
   <asp:Label ID=”IncomeLabel” runat=”server” Text=’<%# Eval (
“Income” )%>’ />
   <br/>
   <br/>
     </ItemTemplate>
     </asp:DataList>
     <asp:SqlDataSource ID=”SqlDataSource1” runat=”server”
ConnectionString=’<%$ConnectionStrings:ConnectionString %>”
     SelectCommand=”SELECT * FROM [footerex]”>
     </asp:SqlDataSource>
   </div>
  </form>
 </body>
</html>
```

Output is:

Id: 1	Id: 2	Id: 3
namet:	namet:	namet:
jacob	yu	YU
Income:	Income:	Income:
25000	5000	10000

6.3.2 Details View Control:

Details view control is used as a data bound control. It is used to render one record at a time. User can insert, update and delete the record from the control. Some of the properties of the DetailsView control is as shown below:

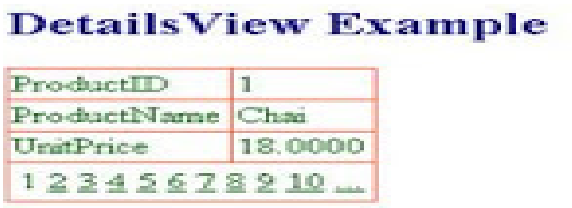
Property	Description
AllowPaging	It is a Boolean value to indicate the control supports navigation
DataSource	It is used to populate the control with the data source object
DataSourceID	It indicates the bound data source control with corresponding adapters
AutoGenerateEditButton	It is a Boolean value to indicate the column can be edited by the user
AutoGenerateDeleteButton	It is a Boolean value to indicate the records can be deleted
DefaultMode	It is used to indicate the default display mode of the control
AutoGenerateRows	It is a Boolean value to indicate the rows are automatically created for each field in the data source

The sample code for the Details View control is as shown below:

```
<%@ Page Language=”C#” %>

<!DOCTYPE html PUBLIC “-//W3C//DTD XHTML 1.0
Transitional //EN”
“http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd” >

<script runat=”server”>
</script>

<html xmlns=”http://www.w3.org/1999/xhtml”>
 <head runat=”server”>
   <title>asp.net DetailsView example: how to use
DetailsView</title>
 </head>
 <body>
   <form id=”form1” runat=”server” >
   <div>
   <h2 style=”color:Navy” >DetailsView Example</h2>
   <asp:SqlDataSource ID=”SqlDataSource1” runat=”server”
ConnectionString=”<%$ ConnectionStrings:
NorthwindConnectionString %>”
   Select Command = “SELECT ProductID, ProductName,
UnitPrice FROM Products”;
   </asp:SqlDataSource>
   <asp:DetailsView ID=”DetailsView” runat=”server”
DataSourceID=”SqlDataSource1” AllowPaging=”true”
ForeColor=”DarkGreen” BackColor=”Snow”
```

```
BorderColor=”Tomato”>
   </asp:DetailsView>
   </div>
   </form>
 </body>
</html>
```

The output is:



6.3.3 Formview Control:

The FormView control is data bound control but it uses templates version of the DetailsView control. The templates are used inside the control for rendering it on the server. Some of the properties of the FormView control are as shown below:

Property	Description
EditItemTemplate	It is used when the record is being edited by the user
InsertItemTemplate	It is used when a record is being created by the user
ItemTemplate	It is the template used to render the record to display only in an application

Methods of FormView control

Methods	Description
InsertItem	It is used to insert record in the database
UpdateItem	It is used to update record in the database
DeleteItem	It is used to delete the record in the database
ChangeMode	It is used to change the working state of the control

The sample code for the FormView control is as shown below:

```
<%@ Page Languag”C#” AutoEventWireup=”true”
CodeFile=”FormView.aspx.cs” Inherits=”FormView” %>

<!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>asp.net FormView: how to use</title>
 </head>
 <body>
   <form id="form1" runat="server">
   <div>
     <h2 style="color:Teal">FormView</h2>
     <asp:SqlDataSource ID="SqlDataSource1" runat="server"
ConnectionString="<%$ConnectionStrings:Northwind %>"
SelectCommand = "SELECT CategoryID, CategoryName,
Description FROM Categories">
     </asp:SqlDataSource>
     <asp:FormView ID="FormView1" runat="server"
DataSourceID="SqlDataSource1" AllowPaging="true">
      <HeaderTemplate>
        Product Catogory
      </HeaderTemplate>
      <ItemTemplate>
        CategoryID:<%#Eval ( "CategoryID")%><br/>
        CategoryName:<%#Eval ( "CategoryName")%><br/>
        Description:<%#Eval ( "Description" )%>
      </ItemTemplate>
      <pagerSettings Mode="Numeric" />
     </asp:FormView>
   </div>
   </form>
 </body>
</html>
```

The code behind file for the control is as shown below:

```
public partial class FormView: System.Web.UI.Page
{
  protected void Page_Load( object sender, EventArgs e)
  {
   if(!this.IsPostBack)
   {
FormView1.HeaderStyle.BackColor=System.Drawing.Color.SeaGreen;
FormView1.HeaderStyle.ForeColor=System.Drawing.Color.Snow;
FormView1.HeaderStyle.Font.Bold=true;
FormView1.PagerStyle.BackColor=System.Drawing.Color.ForestGreen;
FormView1.PagerStyle.ForeColor=System.Drawing.Color.AliceBlue;
FormView1.RowStyle.BackColor=System.Drawing.Color.Green;
FormView1.RowStyle.ForeColor=System.Drawing.Color.White;
   }
  }
}
```

Output is:

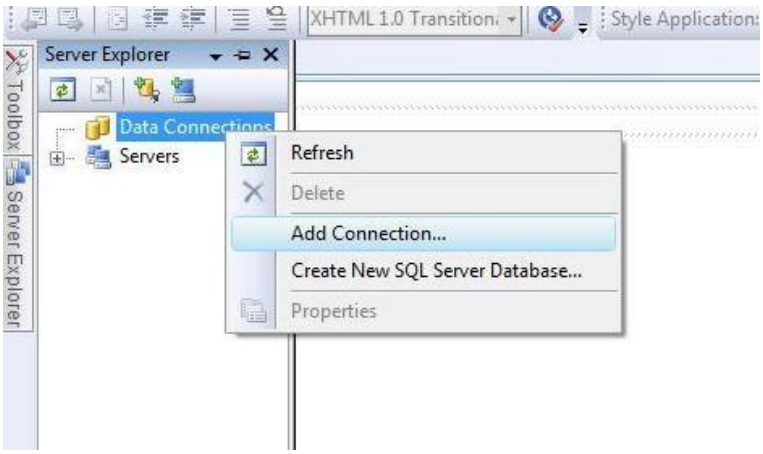
6.3.4 Gridveiw Control:

The GridView control is used to provide flexibility in working and display data from the database. The data in the GridView control is displayed in the tabular format. It has several properties assigned to the control. Some of the properties are as mentioned below:

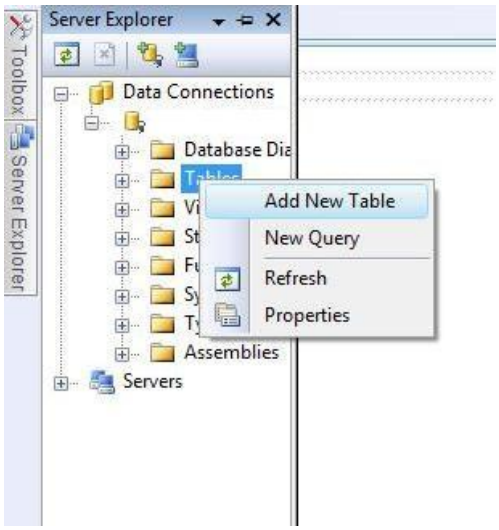
Property	Description
AllowPaging	It is a Boolean value indicating the control supports paging
AllowSorting	It is a Boolean value indicating the control supports sorting
SortExpression	It accepts the current expression determining the order of the row
Datasource	It is used to get or set the data source object containing the data to populate the control
AutoGenerateEditButton	It is a Boolean value indicating that the user can edit the record in the control
DataSourceID	It indicates the data source control to be used
AutoGenerateDeleteButton	It is a Boolean value indicating that the user can delete the record in the control
AutoGenerateColumns	It is a Boolean value to indicate the columns are automatically created for each field of the data source
AutoGenerateSelectButton	It is a Boolean value to indicate the column should be added to select the particular record
SortDirection	It gets the sorting direction of the column for the control
EmptyDataText	It indicates the text to appear when there is no record in the data source

The code sample of the GridView control is as shown below:

1) Add a new connection object to the ASP.NET web application as shown below:

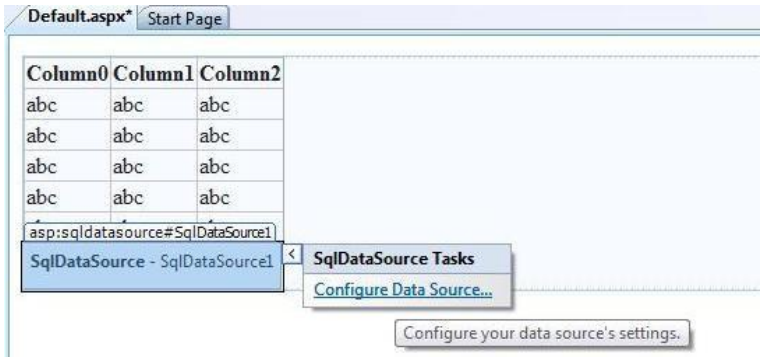


2) Next, add a new table for the connection object created above. The snippet for adding the table is as shown below:



3) Add the fields Sno, Name, Address in the table. Add values to the respective fields in the table

4) Add the GridView and SqlDataSource control to the design view of the web page.



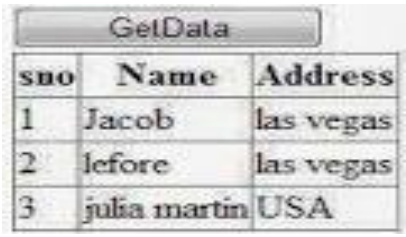
5) The source code for the GridView control is as shown below:

```
<%@Page Language=”C#” AutoEventWireup=”true”
CodeFile=”binding.aspx.cs” Inherits=”binding” %>

<!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” >
     <asp:Button ID=”Button1” runat=”server” onclick=”Button1_Click”
Text=”GetData” Width=”123px” />
     <br/>
     <div>
    <asp:GridView ID=”GridView1” runat=”server”>
    </asp:GridView>
     </div>
  </form>
 </body>
</html>
```

6) The code behind file contains the following code

```
protected void Button1_Click(object sender, EventArgs e)
{
  SqlConnection con = new SqlConnection();
  con.ConnectionString = ConfigurationManager.ConnectionStrings
  [ “ConnectionString” ].ToString();
    con.Open();

  SqlCommand cmd = new SqlCommand();
  cmd.CommandText = “Select * from deltable”;
  cmd.Connection = con;
  DataSet ds = new DataSet();
  da.Fill( ds, “deltable”);
  GridView1.DataSource= ds;
  GridView1.DataBind();
}
```

The output is:



6.3.5 Listview Control:

The ListView control is used to bind to data items returned to the data source and display them. The control displays data in a format defined by using templates and styles. The list of templates supported by the control are as shown below:

Templates	Description
ItemTemplate	It identifies the data bound content to display for single items
ItemSeperatorTemplate	It identifies the content to be rendered between the items
LayoutTemplate	It identifies the root template that defines the main layout
GroupTemplate	It identifies the content of the group layout
GroupSeperatorTemplate	It identifies the content to be rendered between the group of items
EmptyItemTemplate	It identifies the control to render for an empty item when the GroupTemplate is used
EmptyDataTemplate	It identifies the content to render if the data source returns no data
SelectedItemTemplate	It identifies the content to render for the selected data item to differentiate the selected item from the other displayed items
EditItemTemplate	It identifies the content to render when the item is lost
InsertItemTemplate	It identifies the content to render when an item is being inserted

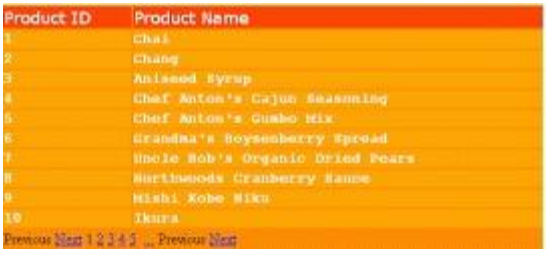
A sample code for the list control is as shown below:

```
        <html xmlns=”http://www.w3.org/1999/xhtml”>
<head id”Head1” runat=”server”>
  <title>ListViewControl – how to use ListView control in asp.net </title>
    <style type=”text/css”>
   .TableCSS
   {
     border-style:none;
     background-color:DarkOrange;
      width:600px;
   }
```

```
   .TableHeader
   {
     background-color:OrangeRed;
     color:snow;
     font-size:large;
     font-family:Verdana;
   }
   .TableData
   {
     Background-color:Orange;
     color:Snow;
     font-family:Courier New;
     font-size: medium;
     font-weight:bold;
      }
    </style>
</head>

 <body>
  <form id="form1" runat="server">
   <div>
   <h2 style="color:Navy; font-style:italic;">ListView Control Example:
How to Use ListView Control</h2>
   <hr width="550" align="left" color="PowderBlue" />
   <asp:SqlDataSource ID="SqlDataSource1" runat="server"
ConnectionString="<%$ConnectionStrings:NorthwindConnectionString"
%>"
   SelectCommand = "Select ProductID, ProductName From products
Order By ProductId"
   >
   </asp:SqlDataSource>
   <br/>
   <asp:ListView ID="ListView1" runat="server"
DataSourceID="SqlDataSource1" >
   <LayoutTemplate>
   <table runat="server" class="TableCSS">
     <tr runat="server" class="TableHeader">
      <td runat="server">ProductID</td>
      <td runat="server">ProductName</td>
     </tr>
     <tr id="ItemPlaceHolder" runat="server">
     <tr runat="server">
      <td runat="server" colspan="2">
      <asp:DataPager ID="DataPager1" runat="server">
      <Fields>
       <asp:NextPreviousPageField ButtonType="Link" />
      </Fields>
      </asp:DataPager>
      </td>
     </tr>
```

```
    </table>
    </LayoutTemplate>
      <ItemTemplate>
        <tr class=”TableData” >
        <td>
          <asp:Label ID=”Label1” runat=”server”
Text=’<%#Eval(“ProductID”)%>’>
          </asp:Label>
        </td>
        <td>
          <asp:Label ID=”Label2” runat=”server” Text=’<%#Eval
(“ProductName”)%>’>
          </asp:Label>
        </td>
        </tr>
       </ItemTemplate>
    </asp:ListView>
     </div>
   </form>
</body>
```

Output is:



6.3.6 Repeater Control:

The Repeater control is a data bound control created by using the templates to display items. The control does not support editing, paging or sorting of data rendered through the control. The list of templates supported by the Repeater control are as mentioned below:

Templates	Description
HeaderTemplate	It contains all the text and controls to be rendered at the beginning of the list
FooterTemplate	It contains all the text and controls to be rendered at the end of the list
AlternatingItemTemplate	It contains the HTML elements and controls to render once for every other row in the data source
SeperatorTemplate	It contains all elements to render between each item
ItemTemplate	It contains the HTML elements and controls to render for each row in the data source

The sample code for the Repeater control is as shown below:

```
<body>
  <form id=”form1” runat=”server”>
    <div>
      <asp:Repeater ID=”RepeaterInformation” runat=”server”>
    <HeaderTemplate>
      <table class=”tblcolor”>
      <tr>
         <b>
         <td>
         Roll No
         </td>
         <td>
         StudentName
         </td>
         <td>
         Total Fees
         </td>
         </b>
      </HeaderTemplate>
      </ItemTemplate>
      <tr class=”tblrowcolor”>
      <td>
         <%#DataBinder.Eval ( Contiane.”DataItem.RollNo”)%>
      </td>
      <td>
         <%#DataBinder.Eval(Contianer,”DataItem.Name”)%>
      </td>
      <td>
         <%#DataBinder.Eval(Contianer.”DataItem.Fees”)%>
      </td>
      </tr>
      </ItemTemplate>
      <SeperatorTemplate>
      <tr>
      <td>
         <hr/>
      </td>
      <td>
         <hr/>
      </td>
      <td>
         <hr/>
      </td>
      </tr>
      </SeperatorTemplate>
      <FooterTemplate>
      <tr>
      <td>
```

```
            School Records displayed
          </td>
          </tr>
          </table>
          </FooterTemplate>
      </asp:Repeater>
       </div>
    </form>
</body>
```

The code behind file is as shown below:

```
        public partial class _Default: System.Web.UI.Page
{
  SqlConnection con;
  SqlCommand cmd = new SqlCommand();
  protected void Page_Load( object sender, EventArgs e)
  {
    con=new SqlConnection(
ConfigurationManager.ConnectionStrings[“constr”].ConnectionString);
    cmd.Connection=con;
    com.Open();
    RepeatInformation.DataSource = cmd.ExecuteReader();
    RepeatInformation.DataBind();
    con.Close();
  }
}
```

Output is:

Roll No	Student Name	Total Fees
1	Patrick	3400
2	John	4500
3	York	1200
School Records displayed		

6.4 LET US SUM UP

This chapter covered the concept of data-binding and its different classifications in ASP.NET. Data Binding is a necessity while developing any application and ASP.NET reduces the workload of programmers and makes it necessary for them to explore more about it.

6.5 UNIT END EXERCISES

1. What is data binding and its types? Explain with example?

2. Define:
    a. DataList
    b. DetailsView
    c. FormView
    d. GridView
    e. ListView
    f. Repeater
3. Explain DataList in detail?
4. Explain DetailView in detail?
5. Explain FormView in detail?
6. Explain GridView with example?
7. Explain ListView with example?
8. Explain Repeater in detail?

6.6 LIST OF REFERENCES

Reference No	Reference Name
1	https://www.tutorialspoint.com/asp.net/asp.net_database_access.htm
2	https://www.guru99.com/insert-update-delete-asp-net.html
3	http://asp.net-informations.com/dbpro/asp-dbpro.htm
4	https://www.w3schools.com/asp/webpages_database.asp

*****

MODULE IV

7

SESSION MANAGEMENT

Unit Structure

7.0 OBJECTIVE

In this chapter we learned sate management and its types i.e client side state management and server-side state management.

We learned also View state,querystring Cookie, how to be hidden field in the Client-Side state management.

7.1 INTRODUCTION

Microsoft frameworks are strongly taken into consideration for their security capabilities. Some of the surprising functions include authentication-authorization control, HTTPS enforcement, error management with Global Exception Handling support in ASP.NET Core, CORS control, etc.

Talking about HTTP, that is this kind of protocol that doesn't have a status. Therefore, if we want to transfer statistics from one page to some other page or even after numerous visits to the identical page, we have to

use country control techniques. Today, we're going to research kingdom control techniques the usage of ASP.NET.

State Management is the procedure in which developers can keep repute and web page data on more than one requests for the identical or extraordinary pages in the net application.

7.2 STATE MANAGEMENT

1. Meaning of State management method is to preserve state of control, webpage, Item/data and consumer/user within the application.
2. In the state management, a new example of the web page class is created on every occasion or every time the page is posted to the server.
3. It approach that everyone information related to the web page and the controls on the web page would be misplaced with every spherical ride.
4. It is nothing but the current value of the page or data
5. Data want be hold in such styles of situations:

A. While navigating through internet web page

B. web page performs the round trip

Why we need state management?:

We need state management because when the user visit to his/her, application browser does the communication with the respective server which is depending upon their requested functionality.

Using HTTP protocol, the application browser or internet browser communicates with the server. When the response returns to client then it is just clean up the resources such as data/Object, Allocated Memory, Session Ids, URL Information.

What are the types of state Management?

State management has two types.

7.2.1 Client-side state management

7.2.2 Server-side state management

7.2.1 CLient-side state management:

In the client-side Management of user or client pages, during the end to end interaction the information is stored in the user or client system.

The main benefit of getting this form of state management is that it saves a tremendous amount of server reminiscence. We reduce the load on the server to preserve state information.

Due to the increased bandwidth, it reduced speed of loading and creating security issue for confidential data such as credit card number, password.

In the web application .NET is support following types of methods in the client-side state management:

7.2.1.1 View State

7.2.1.2 Query String

7.2.1.3 Cookie

7.2.1.4 Hidden field

7.2.1.1 View State:

View State is the method to hold the Value of the Page and Controls among round trips.

It is a Page-Level State Management technique. View State is grown to become on through default and typically serializes the facts in every manipulate at the web webpage irrespective of whether in reality used during the put up returned.

View state is property of every webpage which preserve or save value at the post back time. For the same pages, it is built-in structure for automatically saving the values between multiple requests. It stored the page control value as string which is encoding some hashing technology. To store the information or data view state use Hidden field in the hashing and encoding format. View state is not difficult to implement. There are no server resources are required. It is providing the security which can be compress.

Steps for view state:

1. Open New Project
2. Click on the web on the left side
3. select the ASP.NET EMPTY WEB Application
4. Now, Click on solution explorer, right click on ADD>New Item> Web Form>Name it> click on ADD button, after this you will see following code:

```
1. <%@ page language="C#" autoeventwireup="tru
   e" codebehind="WebForm6.aspx.cs" inherits="
   view_state.WebForm6" %>
2.
```

```
<!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>
          User Name:-
<asp:textbox id="TextBox1" runat="server"></asp:textbox>
          <br />
          Password  :-
<asp:textbox id="TextBox2" runat="server"></asp:textbox>
          <br />
          <asp:button id="Button1" runat="server" onclick="Button1_Click" text="Submit" />
          <asp:button id="Button3" runat="server" onclick="Button3_Click" text="Restore" />
      </div>
      </form>
  </body>
  </html>
```

Now right the code like

```
//Declaration of x and y
public string x, y;
protected void Button1_Click(object sender,
 EventArgs e)
{
    //TextBox1 and TextBox2 Value is Assigning on the variable a and b
    x = TextBox1.Text;
    y = TextBox2.Text;
    //after clicking on Button TextBox value Will be Cleared
    TextBox1.Text = TextBox2.Text = string.Empty;
```

```
10.  }
11.
12.  protected void Button3_Click(object sende
   r, EventArgs e)
13.  {
14.      //value of variable a and b is assing
   ning on TextBox1 and Textbox2
15.      TextBox1.Text = x;
16.      TextBox2.Text =y;
17.  }
18.
```

And execute this code.

You will get, output.

To save the variable in view state:

ViewState[“Subject”]=”Advanced Web Application”;

To retrieve the information in View State:

String value=ViewState[“Subject”];

7.2.1.2. Query string:

Query string is part of the URL, which is passing data or value one to another or next page

The size of URL is 255 character long for example user entered welcome page it will display on next page.

Query string is a easy way to bypass some information from one page to another. The facts cab be without problems pass to 1 web page to every other or to same web page. With query string technique the statistics exceeded in URL of page request.

For send data to different web page Response.Redirect() approach used and for retrieve facts from URL use Request.QueryString() approach used.

In Query String method we can despatched price to only desired web page, and fee will be quickly. So Query string increase the general overall performance of internet application.

Syntax of the query string:

Response.Redirect(“nextpage.aspx?name=value”);

Retrieve information from other page

Request.QueryString[“name”].ToString();

7.2.1.3. Cookie:

It provides in web application to store the user particular information. Whenever user or client visit to websites cookies to storing the preferences.

Cookie is small piece of text information which store the information and request and pages as they go between the web server and browser.

Cookies are associated with a web site not with specific page hence the application browser and server will exchange cookie information.

There two types of Cookies 1. Persistent Cookie and 2. Non-Persistent Cookie

1. Persistent cookie:

The persistent cookie permanently stored on hard drive of user.

Persistent cookie has expiry date time. This type of cookie permanently stored user hard drive till date time we set.

For example: Response.Cookies[“Name”].Value=”IDOL”;

Response.Cookies[“IDOL”].Expires=DateTime.Now.AddMinutes(30);

Using following syntax, we create Cookie:

HttpCookie strname= new HttpCookie(“name”);

Strname.Value=”IDOL”;

Strname.Expires=DateTime.Naow. AddMinutes(30);

Here, Response.Cookies is use to create the Cookie and 30 minutes is expiry time after 30 Minutes cookies will expire.

2. Non persistent cookie:

This type of cookie does not store permanently on user’s hard drive. It store whenever user accessing up information the same browser, once the browser will closed the information will delete automatically.

To read cookie information:

```
If (Request.Cookie[“Name”]!=null)
{Lable1.Test=Request.Cookies[“Name”].Value;
 }
```

7.2.1.4. Hidden Field:

It’s not anything new, as all of us understand from HTML programming. The hidden field is used to hide your client side records. It is not directly

seen to the client on the client interface; however, we may additionally see the value at the source page. So, it is not really helpful if you want to save sensitive information. It is simply used to shop a small amount of information that is often altered.

The Hidden fields are Html input control with hidden type that store in the Html hidden data

In their value property the hidden field can store one value only. The value is stored as string and there fore for some types you need use casting.

The following code statistics the employee variety and has a value of 1001.

7.2.2 Server-Side State Management:

In this type of cookie information is store in the memory. About the user this type of management stores the confidential and sensitive information and disadvantage is that it uses more storage from the server.

7.2.2.1 Session's state

7.2.2.2 Application State

7.2.2.3 Cache

7.2.2.4 Profile

7.2.2.1 Session Sate:

Session state is a time period of time to visit an internet site or website for a particular person. Session can save the user or client information on a server. Session is a quality country control features to shop the patron data on server separately for each consumer.

Session is a best state management features to store the client data on server separately for each user.

Each and every session has their unique ID. Using cookie this ID is being stored in the client machine.

By using session ID the server maintains the state of user information. Without session ID when user makes a request, the ASP is creating session ID and sends it with every request and response to same user.

Syntax of session:

Session["session_name"]="Session value";

Declare session:

Session["name"]="IDOL";
Response.Redirect("nextpage.aspx");

To retrive the session value:

String myvalue=session[“name”].Tostring();
Response.Write(“Name=”+myvalue);

Mechanism of session Managements are:

1. In Proc mode
2. Out Proc mode
   A. Sql Server
   B. State Server

7.2.2.2 Application state:

Application state information is global storage mechanism which is used to store data on server and shared for all clients or users that means data stored in application state is common for all. In that data can be accessible from anywhere in the application.

It is stored in memory on server and in the database it faster to storing and retrieving information. It is called application-level state management.

Data member can be declared and defined in global.asax

To save application data

```
void Session_Start(object sender,EventArgs e)
    {
     Application.Lock();
     Application[“PageCounter”]=(int)Application[“PageCounter”]+1;
     Application.UnLock();
    }
```

To read application data

```
void Page_Load(object sender, EventArgs e)
     {
      int Visitors=(int)Application[“PageCounter”];
      }
```

Following events are of Applications State Management:

1. Application_Start()
2. Session_Start()
3. Application_Error
4. Session_End

5. Application_End
6. Application_Disposed()

7.2.2.3 Cache:

To enhanced website performance caching features does helps. The cache can store any data object, pages they can store in the memory, when first time requested. User can store or cache data object, pages on the other software or web server in the request stream for example browser or proxy server. By allowing commonly requested content to be temporarily stored faster way it is improving server performance.

Types of Caching are as following:

1. Output Caching
2. Data Caching
3. Object Caching
4. Class Caching
5. Configuration Caching

7.2.2.4 Profile:

As the applications must have to manage the users' unique details or information such as last date of visited, require query or enquiry. The ASP.NET features helps to find or recognised users based unique identy. Its profile features associate's information with an individual user and stores the information in a persistent format.

Syntax:

<profile>

<properties>

<add name="LastDateOfVisit">

<add name="enquiry">

<add name="LastQuery">

</properties>

</profile>

7.3 SUMMARY

In this chapter we learned some state management mechanism using ASP.NET MVC. It will help us to understand various method used for enhancing it and provide an idea about state management.

7.4 UNIT END EXERCISES

1. Explain the types of state management.

2. What is Cache explain it with an example.
3. How does Cookie works?
4. What is client-side state management?
5. What is server-side state management?

7.5 LIST OF REFERENCES

- Freeman, Adam. "Pro asp. netmvc 5 platform." Pro ASP. NET MVC 5 Platform. Apress, Berkeley, CA, 2014. ISBN: 1430265418
- Allen, K. Scott, et al. Professional ASP. NET MVC 5. Wrox Press, 2014. ISBN: 1118794753
- MVC Framework - First Application (tutorialspoint.com)
- Application State in ASP.Net (meeraacademy.com)

*****

8

AJAX

Unit structure

8.0 OBJECTIVE

In this chapter we will learned:

1. What is AJAX? Why we need AJAX.
2. How we can create AJAX application.
3. Controls of the AJAX
4. Testing an ASP.NET Ajax application
5. Global.asax and Web Config

8.1 WHAT IS AJAX?

In a simple word ajax means asynchronous JavaScript and XML which used to create communication between client and server without require postback . The benefit of this faster response to user It is client-side script. The MVC Framework contains built-in support for unobtrusive Ajax. We can use helper methods to define features. The features are based on JQuery features.

The desktop and web application have difference i.e. stateless behaviour of web application.

To enable the unobtrusive AJAX support in the MVC application, open the Web.Config file and set the UnobtrusiveJavaScriptEnabled property inside the appSettings section using the following code. If the key is already present in your application, you can ignore this step.

<add key= “UnobtrusiveJavaScriptEnabled” value=”true”/>

After this, just open common layout file _Layout.cshtml. we will add JQuery.

<script src=”~/Scripts/Jquery-ui-1.8.24.min.js” type=”text/javascript”>
</script>
< script src=”~/Scripts/jquery.unobtrusive-ajax.min.js” type=”text/javascript”>
</script>

8.2 APPLICATION WITH UNOBTRUSIVE AJAX

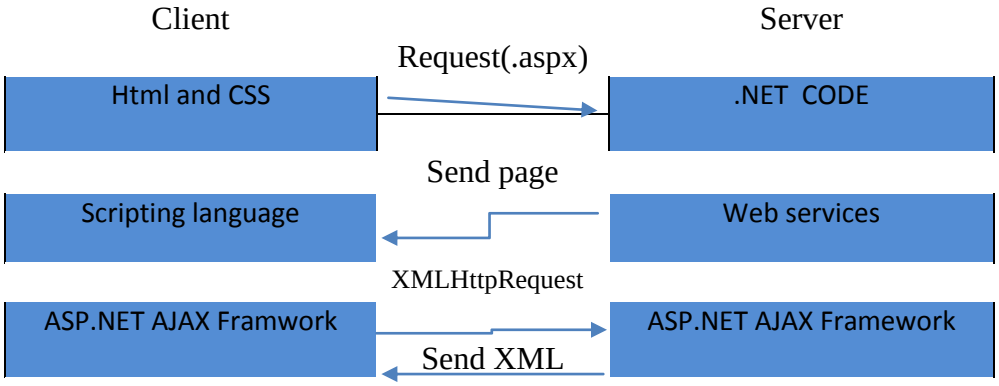
1. First create model file:

```
using System;
namespace MVCAjaxSupportExample.Models

{
Public class User
        {
          Public int UserId{ get;set;}
          Public string firstName{get;set}
          Public string LastName{get;set}
          Public D ateTime Birthdate{ get; set}
          Pubic Role Role { get, set}
        }
        Public enum Role {
                                Admin
                                Employee
                                Guest
                                 }
                }
```

Component of Ajax are following:

1. Ajax Control toolkit
2. XMLHHTTPRequest object in JavaScript
3. Using Jquery AJAX
4. Microsoft Ajax Library
5. Ajax server Control

How the Ajax works?:



8.3 AJAX CONTROLS

1. Update Panel Control
2. ScriptManager Control
3. ScriptManagerProxy Control
4. UpdateProgress Control
5. Timer

1. UpdatePanel:

The UpdatePanel control specifies a part of a Web page which can be updated or refreshed partially based on the update condition. Refreshing a specific part of a Web page is referred as partial-page update. You can also add one or more UpdatePanel control in the Web page. The UpdatePanel control uses the AJAX library to support the partial-page rendering.

It is a container control that contain the all controls and that control define the region or branch which is capable of making page updates.

For example:

<form Id=”form1” runat=”server”>
<asp:ScriptManager ID=”MainScriptManager” runat=”server”/>
<asp:UpdatePanel ID=”pnlWelcome” Text=”Click the button”/>
<ContentTemplate>
<asp:Lable runat=”server” ID=”btnWelcome” OnClick=”btnwlcome_Click” Text=”Upadate lable!” />
</ContentTemplate>
</asp:UpdatePanel>
</form>

.

2. ScriptManager Control:

The ScriptManager manages all ASP.NET AJAX assets on an internet web page. It renders the AJAX library to the browser and helps partial page rendering. It additionally manages partial page uploads and provide get right of entry to internet service method. Without script supervisor we can't use AJAX controls at the web page.

Syntax: <asp:ScriptManager ID=”ScriptManager1 runat=”server”/>

3. ScriptManagerProxy control:

A web page will have handiest one ScriptManager control. If your software has a Master-Content page state of affairs and the MasterPage has a ScriptManager control, then you may use the ScriptManagerProxy control to feature distinctive scripts to the content material pages.

Sometimes, there can be the case when you aren't required to apply AJAX on every and each net page by way of using the ScriptManagerProxy control then you can add AJAX functionalities to precise net pages. Since, ScriptManager at the Master Page will load the AJAX library on every web page that derives from the MasterPage, although they're now not wished, which would lead to a waste of resources.

4. Update Progress control:

This is used to display the progress of partial-page updates. You can use one UpdateProgress control to represent the progress of partial-page updates for the entire page. Also, you can include an UpdateProgress control for every UpdatePanel control. You can customize the default layout and content of the UpdateProgress control to make it more attractive.

It used during the server asynchronous processing. During the partial-page update to complete the use of this is to provide information in the graphics or Text that is displayed. Content inside this element can be an image.text or similar HTML content.

Example:

<asp:Update ID=”UpdatePanel1” runat=”server”>

<Content Template>

<asp:GridView ID=”GridView1” runat=”server”>

</asp:GridView>

<asp:UpdateProgress ID=”UpdateProgress1” runat=”server”>

<ProgressTemplate>

Processing…..

</ProgressTemplate>

</asp:updateProgress>

</ContentTemplate>

</asp:UpdatePanel>

5. Timer:

Timer Control is used to perform postbacks at defined time intervals. You can also use the Timer control with an UpdatePanel control to enable partial-page updates at a defined interval. You can also use the Timer control to post the entire page.

Timer is an Ajax control that can be used to update portion of page on periodic, time basis. You can use if you want to update an image such as on webpage advertisement or stock ticker/news ticker. It is add the code a timer control directly to an UpdatePanel control.

Example:

<asp:Update ID=”UpdatePanel1” runat=”server”>

<Content Template>

<asp:Image ID=”Image1” runat=”server” ImageUrl=”~/images/contoso.png”/>

<asp:Timer ID=”Timer1” runat=”server” Interval=”7000” ontick=”timer1_Tick”>

</asp:Tomer>

</ContentTemplate>

</asp:UpdatePanel>

8.4 TESTING AN ASP.NET AJAX APPLICATION

After you install the Module for ASP.NET Testing on a pc running WAPT or WAPT Pro Workplace, you may be in a position to test ASP.NET web sites containing, especially, ASP.NET AJAX UpdatePanel Controls.

ASP requests normally incorporate __VIEWSTATE and __EVENTVALIDATION parameters which need to be parameterized. There are 2 approaches to switch these parameters in server response:

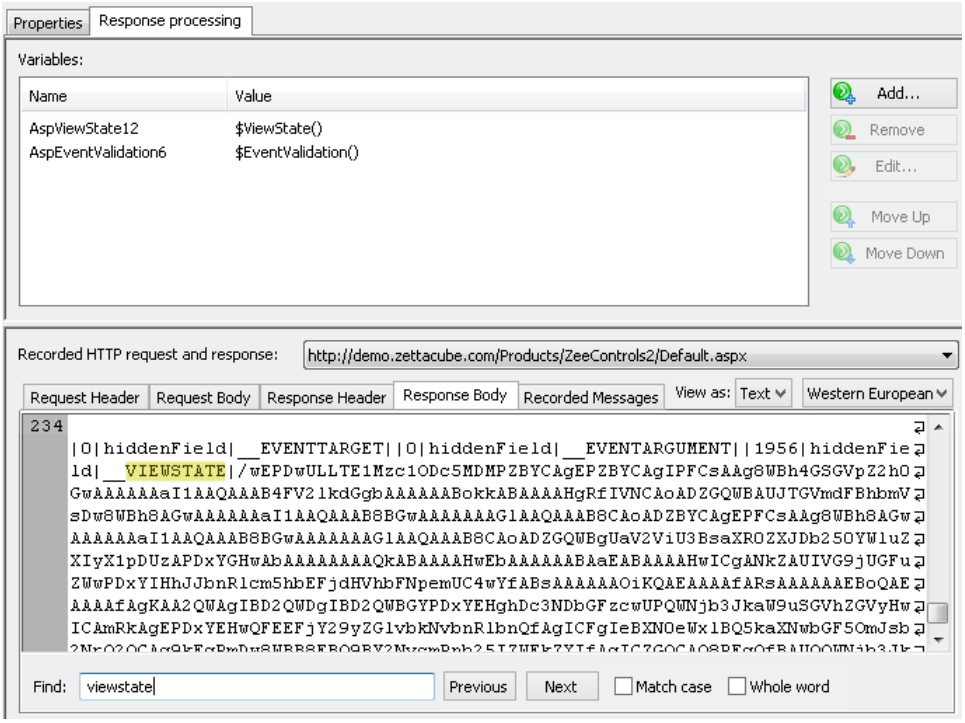
1. In the hidden field of the html code of response, or in the following way:

Reference: Testing of ASP.NET Applications (loadtestingtool.com)

It is just the case of ASP.NET AJAX UpdatePanel Controls usage. In both cases, the Module for ASP.NET Testing will extract __VIEWSTATE and __EVENTVALIDATION parameters from responses and correctly parameterize them. This is done automatically by the module, so you will not need to do any additional work and parameterize ASP requests manually.

When the module finds a new occurrence of __VIEWSTATE (__EVENTVALIDATION) during recording, it automatically creates AspViewState (AspEventValidation) variable calculated

using $ViewState ($EventValidation) function, for example:



Reference: Testing of ASP.NET Applications (loadtestingtool.com)

The module detects from which response the current __VIEWSTATE (__EVENTVALIDATION) was extracted, and substitutes the necessary variable to the corresponding request which uses that __VIEWSTATE (__EVENTVALIDATION). So, the Module for ASP.NET Testing automatically finds __VIEWSTATE and __EVENTVALIDATION parameters in server responses and parameterizes them, so that correct __VIEWSTATEs (__EVENTVALIDATIONs) are extracted from the necessary responses and transferred to the corresponding requests.

If this module is not installed and you record ASP requests, then at the end of recording process you may receive a warning message. It notifies you that some of requests contain ASP, so they will not be parameterized because the module is not installed.

Note that the Module for ASP.NET Testing is optional. It facilitates the design of your profiles and reduces the amount of manual work required to test an ASP.NET site. If you are not sure about the necessity to use it in your case, you can try the following steps.

1. After recording a virtual user profile, verify it using the "Verify Test" button on the toolbar.

2. Take a look at the created logs. You can find them in the "Logs" folder in the left view. If you see errors occurred during the verification, especially 500 (internal server error), 501 or similar ones, it means that profiles have been parameterized incorrectly. In this case you can either install the Module for ASP.NET Testing, or try to parameterize your profiles manually using regular WAPT functions available without the module.

8.5 GLOBAL ASAX AND WEB CONFIGU

The Global.asax, also known as the ASP.NET application file, is located in the root directory of an ASP.NET application. This file contains code that is executed in response to application-level and session-level events raised by ASP.NET or by HTTP modules.

8.6 SUMMARY

From this chapter we have learned Ajax features in MVC. The key to success with Ajax in ASP.NET MVC 5 is in understanding jQuery and making jQuery work for you in your application. Not only is jQuery flexible and powerful, but it also allows you to separate your script code from your markup and write unobtrusive JavaScript. The separation means you can focus on writing better JavaScript code and embracing all the power jQuery has to offer.

8.7 UNIT END EXERCISES

1. Why we need Ajax in MVC?

2. What are the benefits in Ajax?

3. How does the work Ajax?

4. What are the controllers of Ajax?

5. Create one application in Ajax with JQuery.

8.8 LIST OF REFERENCES

- Freeman, Adam. Pro asp. netmvc 5 platform." Pro ASP. NET MVC 5 Platform. Apress, Berkeley, CA, 2014. ISBN: 1430265418

- Allen, K. Scott, et al. Professional ASP. NET MVC 5. Wrox Press, 2014. ISBN: 1118794753
- MVC Framework - First Application (tutorialspoint.com)
- Application State in ASP.Net (meeraacademy.com)
- Testing of ASP.NET Applications (loadtestingtool.com)

*****

MODULE V

9

WEB SERVICES AND WCF

Unit Structure

9.0 OBJECTIVE

From this chapter you under:

- What is mean by Web Service.
- What web service can do and cannot do.
- Identity the key components or technologies involved in web services.
- Identify who manages the web services specification.
- Examine the examples of the web services.
- Impart understanding of Web Techniques and Design Web Services.

9.1 INTRODUCTION

Web services developed by Microsoft in June 2000, Web services as a key component of its .Net initiative, a broad vision for embracing the Internet in the development, engineering, and use of software. Web Services is nothing but software program by using this you can communicating your web application. Web services are web application components. Web Service means to interact with the objects over the internet. It is functionality use in different platforms using protocols.

- It does not dependants on any language you can we can write web services in any language and access the same using any other language.
- It is protocol independent.
- It is platform independent.
- It is self-contained and self-describing.
- It is stateless architecture.
- HTTP and XML open and text-based are the basis for web services.

9.2 WEB SERVICE

Where we can use web service:

- In **XML (Extensible Markup Language)** web service specifies only the data understanding the application regarding programming language.
- By using **SOAP (Simple Object Access Protocol)** communication between the Services and the application done.
- In **WSDL (Web Services Description Languages)** a unique method that is helpful to specify the web services to the other programs.
- With the help of **UDDI (University Description, Discovery, and Integration)** can search web service registries.

1. XML stands for extensible Markup Language.
2. It was designed to store and transferred the data.
3. It was designed for machine as well as human readable.
4. It was self-descriptive.
5. It stores data in plain text format.
6. It also useful to expand and upgrade to new operating system.

XML webservices are registered so that potential users can find them easily. We can do with the Universal Discovery Description and Integration (UDDI).

XML Declaration

<?xml version="1.0" encoding="UTF-8"?>

Syntax:

1. It is case sensitive.
2. first statement must be XML document

3. HTTP protocol is used for overriding the value of encoding.

Whenever we use web services there are two ways to use web services:

1. Reusable application components.
2. Connect the existing software.

9.2.1 Advantages of Web Services:

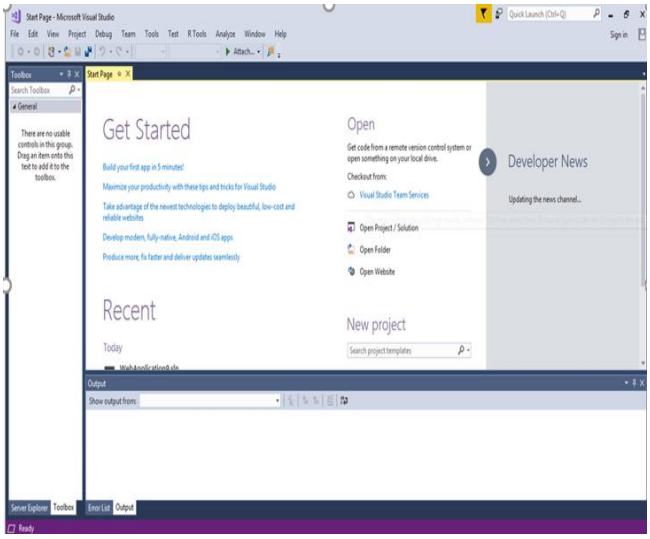
1. It always uses the open, textual content-based totally preferred. net service makes use of all the ones components even they are written in diverse languages for extraordinary platforms.
2. It provides promotes the modular technique of the programming in order that the more than one organization can talk with the identical net service.
3. Internet offerings are clean to put in force however high-priced because they use the existing infrastructure, and we will repackage most of the packages because the web service.
4. It reduces the price of business enterprise application integration and Business to business communications.
5. Web offerings are the interoperable enterprise that carries the 100 carriers and sell them interoperability.

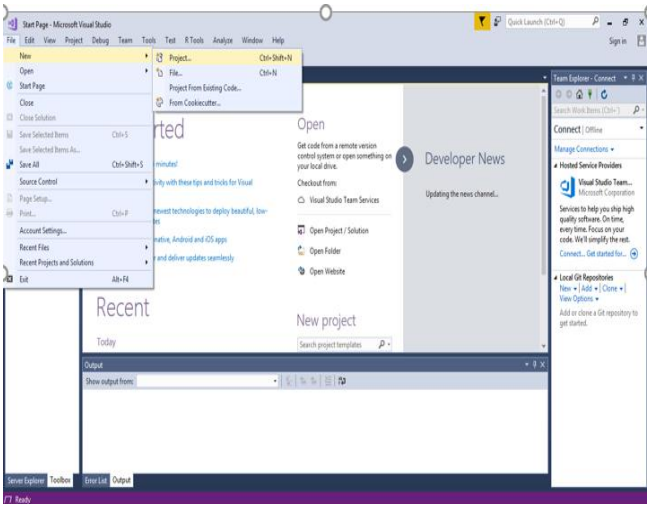
9.2.2 Disadvantage of Web Services:

1. One of the limitations of Web Services is that SOAP, WSDL, UDDI needs to be upgraded.
2. Support for collaboration is the limit of Web Service.
3. Web service limit is also a fee.
4. If we want to use web services in a high-performance environment, in that case, the web service will slow down.
5. Use of the web service increases traffic to the network.
6. The level of security of Web Services is low.
7. We use a standard procedure to describe the quality of certain web services.
8. The Web Service level is in draft form.
9. Maintaining intellectual property is a vendor and is the limit of web service.

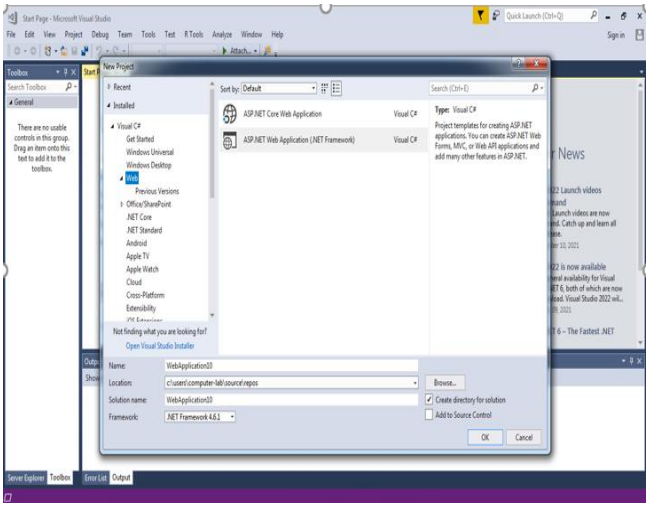
9.2.3 Examples of web services:

Step 1: Open visual studio (version -2017) in menu create new project.

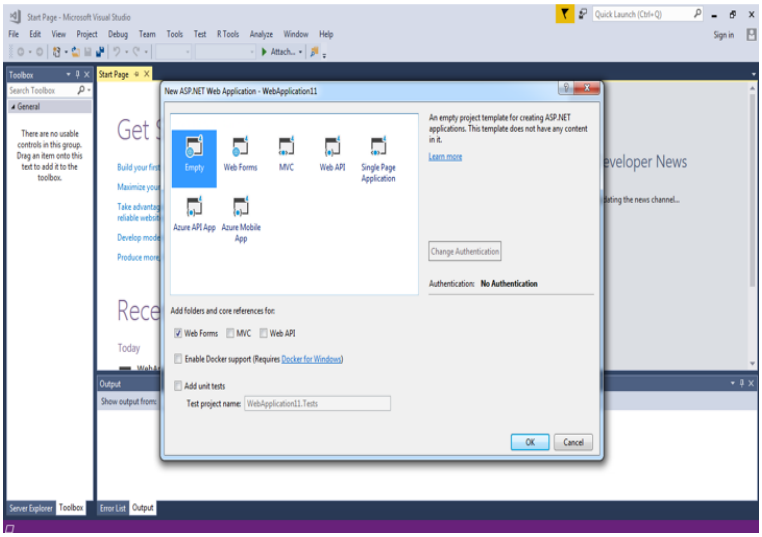




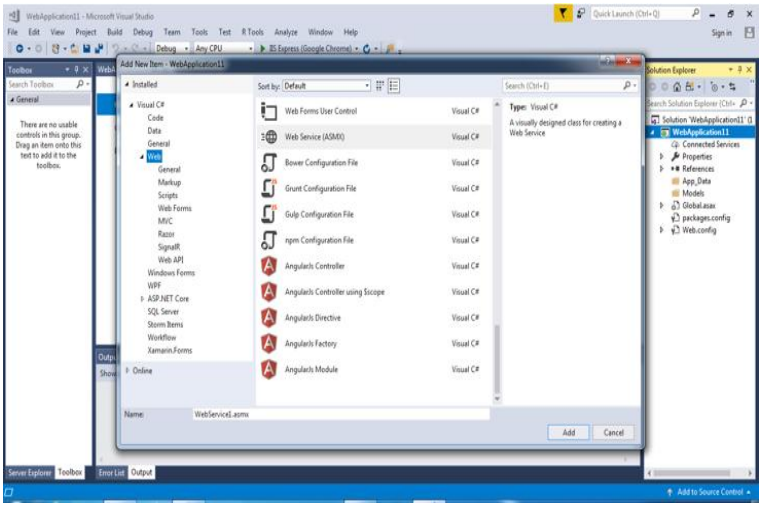
Step 2: New windows appears then you have to select web in left part and in the right parts you have to select ASP .Net Web Application (.Net Framework)

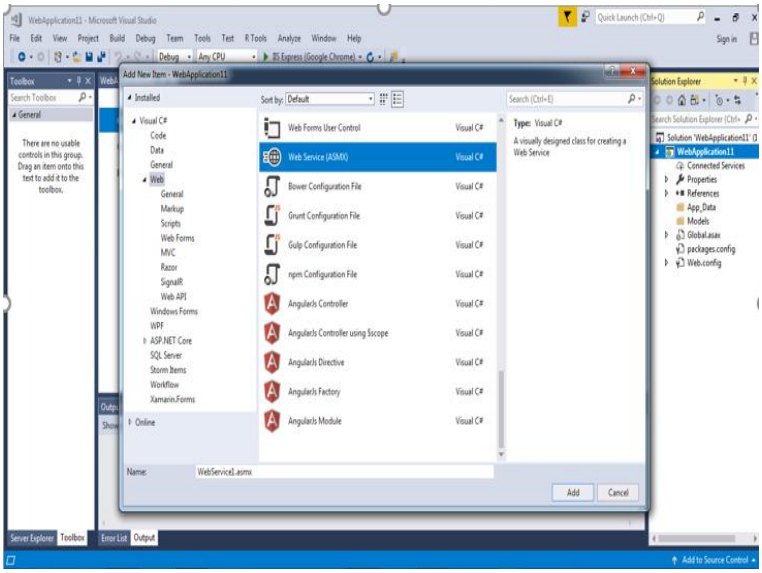


Step 3: Select Web Application as Empty and web form.

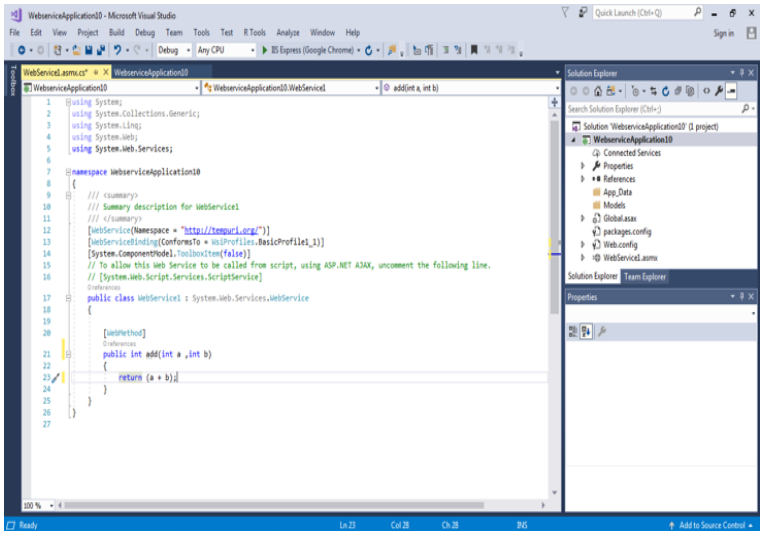


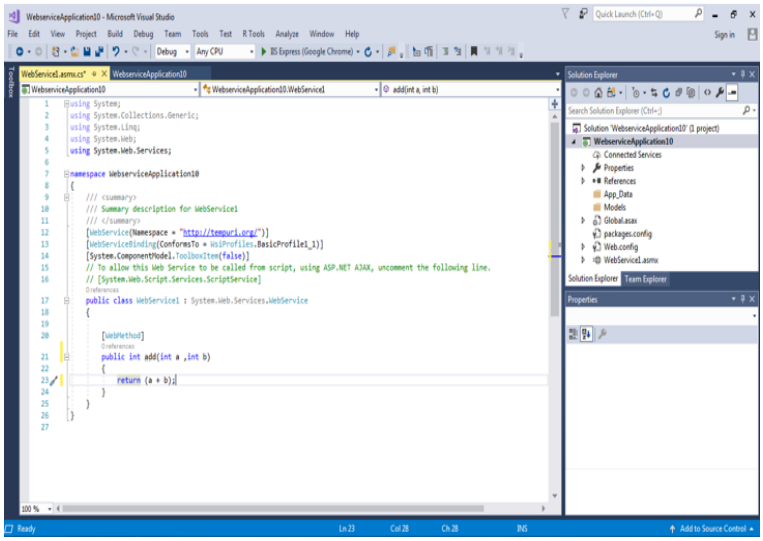
Step 4: Right side windows you can see the name of Web applications select add the new items.



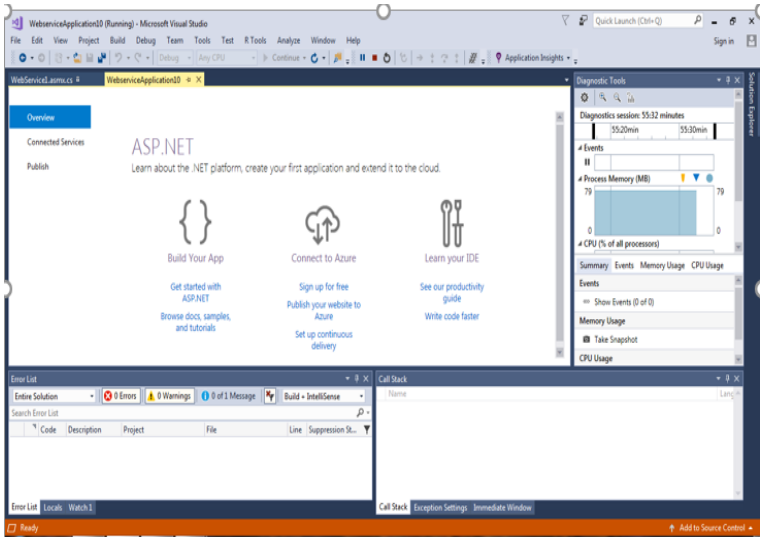


Step 5: We have created one method that is add ()

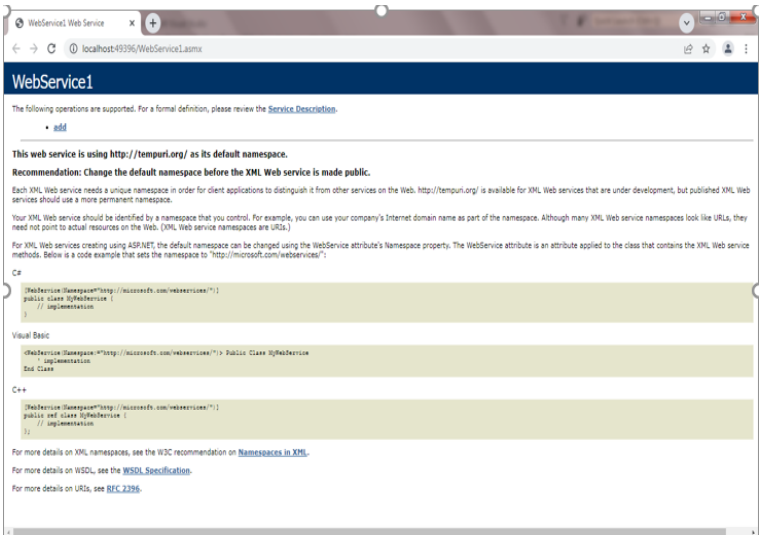




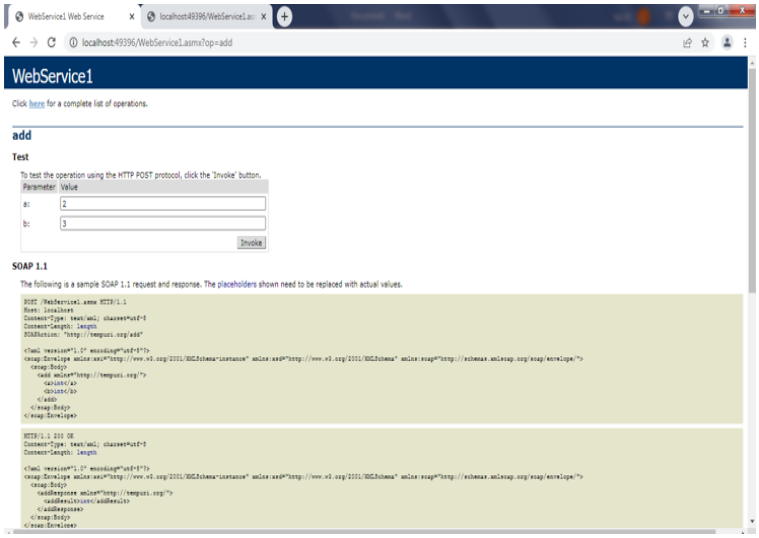
Step 6: Run the method add () by clicking IIS Express.



Step 7: You can see method add ()



Step 8: When you click on add () method, you have to put the values of parameters.



Step 9: You can see output.

9.2.4 Creating and Consuming an XML WCF Service-Simple and Database:

WCF means Windows Communication Foundation. It is a framework for building, configuring, and deploying network-distributed services. Firstly, its Indigo, which enables hosting services in any type of operating system process. WCF is a service-oriented application. Using WCF we can build connected secure, reliable, transacted solutions that integrate across platforms.

It is a framework for building service-oriented applications. Data can be sent as asynchronous messages from one service to another.

A client is an application which uses WCF client to communicate with services or other clients.

To create client application:

- Get the service contract, bindings, and address information for service endpoint: -this can be done by service Model Metadata Utility.
- Create a WCF client using that information: -WCF runtime converts methods into messages send it to service, collect reply from service for that message return value to that client.
- Call operations by using try and catch block
- Close the WCF objects.so that new client object gets a chance to connect

The mainly reasons to use WCF are Distributed Application and Interoperable

Distributed Application means do not run-on single system but multiple systems which are connected over networks.

Interoperable means that an application can consume or connect with another application, but it does not matter in which platform it is developed.

Which consist of three parts

- **WCF Service:** What it is providing.
- **WCF Service host:** Where is it.
- **Service Client:** Who is the client of the Service.

WCF Hosting:

Web services that was hosted inside web server such as IIS using http or Https protocols.

1. IIS

2. WAS (Windows process activation services)
3. Self-hosting
4. Windows service

9.2.5 WCF Features:

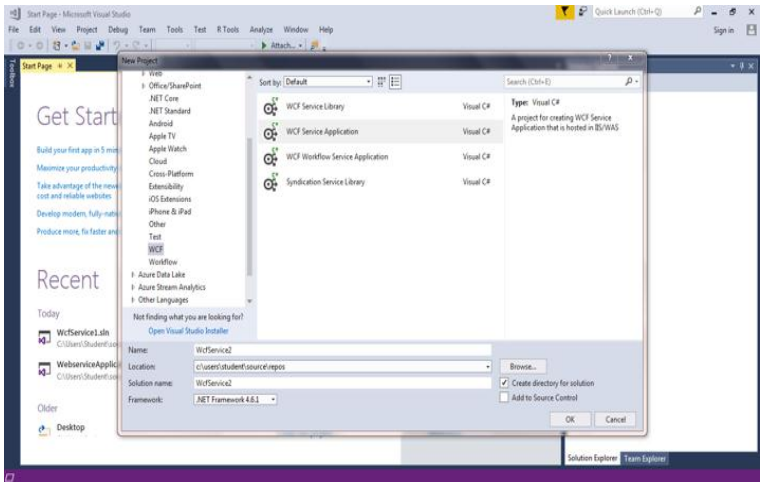
1. **Service Orientation:** It uses Service-oriented architecture (SOA) on which web services uses send and receive data.
2. **Interoperability:** It uses more industry stands for it.
3. **Multiple Message Patterns:** Different patterns are used like request/replay pattern, one way message, duplex exchange pattern
4. **Service Metadata:** It is used to automatically generate and configure clients for accessing WCF services. And published over HTTP and HTTPS.
5. **Security:** Message can be encrypted to protect privacy.
6. **Multiple Transports and Encodings:** whenever Messages can be sent on uses in transport protocols and encodings like HTTP.
7. **Reliable and Queued Messages and Durable:** Reliable message uses reliable session and durable message pattern always saved to the database.
8. **Transactions:** WS-Atomic Transactions, Microsoft Distributed Transaction coordinator and API Transaction
9. **AJAX and REST Support:** supported XML formats
10. **Extensibility:** to customize the behaviour of a service

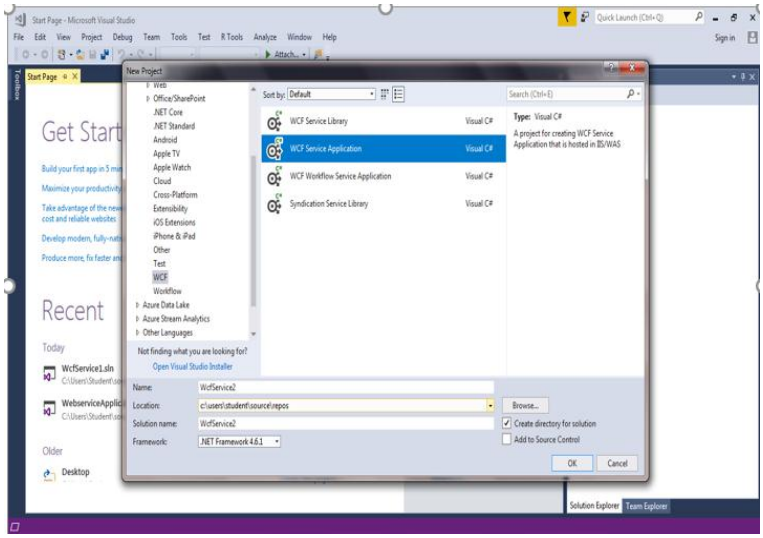
9.2.6 Advantages of WCF:

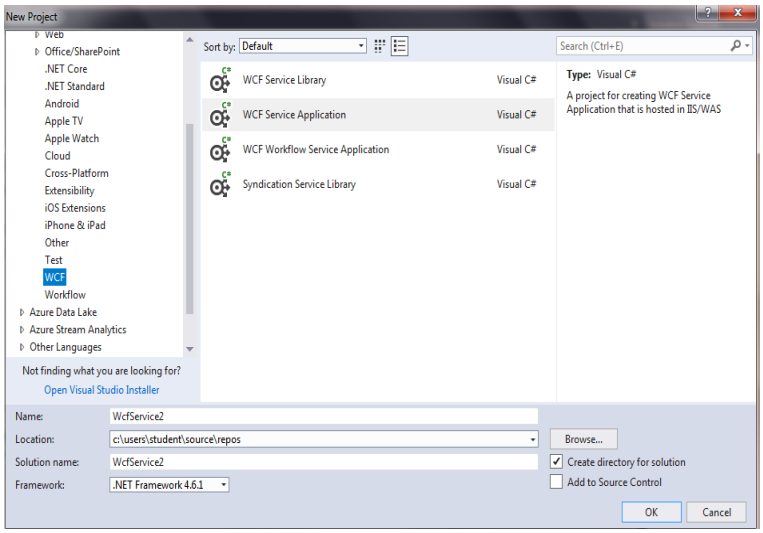
1. It is interoperable with other services when compared to .Net Remoting where the client and service must be .Net.
2. It services provide security and reliability in compared to ASMX web services.
3. In WCF, there is no need to make much change in code for implementing the security model and changing the binding. Instead, it requires small changes in the configuration.
4. WCF has integrated logging mechanism with this changing the configuration file settings will provide this functionality. In other technology developer must write the code.

9.2.7 Examples of WCF Service:

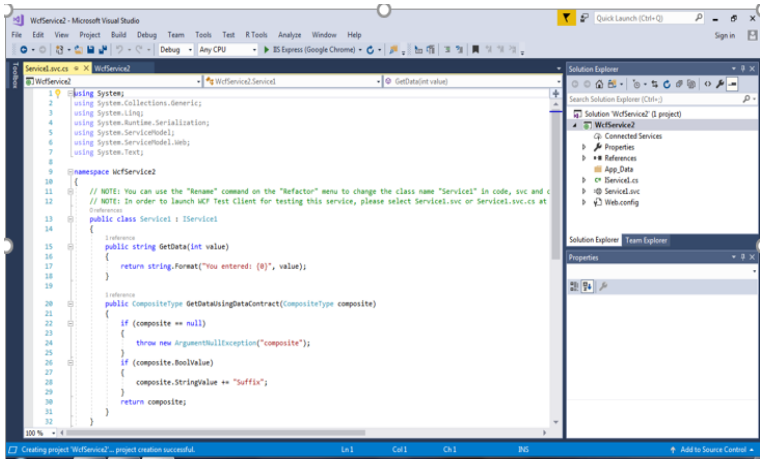
Step 1: For WCF service while creating new project under visual C# select WCF (Window-1) and choose WCF service application as we must apply WCF service (Window-2) then press ok with your WCF service name.





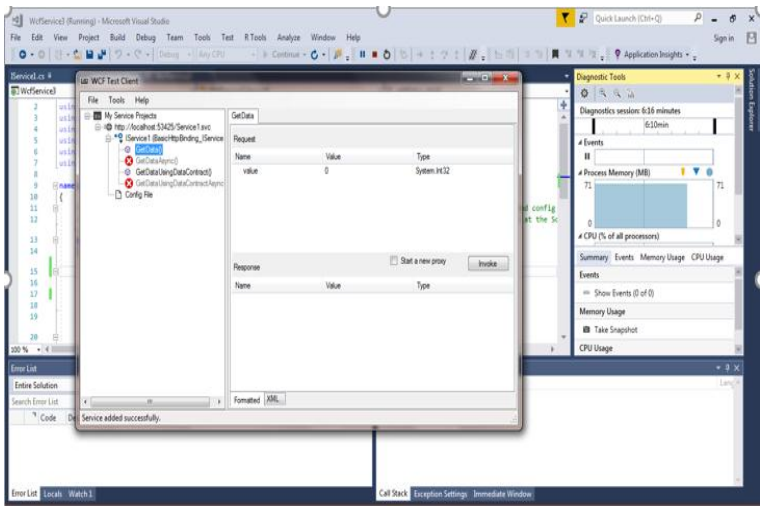


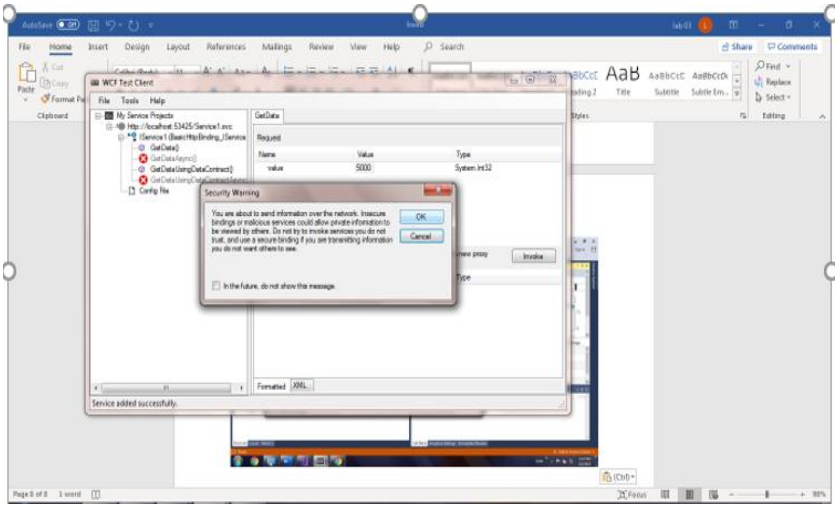
Step 2: Check interface in solution Explorer that is (IService.cs). Here we have GetData() Method so we have to make entry in interface service after that execute your program by clicking on IIS Express (Internet Explorer)

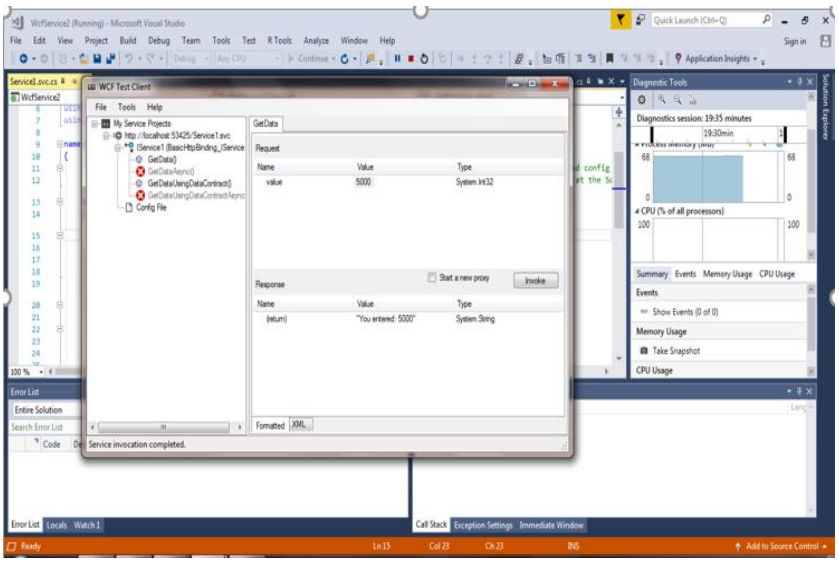


```
using System.Runtime.Serialization;
using System.ServiceModel;
using System.ServiceModel.Web;
using System.Text;

namespace WcfService1
{
    // NOTE: You can use the "Rename" command on the "Refactor" menu to change the interface name "IService1"
    [ServiceContract]
    1 reference
    public interface IService1
    {

        [OperationContract]
        0 references
        string GetData(int value);

        [OperationContract]
        1 reference
        CompositeType GetDataUsingDataContract(CompositeType composite);

        // TODO: Add your service operations here
    }
```

Step 3: After Execution we get new window where we have to double click on our methods that is GetData () and give the values to the parameter which is right side of the window and press the button invoke. It shows the information of sender and receiver.





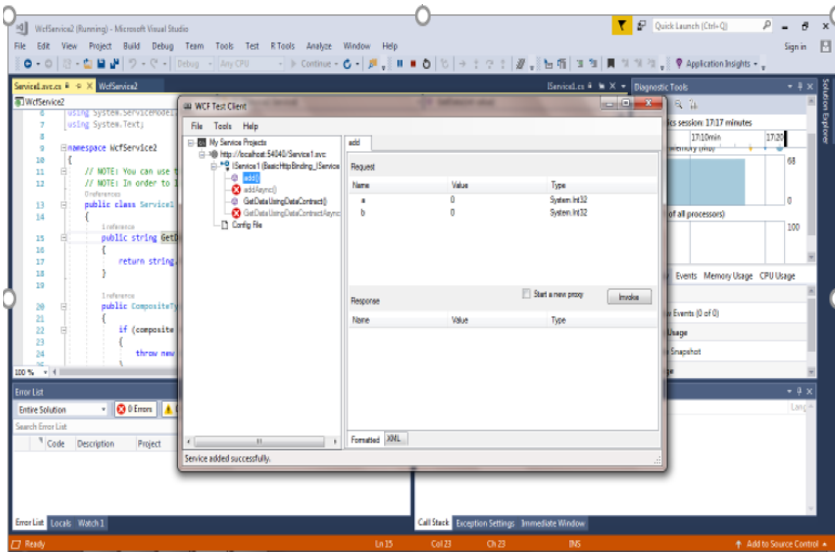


Step 4: We have created add() method

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.ServiceModel.Web;
using System.Text;

namespace WcfService1
{
    // NOTE: You can use the "Rename" command on the "Refactor" menu to change the class name "Service1" in co
    // NOTE: In order to launch WCF Test Client for testing this service, please select Service1.svc or Servic
    0 references
    public class Service1 : IService1
    {
        0 references
        public int add(int a,int b)
        {
            return (a+b);
        }

        1 reference
        public CompositeType GetDataUsingDataContract(CompositeType composite)
        {
            if (composite == null)
```

```
using System.Runtime.Serialization;
using System.ServiceModel;
using System.ServiceModel.Web;
using System.Text;

namespace WcfService1
{
    // NOTE: You can use the "Rename" command on the "Refactor" menu to change the interface name "IService1"
    [ServiceContract]
    1 reference
    public interface IService1
    {

        [OperationContract]
        1 reference
        public int add(int a, int b);

        [OperationContract]
        1 reference
        CompositeType GetDataUsingDataContract(CompositeType composite);

        // TODO: Add your service operations here
    }
```

Step 5: Execute the program and put the values of a and b and press the button invoke. It display addition of two numbers.



9.3 SUMMARY

Web Services is nothing but software program by using this you can communicating your web application. Web services are web application components. Web service types of Reusable application components XML (Extensible Markup Language), SOAP (Simple Object Access Protocol), WSDL (Web Services Description Languages) and UDDI (University Description, Discovery, and Integration) WCF means Windows Communication Foundation. It is a framework for building, configuring, and deploying network-distributed services. We can apply web service and WCF services.

Self-Learning Topics:

Caching Web service responses

9.4 UNIT END EXERCISES

1. What is web service. Explain Example of web service.
2. Explain WCF in details.
3. Explain Advantages of Web Service.
4. Explain Features of WCF.
5. Explain how to start WCF with example.
6. Design Web Application to produce and Consume a WCF Service.
7. Design Web Application to produce and consume a web Service.

9.5 LIST OF REFERENCES

1) https://www.javatpoint.com/web-services-in-c-sharp
2) https://www.aspsnippets.com/Articles/Simple-database-Web-Service-Tutorial-in-ASPNet-with-Example.aspx
3) Spaanjaars, Imar. Beginning ASP. NET 4.5. 1: in C# and VB. John Wiley & Sons, 2014. ISBN: 1861009038
4) MacDonald, Matthew. ASP. NET: The Complete Reference. McGraw-Hill, Inc., 2002. ISBN:
5) https://www.w3schools.com/asp/default.ASP0072125764
6) Spaanjaars, Imar. Beginning ASP. NET 4.5. 1: in C# and VB. John Wiley & Sons, 2014. ISBN: 1861009038
7) Allen, K. Scott, et al. Professional ASP. NET MVC 5. Wrox Press, 2014. ISBN: 1118794753
8) Schildt, Herbert. C# 4.0: the complete reference. Tata McGraw-Hill Education, 2010

*****

MODULE VI

10

DESIGNING MVC APPLICATION

Unit Structure

10.0 OBJECTIVE

In this chapter we will learn how to create and design MVC application using multiple patterns, methods and expressions.

Why we need it?

10.1 INTRODUCTION

In the ultimate couple of years, I was simply randomly going via a questionnaire section and located out there are numerous questions on MVC, especially on the novice stage, or beginner degree to ASP.NET you can say. There are many queries approximately which is first-rate, webforms or MVC, and if MVC, which version. As there are a few great articles which are associated with MVC on C# Corner, this newsletter will brief you about MVC, variations, a pattern software, and solution the question: Why MVC?

Today in case you do marketplace research the decision for for MVC is massive and it's far growing in numbers. Each and each company these days desires a MVC developer due to the fact every time a modern-day mission for development comes, they certainly need to broaden their product in MVC. MVC Version 5 has been launched with new abilities and the popularity of MVC is developing each day.

If your present-day assignment is in webforms and you find it flawlessly every day, you could say why MVC?

It is important distinguished between the MVC architecture pattern and the ASP.net MVC framework. The MVC.NET is old. It dates again to 1978 and the Smalltalk task at Xerox PARC—however it has won enormous reputation today as a pattern for Web programs, for the following reason:

1. User interaction with an MVC application follows a natural cycle: the user takes an action, and in response the application changes its data model and delivers an updated view to the user. And then the cycle repeats. This is a convenient fit for Web applications delivered as a series of HTTP requests and responses.

2. Web applications necessitate combining several technologies (databases, HTML, and executable code, for example), usually split into a set of tiers or layers. The patterns that arise from these combinations map naturally onto the concepts in MVC.

The ASP.NET MVC Framework implements the MVC pattern and, in doing so, provides greatly improved separation of concerns. In fact, ASP.NET MVC implements a modern variant of the MVC pattern that is especially suitable for Web applications.

MVC works as per following way of the diagram:

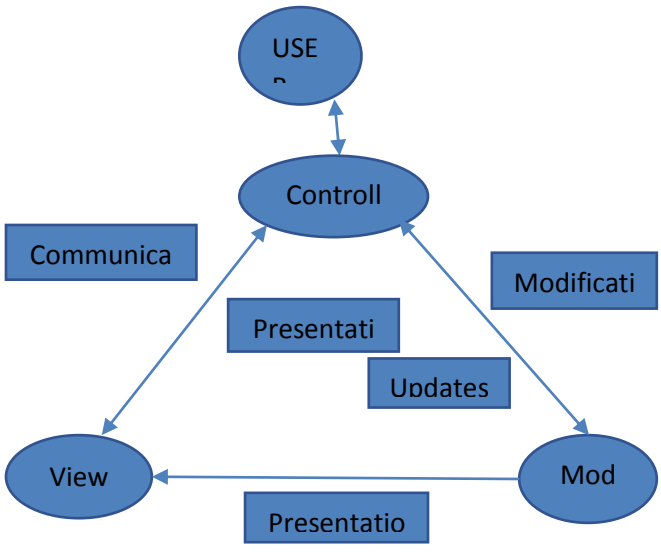


Figure 1

10.2 DESIGNING MVC APPLICATION

10.2.1 First application of MVC:

Visual Studio Express contains all of the features we need to create, test and deploy an MVC Framework application, but some of those features are hidden away until we ask for them. To enable all of the features, select Expert Settings from the Visual Studio Tools ------> Settings menu.

10.2.2 Creating New Asp.Net Mvc Project:

By creating new MVC framework project in Visual studio.

Select New Project from File menu then Open New Project dialog.

You can see in the following diagram, if we select the web templates in Visual C# section, we will see the Web Application Project template. Select this Project type.

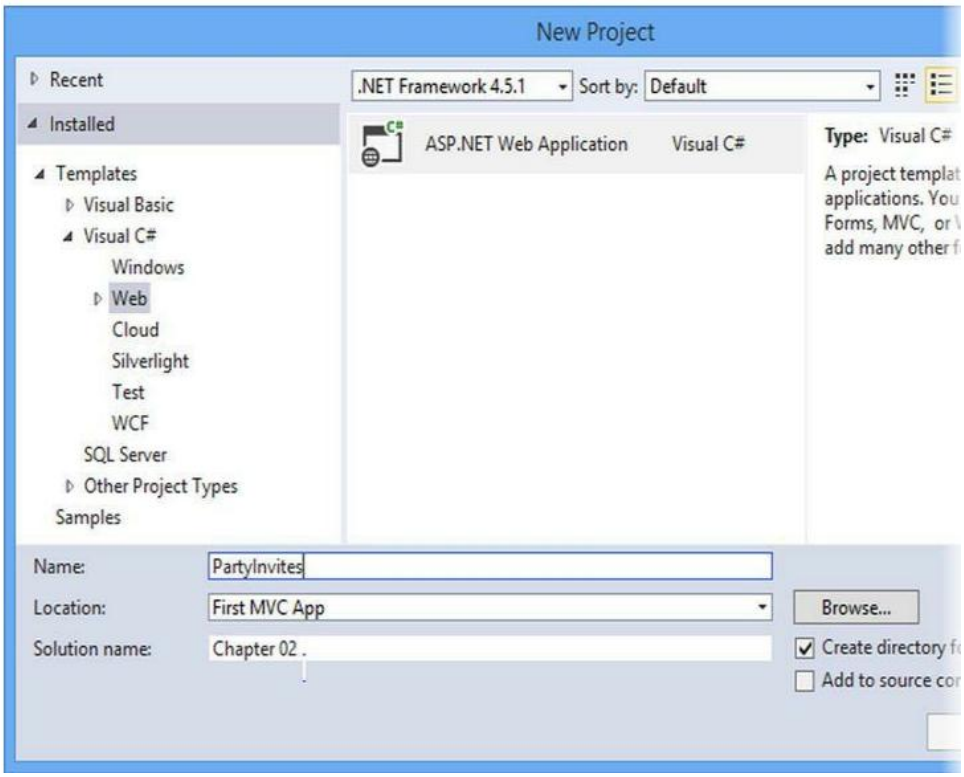


Figure 2 (ref. Pro ASP.NET MVC 5 Adam Freeman)

Give or set the Name of new project for example PartyInvites and click on Ok Button to continue.

which will ask us to set the initial content for the ASP.NET project Select Empty template Check MVC in the Add folders and core references section.

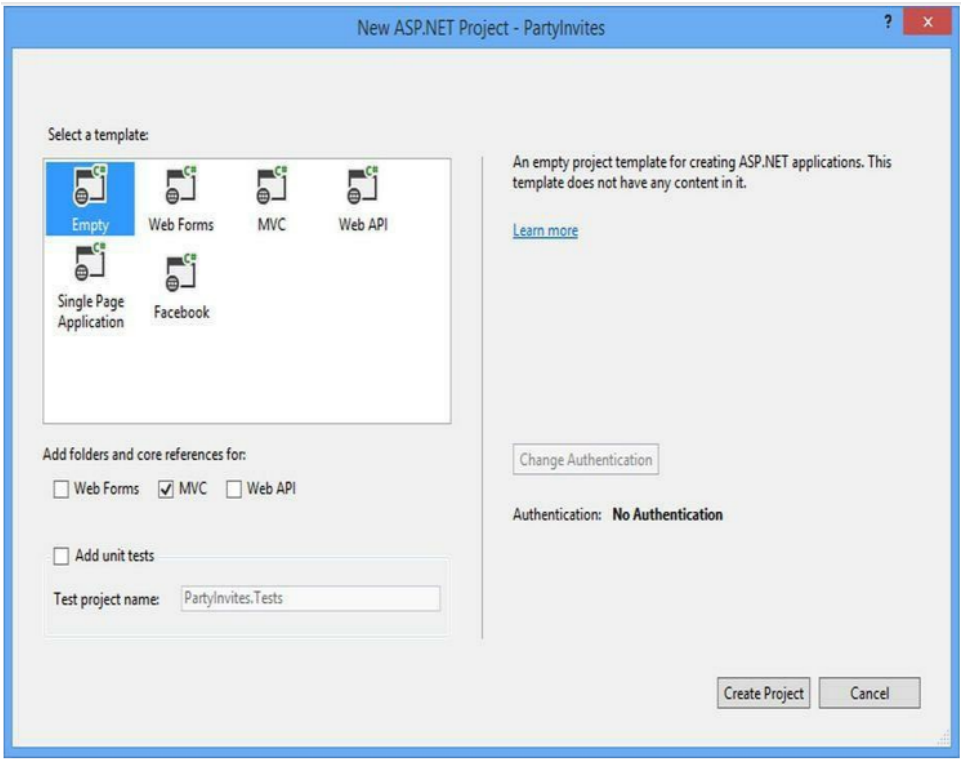


Fig 3 (ref. Pro ASP.NET MVC 5 Adam Freeman)

It will create a basic MVC project with minimal predefined content and it will be starting point which we can use for all of the examples in this chapter. Now need to click OK button to create new project.

Note: The other template alternatives/options are intended to provide you a extra complete starting point for your ASP.NET tasks.

Once the Visual studio creates the project, we will see the number of files folder in the Solution Explorer window.

Which is the default project structure for a new MVC project default and we can be understand the purpose purpose of each of the files and folders that Visual Studio creates.

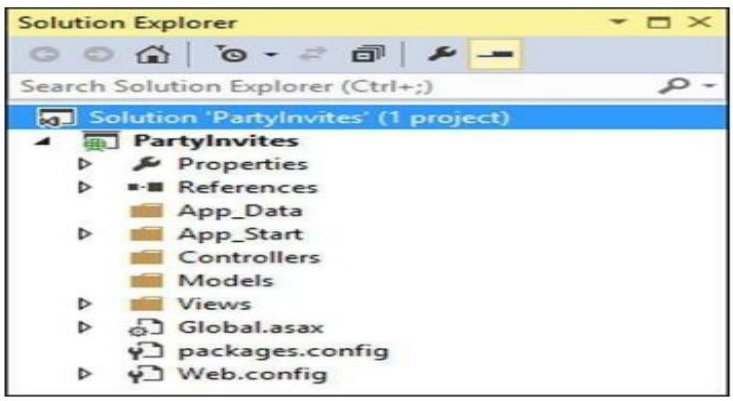


Fig 4 (ref. Pro ASP.NET MVC 5 Adam Freeman)

We can try to run the application now by selecting Start Debugging from the Debug menu.

We can see the result in below fig Because when we started with the empty project template, the application does not contain anything to run, so the server generates a 404 Not Found Error.

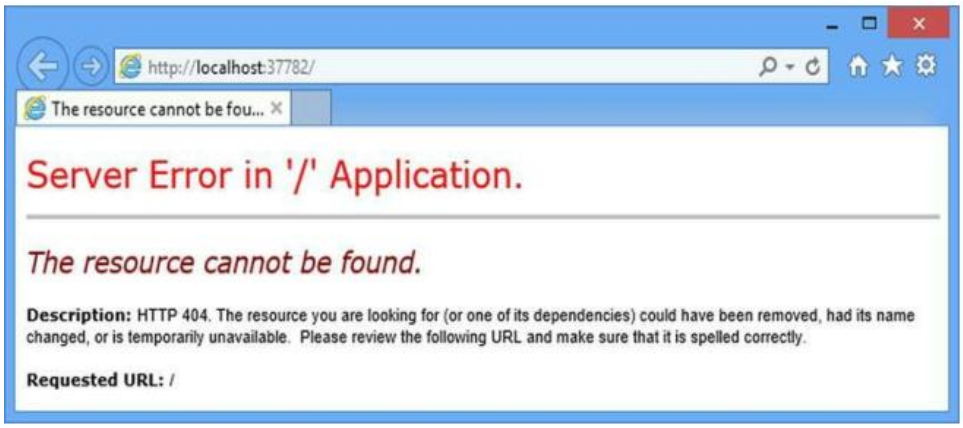


Fig 5 (ref. Pro ASP.NET MVC 5 Adam Freeman)

When we are finished, need to be sure to stop debugging by closing the browser window that shows the error, or by going back to

Visual Studio and selecting Stop Debugging from the Debug menu.

As we have just seen, Visual Studio opens the browser to display the project. The default browser is, of course, Internet.

Explorer, but you can select any browser that you have installed by using the toolbar shown in Figure 1.5. As the figure shows, we have a range of browsers installed, which we find useful for testing web apps during development.

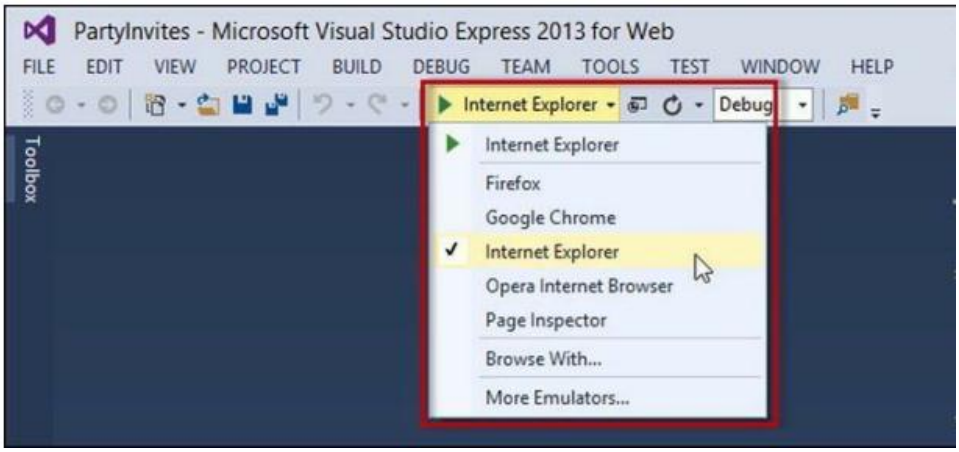


Fig 6 (ref. Pro ASP.NET MVC 5 Adam Freeman)

Now, will create the first controller in the application. Controllers are just simple C# classes, in which contains multiple public methods, we can call it action methods. now to add new controller, right click on the Controllers

folder in the project and select Add -□Controller. Please Name this Controller and click Add.

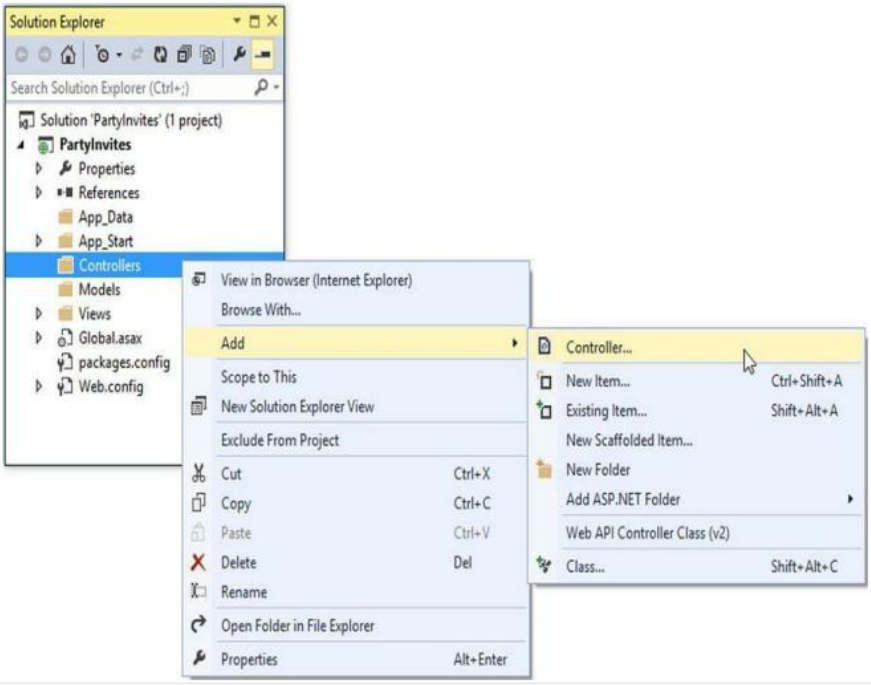


Fig 7 (ref. Pro ASP.NET MVC 5 Adam Freeman)

Whenever the Add Scaffold dialog appears, select the MVC 5 Controller-Empty Option as following fig 1.7 and Click the Add Button.

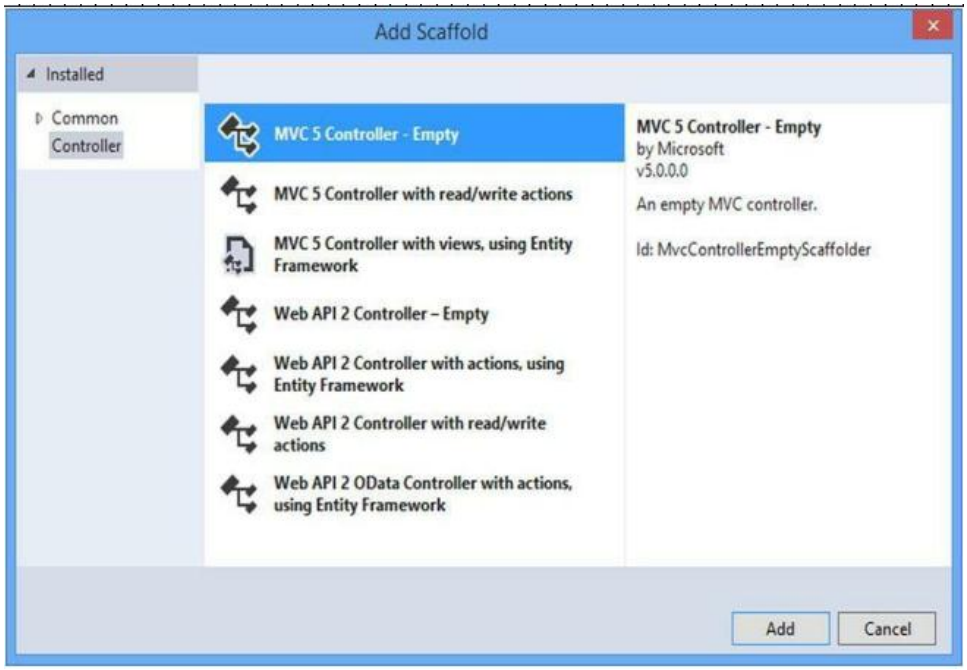


Fig 8 (ref. Pro ASP.NET MVC 5 Adam Freeman)

The Add Controller dialog will appear. Set the name to HomeController and click the Add button. There are several conventions represented in this

name: names given to controllers should indicate their purpose; the default controller is called Home and controller names have the suffix Controller.

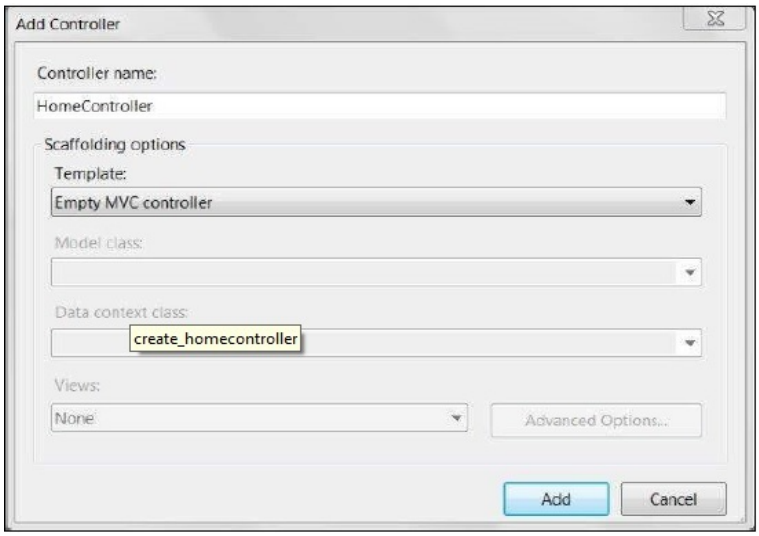


Fig 9 (ref. Pro ASP.NET MVC 5 Adam Freeman)

Visual Studio will create a new C# file in the Controllers folder called HomeController.cs and open it for editing. I have listed the default contents that Visual Studio puts into the class file in. You can see that the class is called HomeController and it is derived from the Controller class, which is found in the System.Web.Mvc namespace.

Listing 1

```
using system;
using system.collections.generic;
using system.Linq;
using system.web;
using system.web.Mvc;
        namespaces PartyInvites.Controllers
        {
            public class HomeController:Controller
                    {
Public ActionResult Index()
        {
          return View();
         }
    }
}
```

MVC is to make a couple of simple changes to the controller class. Edit the code in the

HomeController.cs file so that it matches Listing 2. We have highlighted the statements that have changed so they are easier to see.

Listing 2

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
namespace PartyInvites.Controllers
{
public class HomeController: Controller
{
public string Index()
{
return "Hello World";
}
}
}
```

We have changed the action method called Index so that it returns the string “Hello World”. Run the project again by selecting Start Debugging from the Visual Studio Debug menu. The browser will display the result of the Index action method, as shown in following fig 1.9

Figure 10(ref. Pro ASP.NET MVC 5 Adam Freeman)

10.3 CREATING SIMPLE DATA-ENTRY APPLICATION WITH VALIDATION:

Just Imagine Client has to host new year party and has to ask us to create web application that allows to invitees to electronic RSVP and ask following key features

1. A home page that shows information about the party
2. A form that can be used to RSVP

3. Validation for the RSVP form, which will display a thank-you page
4. RSVPs e-mailed to the party host when complete

In the following sections, we will build up the MVC project we created at the start of the chapter and add these features. We can check the first item off the list by applying what we covered earlier and add some HTML to my existing view to give details of the party. Listing 3.1 shows the additions we made to the Views/Home/Index.cshtml file.

Listing 3.1

```
@{
Layout = null;
}
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width" />
<title>Index</title>
</head>
<body>
<div>
@ViewBag.Greeting World (from the view)
<p>We're going to have an exciting party.<br/>
(To do: sell it better. Add pictures or something.)
</p>
</div>
</body>
</html>
```

We are on our way. If you run the application, you'll see the details of the party—well, the placeholder for the details, but you get the idea—as shown in following fig 11

Figure 11 (ref. Pro ASP.NET MVC 5 Adam Freeman)

10.3.1 Designing A Data Model:

In MVC, the M stands for version, and it's far the most essential a part of the software. The model is the illustration of the real global items, techniques, and policies that define the problem, called the area, of the utility. The version, frequently called a website model, consists of the C# objects (referred to as domain items) that make up the universe of the utility and the techniques that manage them. The perspectives and controllers expose the domain to the clients in a steady manner and a well-designed MVC software starts with a well-designed model, that's then the focus as controllers and perspectives are delivered.

We don’t need a complicated model for the PartyInvites application because it's miles this type of easy software and we need to create simply one domain class which I will call GuestResponse.

This object will be chargeable for storing, validating, and confirming an RSVP.

The MVC convention is that the classes that make up a model are placed inside the Models folder, which Visual Studio created as part of the initial project setup. Right-click Models in the Solution Explorer window and select Add followed by Class from the pop-up menus.

Set the file name to GuestResponse.cs and click the Add button to create the class. Edit the contents of the class to match Listing 3.2

If you don’t have a Class menu item, then you probably left the Visual Studio debugger running. Visual Studio restricts the changes you can make to a project while it is running the application.

Listing 3.2 The GuestResponse Domain Class Defined in the GuestResponse.cs File

```
namespace PartyInvites.Models
{
public class GuestResponse
{
public string Name {get; set;}
public string Email {get; set;}
public string Phone {get; set;}
public bool? WillAttend {get; set;}
}
}
```

10.3.2 Linking Action Method:

Now, the application goal is to include an RSVP form, so we need to add a link to it from my Index.cshtml view, as

shown in Listing 3.3.

Adding a Link to the RSVP Form in the Index.cshtml File

```
@{
Layout = null;
} <
!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width" />
<title>Index</title>
</head>
<body>
<div>
@ViewBag.Greeting World (from the view)
<p>We're going to have an exciting party.<br />
(To do: sell it better. Add pictures or something.)
</p>
@Html.ActionLink("RSVP Now", "RsvpForm")
</div>
</body>
</html>
```

Html.ActionLink is an HTML helper method. The MVC Framework comes with a collection of built-in helper methods that are convenient for rendering HTML links, text inputs, checkboxes, selections, and other kinds of content. The ActionLink method takes two parameters: the first is the text to display in the link, and the second is the action to perform when the user clicks the link. I explain the complete set of HTML helper methods in next point. We can see the link that the helper creates by starting the project, as shown in figure12

Figure 12 ref. Pro ASP.NET MVC 5 Adam Freeman)

10.3.3 Creating Action Method:

You will see a 404 Not Found error if you click the link. That's because we have not yet created the action method that corresponds to the /Home/RsvpForm URL. we do this by adding a method called RsvpForm to the HomeController class, as shown in Listing 3.4

Listing 3.4

Adding a New Action Method in the HomeController.cs File

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
namespace PartyInvites.Controllers
{
public class HomeController : Controller
{
public ViewResult Index()
{
int hour = DateTime.Now.Hour;
ViewBag.Greeting = hour < 12 ? "Good Morning" : "Good
Afternoon";
return View();
}
public ViewResult RsvpForm()
{
return View();
}
}
}
```

To add a view for the RsvpForm action method, but in a slightly different way—we need to create a stronglytyped view.

A strongly typed view is intended to render a specific domain type, and if I specify the type I want to work with

(GuestResponse in this case), MVC can create some helpful shortcuts to make it easier.

Right-click the RsvpForm method in the code editor and select Add View from the pop-up menu to open the AddView dialog window. Ensure that the View Name is set as RsvpForm, set Template to Empty and select GuestResponse from the drop-down list for the Model Class field. Leave the View Options boxes unchecked, as shown in figure 13

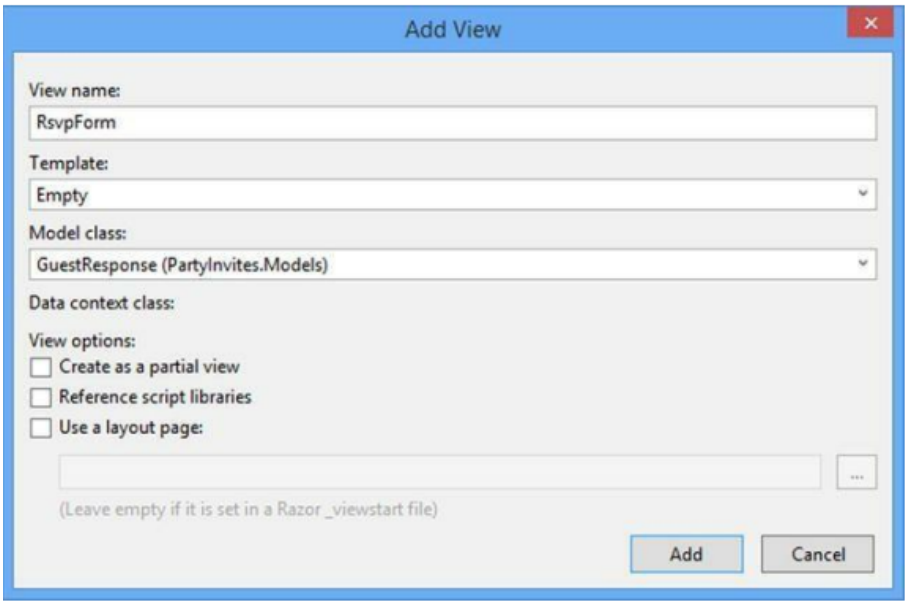


Figure 13 (ref. Pro ASP.NET MVC 5 Adam Freeman)

Click the Add button and Visual Studio will create a new file called RvspForm.cshtml in the Views/Home folder and open it for editing. We can see the initial contents in Listing 3.5.

This is another skeletal HTML file, but it contains a @modelRazor expression. As you will see in a moment, this is the key to a strongly typed view and the convenience it offers.

Listing 3.5.

The Initial Contents of the RsvpForm.cshtml File

```
@model PartyInvites.Models.GuestResponse
@{
Layout = null;
} <
!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width" />
<title>RsvpForm</title>
</head>
<body>
```

</body>
</html>

Building Form:

Now that we have created the strongly typed view, we can build out the contents of RsvpForm.cshtml to make it into an HTML form for editing GuestResponse objects, as shown in

Listing 3.6

Creating a Form View in the RsvpForm.cshtml File

@model PartyInvites.Models.GuestResponse
@{
Layout = null;
} <
!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width" />
<title>RsvpForm</title>
</head>
<body>
@using (Html.BeginForm()) {
Your name: @Html.TextBoxFor(x => x.Name)
Your email: @Html.TextBoxFor(x => x.Email)
Your phone: @Html.TextBoxFor(x => x.Phone)

47

Will you attend?

@Html.DropDownListFor(x => x.WillAttend, new[] {
new SelectListItem() {Text = "Yes, I'll be there",
Value = bool.TrueString},
new SelectListItem() {Text = "No, I can't come",
Value = bool.FalseString}
}, "Choose an option")

```
<input type="submit" value="Submit RSVP" />
}
</body>
</html>
```

For each property of the GuestResponse model class, I use an HTML helper method to render a suitable HTML input control. These methods let you select the property that the input element relates to using a lambda expression, like

this:

```
...
@Html.TextBoxFor(x => x.Phone)
...
```

The HTML TextBoxFor helper method generates the HTML for an input element, sets the type parameter to text, and sets the id and name attributes to Phone (the name of the selected domain class property) like this:

```
<input id="Phone" name="Phone" type="text" value="" />
```

This handy feature works because the RsvpForm view is strongly typed, and I have told MVC that GuestResponse is the type that I want to render with this view. This provides the HTML helper methods with the information they need to understand which data type I want to read properties from via the @model expression.

Don’t worry if you aren’t familiar with C# lambda expressions. I provide an overview in next topic, but an alternative to using lambda expressions is to refer to the name of the model type property as a string, like this:

```
...
@Html.TextBox("Email")
...
```

I find that the lambda expression technique prevents me from mistyping the name of the model type property, because Visual Studio IntelliSense pops up and lets me pick the property automatically, as shown in figure 14

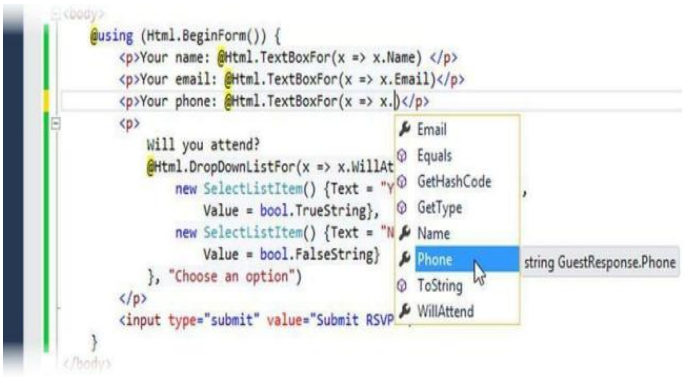


Figure 15 (ref. Pro ASP.NET MVC 5 Adam Freeman)

Setting starts to URL:

Visual Studio will, in an effort to be helpful, make the browser request a URL based on the view that is currently being edited. This is a hit-and-miss feature because it doesn't work when we are editing other kinds of file and because we can't just jump in at any point in most complex web apps.

To set a fixed URL for the browser to request, select PartyInvites Properties from the Visual StudioProject menu, select the Web section and check the Specific Page option in the Start Action category, as shown in

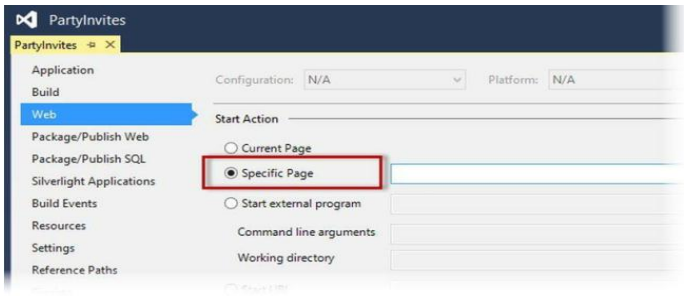


Figure 16 (ref. Pro ASP.NET MVC 5 Adam Freeman)

We can see the form in the RsvpForm view when you run the application and click the RSVP Now link. Figure 17 shows the result.

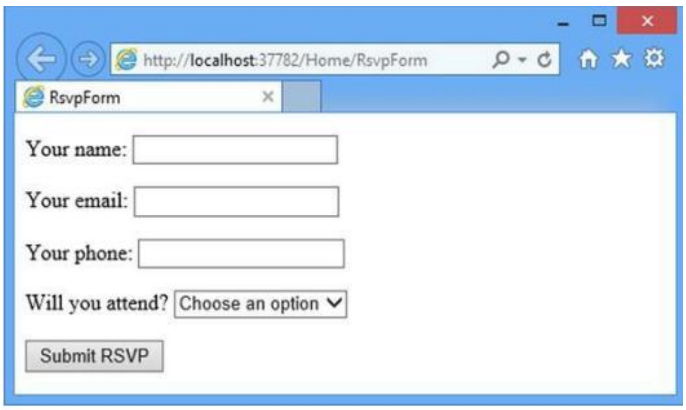


Figure 17 (ref. Pro ASP.NET MVC 5 Adam Freeman)

Handling Form:

We have not yet told MVC what we want to do when the form is posted to the server. As things stand, clicking the Submit RSVP button just clears any values we have entered into the form. That is because the form posts back to the RsvpForm action method in the Home controller, which just tells MVC to render the view again.

To receive and process submitted form data, use a clever feature and add a second RsvpForm action method in order to create the following:

1. A method that responds to HTTP GET requests: A GET request is what a browser issues normally each time someone clicks a link. This version of the action will be responsible for displaying the initial blank form when someone first visits/Home/RsvpForm.

2. A method that responds to HTTP POST requests: By default, forms rendered using Html.BeginForm()are submitted by the browser as a POST request. This version of the action will be responsible for receiving submitted data and deciding what to do with it.
   Handing GET and POST requests in separate C# methods helps to keep my controller code tidy, since the two methods have different responsibilities. Both action methods are invoked by the same URL, but MVC makes sure that the appropriate method is called, based on whether I am dealing with a GET or POST request. Listing 3.7 shows the changes to applied to the HomeController class.

Listing 3.7.

Adding an Action Method to Support POST Requests in the HomeController.cs File

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using PartyInvites.Models;
namespace PartyInvites.Controllers {
public class HomeController : Controller {
public ViewResult Index() {
int hour = DateTime.Now.Hour;
ViewBag.Greeting = hour < 12 ? "Good Morning" : "Good
Afternoon";
return View();
} [
HttpGet]
```

```
public ViewResult RsvpForm() {
return View();
} [
HttpPost]
public ViewResult RsvpForm(GuestResponse guestResponse) {
// TODO: Email response to the party organizer
return View("Thanks", guestResponse);
}
}
}
```

Rendering other views:

The second overload of the RsvpForm action method also demonstrates how to tell MVC to render a specific view in response

to a request, rather than the default view. Here is the relevant statement:

```
...
return View("Thanks", guestResponse);
...
```

This call to the View method tells MVC to find and render a view called Thanks and to pass the GuestResponse object to the view. To create the view I specified, right-click on any of the HomeController methods and select AddView from the pop-up menu and use the Add View dialog to create a strongly typed view called Thanks that uses the GuestResponse model class and that is based on the Empty template. (See the Adding a Strongly Typed View section for step-by-step details if needed).

Visual Studio will create the view as Views/Home/Thanks.cshtml. Edit the new view

so that it matches Listing 3.8— the markup you need to add.

Listing 3.8

The Contents of the Thanks.cshtml File

```
@model PartyInvites.Models.GuestResponse
@{
Layout = null;
} <
!DOCTYPE html>
<html>
<head>
```

```
<meta name="viewport" content="width=device-width" />
<title>Thanks</title>
</head>
<body>
<div>
<h1>Thank you, @Model.Name!</h1>
@if (Model.WillAttend == true)
{
@:It's great that you're coming. The drinks are already in the
fridge!
} else {
@:Sorry to hear that you can't make it, but thanks for letting
us know.
}
</div>
</body>
</html>
```

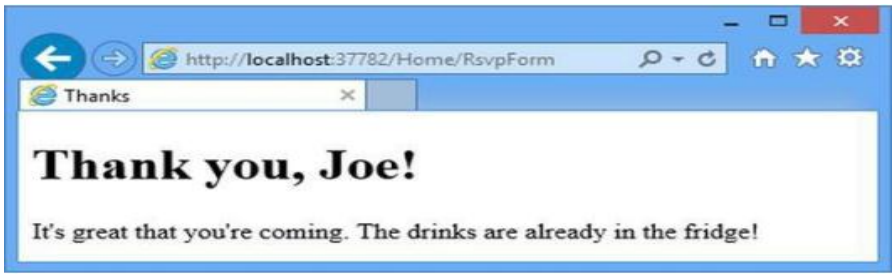


Figure 18 (ref. Pro ASP.NET MVC 5 Adam Freeman)

With Validation/Adding Validation:

Without validation, users could enter nonsense data or even submit an empty form. In an MVC application, validation is typically applied in the domain model, rather than in the user interface. This

means that it is able to define validation criteria in one place and have it take effect anywhere in the application that the model class is used. ASP.NET MVC supports declarative validation rules defined with attributes from the System.ComponentModel.DataAnnotations namespace, meaning that validation constraints are expressed using the standard C# attribute features.

Listing 3.9 shows how to applied these attributes to the GuestResponse model class.

Applying Validation in the GuestResponse.cs File

```
using System.ComponentModel.DataAnnotations;
namespace PartyInvites.Models
{
public class GuestResponse
{
[Required(ErrorMessage = "Please enter your name")]
public string Name { get; set;}
[Required(ErrorMessage = "Please enter your email address")]
[RegularExpression(".+\\@.+\\..+",
ErrorMessage = "Please enter a valid email address")]
public string Email { get; set; }
[Required(ErrorMessage = "Please enter your phone number")]
public string Phone { get; set; }
[Required(ErrorMessage = "Please specify whether you'll attend")]
public bool? WillAttend {get; set;}
}
}
```

There has been a validation problem using the ModelState.IsValid property in the controller class.

Listing 3.10 shows how I have done this in the POST-enabled RsvpForm action method in the Home controller class.

Listing 3.10 Checking for Form Validation Errors in the HomeController.cs File

```
...
[HttpPost]
public ViewResult RsvpForm(GuestResponse guestResponse) {
if (ModelState.IsValid) {
// TODO: Email response to the party organizer
return View("Thanks", guestResponse);
} else {
// there is a validation error
return View();
}
}
```

If there are no validation errors, then MVC to render the Thanks view, just as we did previously. If there are validation errors, re-render the RsvpForm view by calling the View method without any parameters.

Just displaying the form when there is an error is not helpful—also need to provide the user with some indication of what the problem is and why we could not accept their form submission. I do this by using the Html.ValidationSummary helper method in the RsvpForm view, as shown in Listing 3.11

Listing 3.11

Using the Html.ValidationSummary Helper Method in the RsvpForm.cshtml File

```
@model PartyInvites.Models.GuestResponse
@{
Layout = null;
}
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width" />
<title>RsvpForm</title>
</head>
<body>
@using (Html.BeginForm())
{
@Html.ValidationSummary()
<p>Your name: @Html.TextBoxFor(x => x.Name) </p>
<p>Your email: @Html.TextBoxFor(x => x.Email)</p>
<p>Your phone: @Html.TextBoxFor(x => x.Phone)</p>
<p>
Will you attend?
@Html.DropDownListFor(x => x.WillAttend, new[]
{


new SelectListItem() {Text = "Yes, I'll be there",Value = bool.TrueString},new SelectListItem();
{
Text = "No, I can't come",Value = bool.FalseString}
}, "Choose an option"()
```

```
</p>
<input type="submit" value="Submit RSVP" />
}
</body>
</html>
```

If there are no errors, the Html.ValidationSummary method creates a hidden list item as a placeholder in the form. MVC makes the placeholder visible and adds the error messages defined by the validation attributes. You can see how this appears in Figure 3.12

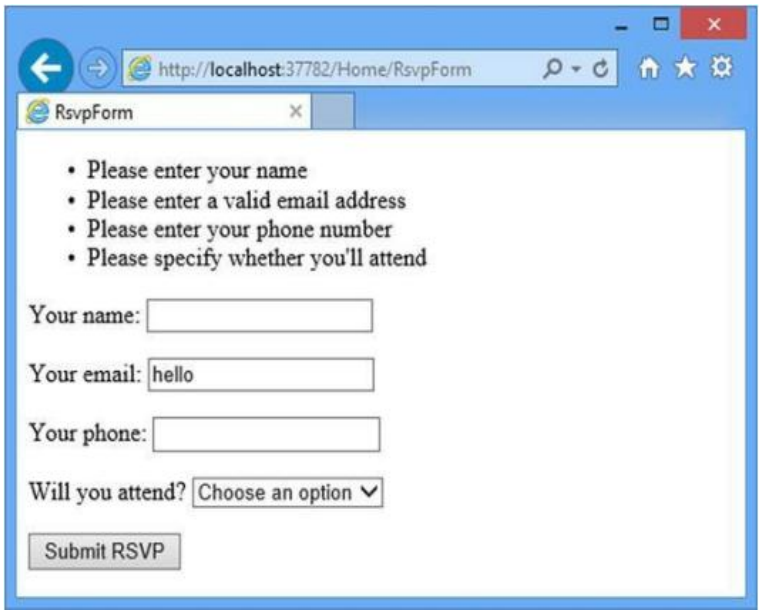


Figure 19 (ref. Pro ASP.NET MVC 5 Adam Freeman)

Highlighting Invalid Fields:

The HTML helper methods that create text boxes, drop-downs, and other elements have a handy feature that can be used in conjunction with model binding. The same mechanism that preserves the data that a user entered in a form can also be used to highlight individual fields that failed the validation checks.

When a model class property has failed validation, the HTML helper methods will generate slightly different HTML. As an example, here is the HTML that a call to Html.TextBoxFor(x => x.Name) generates when there is no validation error:

```
<input data-val="true" data-val-required="Please enter your name"
id="Name" name="Name" type="text" value="" />
```

And here is the HTML the same call generates when the user doesn’t provide a value (which is a validation error because applied the Required attribute to the Name property in the GuestResponse model class):

<input class="input-validation-error" data-val="true" data-val-required="Please enter your name" id="Name" name="Name" type="text" value="" />

The highlighted the difference in bold: the helper method added a class called input-validation-error to the input element. It can take advantage of this feature by creating a style sheet that contains CSS styles for this class and the others that different HTML helper methods apply.

The convention in MVC projects is that static content, such as CSS style sheets, is placed into a folder called Content. Create this folder by right-clicking on the PartyInvites item in the Solution Explorer, selecting Add New Folder from the menu and setting the name to Content.

To create the CSS file, right click on the newly created Content folder, select Add New Item from the menu and choose Style Sheet from the set of item templates. Set the name of the new file to Styles.css

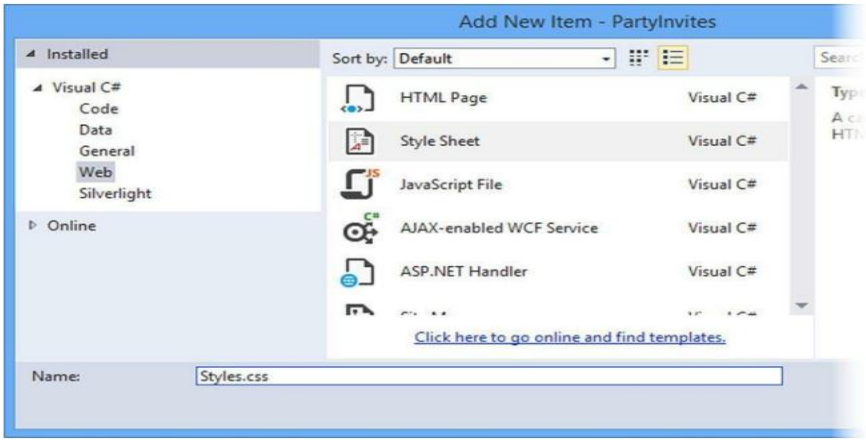


Figure 20 (ref. Pro ASP.NET MVC 5 Adam Freeman)

Click the Add button and Visual Studio will create the Content/Styles.css file. Set the content of the new file to match listing 3.11

Listing 3.11

The Contents of the Styles.css File

```
.field-validation-error  {color: #f00;}
.field-validation-valid  { display: none;}
.input-validation-error   {boreder: 1px solid # f00; background-color:free;}
.validation-summary-errors { font-weight: bold; color: #f00;}
.validation-summary-valid  { display: none;}
```

To use this style sheet, add a new reference to the head section of RsvpForm view.

Just Add link elements to views just as you would to a regular static HTML file, although , It show you the bundles feature that allows

JavaScript and CSS style sheets to be consolidated and delivered to the browsers over a single HTTP request.

Adding the Link Element in the RsvpForm.cshtml File.

```
@model PartyInvites.Models.GuestResponse
@{
Layout = null;
} <!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width" />
<link rel="stylesheet" type="text/css" href="~/Content/Styles.css"/>
<title>RsvpForm</title>
</head>
<body>
@using (Html.BeginForm())
{
@Html.ValidationSummary()
<p>Your name: @Html.TextBoxFor(x => x.Name) </p>
<p>Your email: @Html.TextBoxFor(x => x.Email)</p>
<p>Your phone: @Html.TextBoxFor(x => x.Phone)</p>
<p>
Will you attend?
@Html.DropDownListFor(x => x.WillAttend, new[]
{
new SelectListItem() {Text = "Yes, I'll be there",
Value = bool.TrueString},
new SelectListItem() {Text = "No, I can't come",
Value = bool.FalseString}
}, "Choose an option")
</p>
<input type="submit" value="Submit RSVP" />
}
</body>
</html>
```

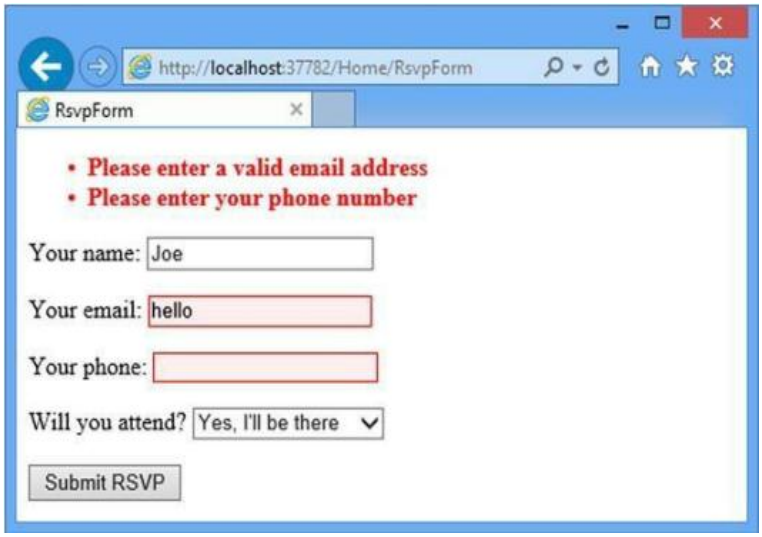


Figure 21 (ref. Pro ASP.NET MVC 5 Adam Freeman)

Adding Bootstrap to the Index.cshtml File

```
@{Layout = null;
} <!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width" />
<link href="~/Content/bootstrap.css" rel="stylesheet" />
<link href="~/Content/bootstrap-theme.css" rel="stylesheet" />
<title>Index</title>
<style>
.btn a { color: white; text-decoration: none}
body { background-color: #F1F1F1; }
</style>
</head>
<body>
<div class="text-center">
<h2>We're going to have an exciting party!</h2>
<h3>And you are invited</h3>
<div class="btn btn-success">
@Html.ActionLink("RSVP Now", "RsvpForm")
</div>
</div>
</body>
</html>
```

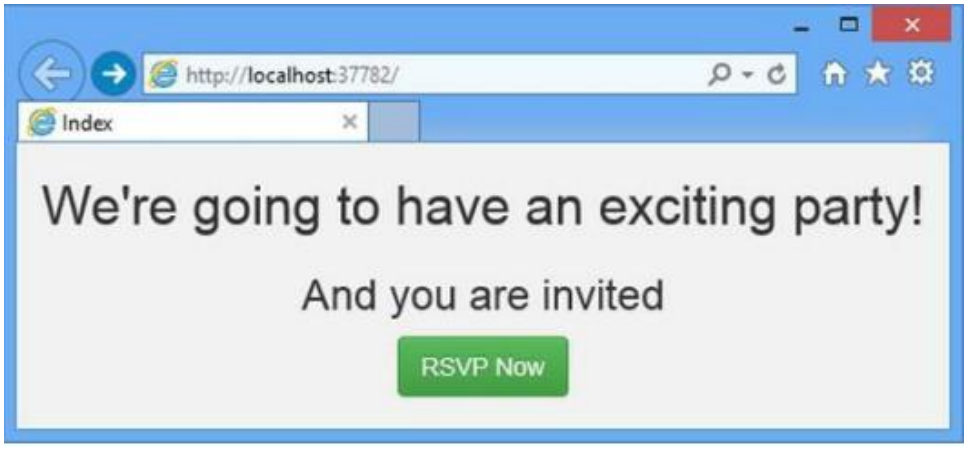


Figure 21 (ref. Pro ASP.NET MVC 5 Adam Freeman)

Styling the RsvpForm View

Bootstrap defines classes that can be used to style forms. I am not going to go into detail, but you can see how I have applied

these classes in Listing 2-22.

Listing 2-22. Adding Bootstrap to the RsvpForm.cshtml File

```
@model PartyInvites.Models.GuestResponse
@{
Layout = null;
} <
!DOCTYPE html>
<html>
<head>
<link href="~/Content/bootstrap.css" rel="stylesheet" />
<link href="~/Content/bootstrap-theme.css" rel="stylesheet" />
<meta name="viewport" content="width=device-width" />
<link href="~/Content/Styles.css" rel="stylesheet" />
<title>RsvpForm</title>
61
</head>
<body>
<div class="panel panel-success">
<div class="panel-heading text-center"><h4>RSVP</h4></div>
<div class="panel-body">
@using (Html.BeginForm()) {
@Html.ValidationSummary()
```

```
<div class="form-group">
<label>Your name:</label>
@Html.TextBoxFor(x => x.Name, new { @class = "formcontrol"})
</div>
<div class="form-group">
<label>Your email:</label>
@Html.TextBoxFor(x => x.Email, new { @class = "formcontrol"})
</div>
<div class="form-group">
<label>Your phone:</label>
@Html.TextBoxFor(x => x.Phone, new { @class = "formcontrol"})
</div>
<div class="form-group">
<label>Will you attend?</label>
@Html.DropDownListFor(x => x.WillAttend, new[] {
new SelectListItem() {Text = "Yes, I'll be
there",
Value = bool.TrueString},
new SelectListItem() {Text = "No, I can't
come",
Value = bool.FalseString}
}, "Choose an option", new { @class = "formcontrol" })
</div>
<div class="text-center">
<input class="btn btn-success" type="submit"
value="Submit RSVP" />
</div>
}
</div>
</div>
</body>
</html>
```

The Bootstrap classes in this example create a panel with a header, just to give structure to the layout. To style the form, used the form-group class, which is used to style the element that contains the label and the associated input or select element.

These elements are created using HTML helper methods, which means that there are not statically defined elements available to which I can apply the required form-control class. Fortunately, the helper methods take an optional object argument that lets me specify attributes on the elements that they create, as follows:

@Html.TextBoxFor(x => x.Name, new { @class = "form-control"})

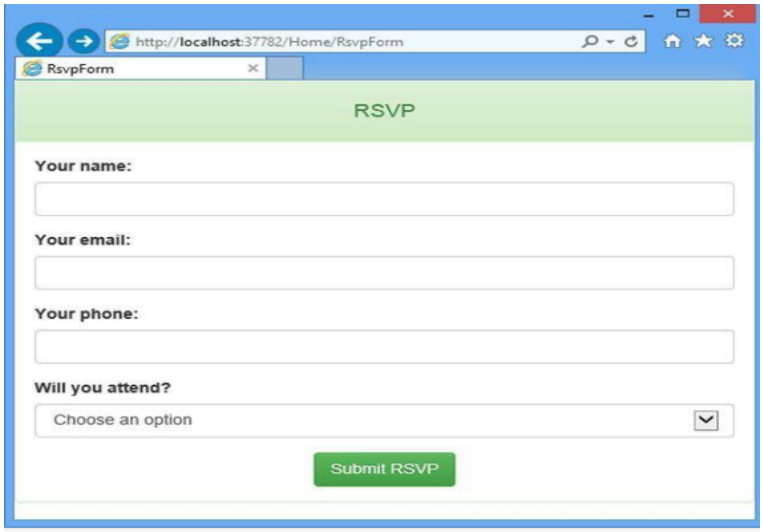


Figure 22 (ref. Pro ASP.NET MVC 5 Adam Freeman)

Styling the Thanks View

The last view file to style is Thanks.cshtml and you can see how I have done this in Listing . You will notice that the markup we have added is similar to that in the Index.cshtml view. To make an application easier to manage, it is a good principal to avoid duplicating code and markup wherever possible and will introduce you to Razor layouts

Applying Bootstrap to the Thanks.cshtml File

```
@model PartyInvites.Models.GuestResponse
@{
Layout = null;
}
<!DOCTYPE html>
<html>
<head>
<link href="~/Content/bootstrap.css" rel="stylesheet" />
<link href="~/Content/bootstrap-theme.css" rel="stylesheet" />
<meta name="viewport" content="width=device-width" />
<title>Thanks</title>
<style>
```

```
body { background-color: #F1F1F1; }
</style>
</head>
<body>
<div class="text-center">
<h1>Thank you, @Model.Name!</h1>
<div class="lead">
@if (Model.WillAttend == true)
{
@:It's great that you're coming. The drinks are already in
the fridge!
}
else
{
@:Sorry to hear that you can't make it, but thanks for
letting us know.
}
</div>
</div>
</body>
</html>
```

Figure 23 (ref. Pro ASP.NET MVC 5 Adam Freeman)

4. Using Automatically Implemented Property:

The regular C# property feature lets you expose a piece of data from a class in a way that decouples the data from how it is set and retrieved. Following Listing contains a simple example in a class called of Product, which added to the Models folder of the

LanguageFeatures project in a class file called Product.cs

Listing

Defining a Property in the Product.cs File

```
namespace LanguageFeatures.Models {
public class Product {
private string name;
public string Name {
get { return name; }
set { name = value; }
}
}
}
```

The property, called Name, is shown in bold. The statements in the get code block (known as the getter) are performed when the value of the property is read, and the statements in the set code block (known as the setter) are performed when a value is assigned to the property (the special variable value represents the assigned value). A property is consumed by other

classes as though it were a field, as shown in Listing, which shows an AutoProperty action method I added to the Home controller.

Consuming a Property in the HomeController.cs File

```
using System;
using System.Web.Mvc;
using LanguageFeatures.Models;
namespace LanguageFeatures.Controllers {
public class HomeController : Controller {
public string Index() {
return "Navigate to a URL to show an example";
}
public ViewResult AutoProperty() {
// create a new Product object
Product myProduct = new Product();
// set the property value
myProduct.Name = "Kayak";
// get the property
string productName = myProduct.Name;
// generate the view
return View("Result",
(object)String.Format("Product name: {0}", productName));
}
```

```
}
}
```

We can see that the property value is read and set just like a regular field. Using properties is preferable to using fields because we can change the statements in the get and set blocks without needing to change the classes that depend on the property.

Product name: Kayak

Properties added to the Product class in the Product.cs file.

Verbose Property Definitions in the Product.cs File

```
namespace LanguageFeatures.Models
{
public class Product
{
private int productID;
private string name;
private string description;
private decimal price;
private string category;
public int ProductID
   {
      get { return productID; }
      set { productID = value; }
    }
public string Name
{
     get { return name; }
     set { name = value; }
}
public string Description
{
   get { return description; }
   set { description = value; }
} /
/...and so on...
}
}
```

10.4 USING AUTOMATICALLY IMPLEMENTED PROPERTIES

Using Automatically Implemented Properties in the Product.cs File:

```
namespace LanguageFeatures.Models {
public class Product {
public int ProductID { get; set; }
public string Name { get; set; }
public string Description { get; set; }
public decimal Price { get; set; }
public string Category { set; get; }
}
}
```

Do not define the bodies of the getter and setter or the field that the property is backed by. Both of these are done for me by the C# compiler when the class is compiled. Using an automatic property is no different from using a regular property;

The code in the action method in Listing will work without any modification

Reverting from an Automatic to a Regular Property in the Product.cs File

```
namespace LanguageFeatures.Models {
public class Product {
private string name;
public int ProductID { get; set; }
public string Name
 {
   get
 {
 return ProductID + name;
 }
 set
{
name = value;
}
}
public string Description {get; set;}
public decimal Price {get; set;}
```

```
public string Category {set; get;}
}
}
```

Note that I must implement both the getter and setter to return to a regular property. C# does not support mixing automatic- and regular-style getters and setters in a single property.

10.5 USING OBJECT AND COLLECTION INITIALIZER

Another tiresome programming task is constructing a new object and then assigning values to the properties, as illustrated by Listing which shows the addition of a CreateProduct action method to the Home controller.

Constructing and Initializing an Object with Properties in the HomeController.cs File

```
using System;
using System.Web.Mvc;
using LanguageFeatures.Models;
namespace LanguageFeatures.Controllers
{
public class HomeController : Controller
{
public string Index()
{
return "Navigate to a URL to show an example";
}
public ViewResult AutoProperty()
{
// ...statements omitted for brevity...
}
public ViewResult CreateProduct()
{
// create a new Product object
Product myProduct = new Product();
// set the property values
myProduct.ProductID = 100;
myProduct.Name = "Kayak";
myProduct.Description = "A boat for one person";
myProduct.Price = 275M;
myProduct.Category = "Watersports";
```

```
return View("Result",
(object)String.Format("Category: {0}",
myProduct.Category));
}
}
}
```

Using the Object Initializer Feature in the HomeController.cs File

```
...
public ViewResult CreateProduct()
{
// create and populate a new Product object
Product myProduct = new Product
{
ProductID = 100, Name = "Kayak",
Description = "A boat for one person",
Price = 275M, Category = "Watersports"
};
return View("Result",(object)String.Format("Category: {0}",
myProduct.Category));
}
```

Initializing Collections and Arrays in the HomeController.cs File

```
using System;
using System.Collections.Generic;
using System.Web.Mvc;
using LanguageFeatures.Models;
namespace LanguageFeatures.Controllers {
public class HomeController : Controller {
public string Index()
{
return "Navigate to a URL to show an example";
} /
/ ...other action methods omitted for brevity...
public ViewResult CreateCollection()
{
string[]stringArray = { "apple", "orange", "plum" };
List<int> intList = new List<int> { 10, 20, 30, 40 };
Dictionary<string, int> myDict = new Dictionary<string, int>
```

```
{
{ "apple", 10}, {"orange", 20}, {"plum", 30}
};
return View("Result", (object)stringArray[1]);
}
}
}
```

10.6 USING EXTENSION METHODS

Extension methods are a convenient way of adding methods to classes that you do not own and cannot modify directly. Listing shows a ShoppingCart class, which is added to the Models folder in a file called ShoppingCart.cs file and which represents a collection of Product objects.

The ShoppingCart Class in the ShoppingCart.cs File

```
using System.Collections.Generic;
namespace LanguageFeatures.Models;
 {
public class ShoppingCart
{
public List<Product> Products {get; set;}
}
}
```

This is a simple class that acts as a wrapper around a List of Product objects

Listing shows the MyExtensionMethods

Models folder in the MyExtensionMethods.cs

file. Listing Defining an Extension Method in the MyExtensionMethods.cs File

```
namespace LanguageFeatures.Models
{
public static class MyExtensionMethods
{
public static decimal TotalPrices(this ShoppingCart cartParam)
{
decimal total = 0;
foreach (Product prod in cartParam.Products)
 {
```

```
total += prod.Price;
}r
eturn total;
}
}
}
```

Applying an Extension Method in the HomeController.cs File

```
using System;
using System.Collections.Generic;
using System.Web.Mvc;
using LanguageFeatures.Models;
namespace LanguageFeatures.Controllers
{
public class HomeController : Controller
{
public string Index()
 {
return "Navigate to a URL to show an example";
} /
/ ...other action methods omitted for brevity...
public ViewResult UseExtension()
{
// create and populate ShoppingCart
ShoppingCart cart = new ShoppingCart
 {
Products = new List<Product>
{
new Product {Name = "Kayak", Price = 275M},
new Product {Name = "Lifejacket", Price = 48.95M},
new Product {Name = "Soccer ball", Price = 19.50M},
new Product {Name = "Corner flag", Price = 34.95M}
}
};
// get the total value of the products in the cart
decimal cartTotal = cart.TotalPrices();
return View("Result",
```

```
(object)String.Format("Total: {0:c}", cartTotal));
}
}
}
```

10.7 USING LAMBDA EXPRESSION

We can use a delegate to make my FilterByCategory method more general. That way, the delegate that will be invoked against each Product can filter the objects in any way I choose, as illustrated by Listing, which shows the Filter extension method I added to the MyExtensionMethods class.

Listing Using a Delegate in an Extension Method in the MyExtensionMethods.cs File

```
using System;
using System.Collections.Generic;
namespace LanguageFeatures.Models;
{
public static class MyExtensionMethods
{
decimal total = 0;
foreach (Product prod in productEnum)
{
total += prod.Price;
}
return total;
}
public static IEnumerable<Product> FilterByCategory
(this IEnumerable<Product> productEnum, string
categoryParam)
{
foreach (Product prod in productEnum)
{
if (prod.Category == categoryParam)
{
yield return prod;
}
}
}
```

```
public static IEnumerable<Product> Filter
(this IEnumerable<Product> productEnum, Func<Product, bool>
selectorParam)
{
foreach (Product prod in productEnum)
{
if (selectorParam(prod))
{
yield return prod;
}
}
}
}
}
```

Using the Filtering Extension Method with a Func in the HomeController.cs File

```
public ViewResult UseFilterExtensionMethod() {
// create and populate ShoppingCart
IEnumerable<Product> products = new ShoppingCart {
Products = new List<Product> {
new Product {Name = "Kayak", Category = "Watersports", Price =
275M},
new Product {Name = "Kayak", Category = "Watersports", Price =
48.95M},
new Product {Name = "Soccer ball", Category = "Soccer", Price
= 19.50M},
new Product {Name = "Corner flag", Category = "Soccer", Price
= 34.95M}
}
};
Func<Product, bool> categoryFilter = delegate(Product prod)
 {
```

```
return prod.Category == "Soccer";
};
decimal total = 0;
foreach (Product prod in products.Filter(categoryFilter))
{
total += prod.Price;
}
return View("Result", (object)String.Format("Total: {0}", total));
}
```

Using a Lambda Expression to Replace a Delegate Definition in the HomeController.cs File

```
public ViewResult UseFilterExtensionMethod()
{
// create and populate ShoppingCart
IEnumerable<Product> products = new ShoppingCart {
Products = new List<Product> {
new Product {Name = "Kayak", Category = "Watersports", Price =
275M},
new Product {Name = "Kayak", Category = "Watersports", Price =
48.95M},
new Product {Name = "Soccer ball", Category = "Soccer", Price
= 19.50M},
new Product {Name = "Corner flag", Category = "Soccer", Price
= 34.95M}
}
};
Func<Product, bool> categoryFilter = delegate(Product prod)
 {
return prod.Category == "Soccer";
};
```

```
decimal total = 0;

foreach (Product prod in products.Filter(categoryFilter))

{

total += prod.Price;

}
```

Other forms of lambda expression:

```
1.prod => EvaluateProduct(prod)

2. (prod, count) => prod.Price > 20 && count > 0

3. (prod, count) => {

//...multiple code statements

return result;

}
```

10.8 PROGRAMS BASED ON MVC PATTERN

MVC Pattern:

The history of MVC Pattern:

The time period version-view-controller has been in use since the past due 1970s and arose from the Smalltalk challenge at Xerox PARC, wherein it became conceived as a way to arrange a few early GUI applications. Some of the first-class detail of the authentic MVC sample changed into tied to

Smalltalk-unique standards, including monitors and tools, however the broader standards are nonetheless relevant to programs—and they may be specifically nicely acceptable to Web applications.

Interactions with an MVC software observe a herbal cycle of consumer movements and examine updates, where the view is assumed to be stateless. This fits well with the HTTP requests and responses that underpin a Web utility.

Further, MVC forces a separation of worries—the domain version and controller logic are decoupled from the user interface. In a Web application, which means that the HTML is kept apart from the relaxation of the software, which makes maintenance and testing simpler and easier. It become Ruby on Rails that led to renewed mainstream interest in MVC and it remains the implementation template for the MVC pattern. Many different MVC frameworks have seeing that emerged and confirmed the advantages of MVC—including, of path, ASP.NET MVC.

The ASP.NET Implementation of MVC:

In MVC, controllers are C# lessons, normally derived from the System.Web.Mvc.Controller elegance. Each public technique in a category derived from Controller is an movement approach, which is related to a configurable URL through the ASP.NET routing device. When a request is despatched to the URL related to an movement approach, the statements in the controller magnificence are completed on the way to carry out a few operation on the domain model and then choose a view to display to the consumer. Figure shows the interactions between the controller, model, and view.

The ASP.NET MVC Framework uses a view engine, which is the component responsible for processing a view in order to generate a response for the browser. Earlier versions of MVC used the standard ASP.NET view engine, which processed ASPX pages using a streamlined version of the Web Forms markup syntax. MVC 3 introduced the Razor view engine, which was refined in MVC 4 (and unchanged in MVC5) and that uses a different syntax entirely.

10.9 FORMS AND HTML HELPERS

Helper:

Creating custom helper method:

The simplest kind of helper method is an inline helper, which is defined within a view. I can create an inline helper to simplify the example view using the @helper tag, as shown in Listing 21-3.

Listing .

Creating an Inline Helper Method in the Index.cshtml File

```
@model string
@{
Layout = null;
}
@helper ListArrayItems(string[] items)
{
foreach(string str in items)
{
<b>@str </b>
}
}
<!DOCTYPE html>
<html>
```

```
<head>
<meta name="viewport" content="width=device-width" />
<title>Index</title>
</head>
<body>
<div>
Here are the fruits: @ListArrayItems(ViewBag.Fruits)
</div>
<div>
Here are the cities: @ListArrayItems(ViewBag.Cities)
</div>
<div>
Here is the message:
<p>@Model</p>
</div>
</body>
</html>
```

Changing the Contents of a Helper Method in the Index.cshtml File...

```
@helper ListArrayItems(string[] items) {
<ul>
@foreach(string str in items) {
<li>@str</li>
}
</ul>
}
```

Creating an external helper Method:

Inline helpers are convenient, but they may be used only from the view wherein they're declared and, if they're too complicated, they could take over that view and make it tough to read.

The alternative is to create an outside HTML helper technique, that is expressed as a C# extension method. External helper methods can be used greater extensively, however are a touch more awkward to put in writing, due to the fact C# doesn’t certainly take care of HTML element era elegantly. To display this selection, I delivered an Infrastructure folder to the example task and created a new CustomHelpers.Cs elegance report inside it. You can see the contents of this document in listing

Listing:

The Contents of the CustomHelpers.cs File

```
using System.Web.Mvc;
namespace HelperMethods.Infrastructure
{
public static class CustomHelpers
{
public static MvcHtmlString ListArrayItems(this HtmlHelper html,
string[] list)
{
TagBuilder tag = new TagBuilder("ul");
foreach(string str in list)
{
TagBuilder itemTag = new TagBuilder("li");
itemTag.SetInnerText(str);
tag.InnerHtml += itemTag.ToString();
}
return new MvcHtmlString(tag.ToString());
}
}
}
```

10.10 DEFINING THE MODEL

To start with a simple domain model called Product, defined in a class file called Product.cs, added to the Models folder. You can see the contents of the new file in Listing.

Listing.

The Contents of the Product.cs File

```
namespace Razor.Models
{
public class Product
{
public int ProductID { get; set; }
public string Name { get; set; }
public string Description { get; set; }
public decimal Price { get; set; }
```

```
public string Category { set; get; }
}
}
```

10.11 SUMMARY

In this chapter, we created a new MVC project and used it to assemble a simple MVC facts-entry application, giving you a first glimpse of the MVC Framework structure and approach. We ignored some key capabilities (consisting of Razor syntax, routing, and automated testing), however we come back to those topics in depth in later chapters. In the subsequent bankruptcy,describe the MVC structure, design styles, and strategies that we use all through the relaxation of this e-book and which form the foundation for powerful improvement with the MVC Framework.

10.12 UNIT END EXERCISES

1. Tell us something about views and model?
2. Briefly explain why ASP.NET MVC?
3. What is use of Views in ASP.NET MVC?
4. How does works controller in ASP.NET MVC?
5. What is View Data?
6. What are the helpers in ASP.NET MVC?

10.13 LIST OF REFERENCES

- Freeman, Adam. "Pro asp. netmvc 5 platform." Pro ASP. NET MVC 5 Platform. Apress, Berkeley, CA, 2014. ISBN: 1430265418
- Allen, K. Scott, et al. Professional ASP. NET MVC 5. Wrox Press, 2014. ISBN: 1118794753
- MVC Framework - First Application (tutorialspoint.com)

*****

11

VIEW

Unit structure

11.0 OBJECTVIES

1. In this chapter we learn using Partial views how to reduce the duplication in views?
2. How to specify a default layout in views?
3. How to pass data values to the view from the controller?
4. How to generate different content based on the data values?
5. How add a namespace to a view?

11.1 REDUCE DUPLICATION IN VIEWS

In the previous chapters we had learned that what is view and their types:

A View, in the context of Model View Controller (MVC) architecture, It is a software class that contains a template and data from produces a response for the browser which receive data from the Controller of the MVC and packages it and present it to the browser for display.

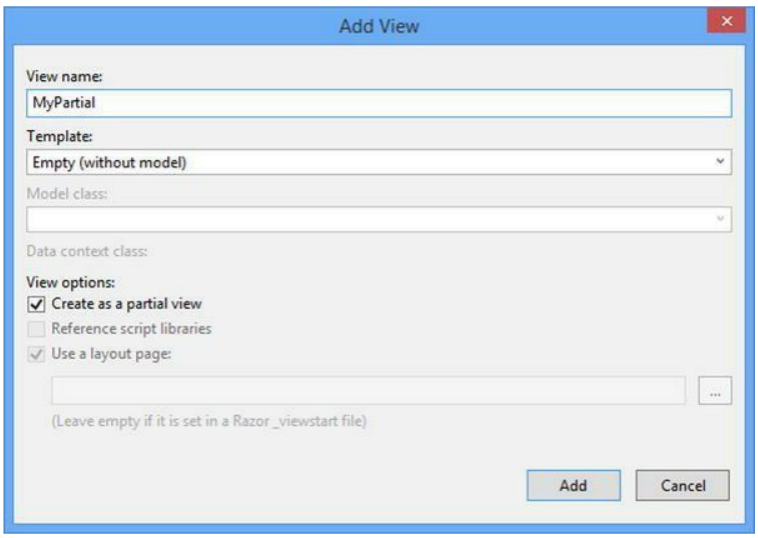
There are many types of views such as custom view engine, Razor view engine, partial view. To reduce the duplication in views we can use partial view.

We will often need to use same fragments of Razor tags and HTML markup in several different places in the application. Rather than duplicate the content, using partial views which are separate view files that content, fragments of tags and markup that can be included in other views.

In this chapter we can see how to create and use partial views? How it is work and demonstrate the technique available for passing view data to a partial view.

Creating Partial View:

1. Just start by creating Partial View called MyPartial
2. Then right click on the View/Shared folder
3. Select Add from popup menu
4. Visual studio will display the Add View dialog window, which you have seen previous chapters.
5. Set View Name to MyPartial Template to Empty and check the Create as partial view option, as following figure.



Click the Add button and Visual Studio will create partial view, which is initially empty. After adding the content as following in The Content of the MyPartial.cshtml File

This is the message from the partial view. @Html.ActionLink ("This is a link to the Index action", "Index")

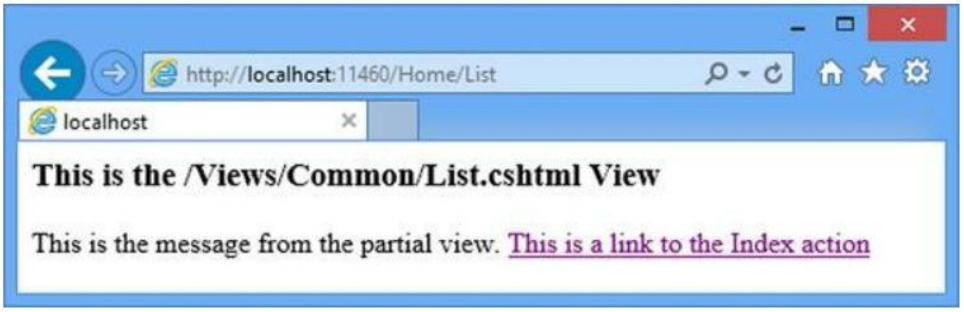
Note: use Add MVC 5 View page (razor) and set contents you require directly.

Consuming a Partial View in the List.cshtml File @{ ViewBag.Title = "List"; 548 Layout = null; }

This is THE/Views/Common/List.cshtml
View@Html.Partial("MyPartial")

We specify the name of the partial view file without the file extension. The view engine will look for the partial view that we have specified in the usual locations, which means the /Views/Home and /Views/Shared folders for this example, since we called the Html.Partial method in a view that is being rendered for the Home controller. (I set the Layout variable to

null so that we do not have to specify the sections we defined in the_Layout.cshtml file used earlier in the chapter



We can create strongly typed partial views, and then pass view model objects to be used when the partial view is rendered. To demonstrate this feature, created a new typed partial view called MyStrongTypedPartial.cshtml in the/View/Shared folder. This time, rather than use the scaffold option, after selected Add MVC 5 View page, set the name MyStrongTypedPartial and clicked the OK button to create the view. As explained previous there is nothing about the file itself that denotes a partial view, just the content and the way it is used.

The Contents of the MyStronglyTypedPartial.cshtml File.

```
@model IEnumerable<string>
<div>
This is the message from the partial view
<ul> @foreach(string str in model)
    {
      <li>@str</li>
    }
 </div>
```

We use a Razor @foreach loop to display the contents of the view model object as items in an HTML list. To demonstrate the use of this partial view, we updated the/Views/Common/List.cshtml file as shown in

Consuming a Strongly Typed Partial View in the List.cshtml File

```
@{
ViewBag.Title = "List";
Layout = null;
} <
h3>This is the /Views/Common/List.cshtml View</h3>
@HTML Partial(“MyStronglyTypedPartial”,new
[]{“Apple,”Orange”,”Pear”})
```

The difference from the previous example is that I pass an additional argument to the Partial helper method which defines the view model object. You can see the strongly typed partial view in use by starting the application and navigating to the

Partial view using child Action:

Child action are action method invoked from within a view. This lets you avoid repeating controller logic that you want to use in several places in the application. Child actions as partial views are to views. Whenever we want to display some data-driven widget that appears on multiple pages and contain data unrelated to the main action that is running. As using this technique in the SportsStore example to include a data-driven navigation menu on every page, without needing to supply the navigation data directly from every action method. The navigation data was supplied independently by the child action.

Any action can be use child action. To demonstrate the child action need to be added new action method to the Home Controller as shown

Adding a Child Action in the HomeController.cs File

```
using System;
using System.Web.Mvc;
namespace WorkingWithRazor.Controllers
{
public class HomeController:Controller
{
public ActionResult Index()
{
string[] names = { "Apple", "Orange", "Pear" };
return View(names);
}
public ActionResult List()
{
return View();
}
```

```
[ChildActionOnly]
public ActionResult Time()
{
return PartialView(DateTime.Now);
}
}
}
```

The action method is called Time and it renders a partial view by calling the PartialView method. The ChildActionOnly attribute ensures that an action method can be called only as a child method from

within a view. An action method doesn't need to have this attribute to be used as a child action, but I tend to use it to prevent the

action methods from being invoked as a result of a user request.

Having defined an action method, I need to create the partial view that will be rendered when the action is invoked. Child actions are typically associated with partial views, although this is not compulsory.

the/Views/Home/Time.cshtml view that I created for this demonstration. This is a strongly typed partial view whose view model is a DateTime object.

The Contents of the Time.cshtml File

```
@model DateTime
<p>The time is: @Model.ToShortTimeString()</p>
```

Child actions are called using the Html.Action helper with this helper the action method is executed, the ViewResult is processed, and the output is injected into the response to the client.

As shown following changes has been shows the changes made to the /Views/Common/List.cshtml file to render the child action.

Calling a Child Action in the List.cshtml File:

```
@{
ViewBag.Title = "List";
Layout = null;
} <h3>This is the /Views/Common/List.cshtml View</h3>

@Html.Partial("MyStronglyTypedPartial", new [] {"Apple", "Orange","Pear"})
@Html.Action("Time")
```

You can see the effect of the child action by starting the application and navigating to the /Home/List URL again, as shown in figure.

To call action method in other controller name

@Html.Action("Time", "MyController")

[ChildActionOnly]

public ActionResult Time(DateTime time) {

return PartialView(time);

}

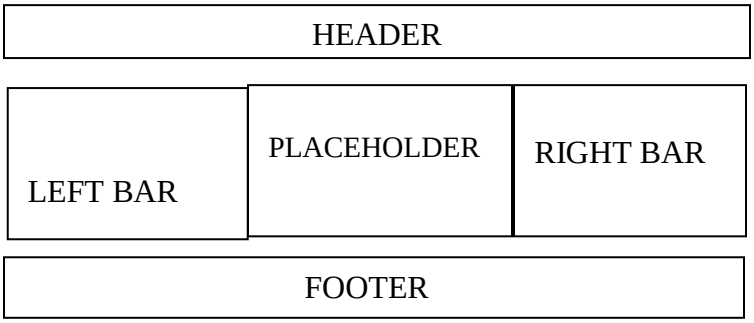
Then can invoke it from view as follows:

(@Html.Action("Time", new { time = DateTime.Now })

In this section, you will find out about the format view in ASP.NET MVC.

An utility may additionally contain a selected UI portion that stays the identical in the course of the utility, together with header, left navigation bar, right bar, or footer phase. ASP.NET MVC added a Layout view which incorporates those common UI quantities so that we do not should write the same code in each page. The format view is the same as the grasp page of the ASP.NET webform software.

For example, an application UI may additionally include a header, left menu bar, right bar, and footer segment that remains the same on each web page. Only the middle phase adjustments dynamically, as shown



11.2 SPECIFY A DEFAULT LAYOUT

The layout view has extension like other view such as .cshtml or .vbhtml,Layout views are shared with multiple views, so it stored in the shared folder compulsory. By default a layout view _Layout.cshtml is created when we are creating MVC application ,

Adding Markup to the ProductSummary.cshtml File

@model SportsStore.Domain.Entities.Product

<h3>

@Model.Name

<span class="pull-right label
labelprimary">@Model.Price.ToString("c")

</h3>

 @Model.Description

Now need to update Views/Products/List.cshtml so that it uses the partial view

Using a Partial View in the List.cshtml File:

@model SportsStore.WebUI.Models.ProductsListViewModel

@{

ViewBag.Title = "Products";

} @

foreach (var p in Model.Products) {

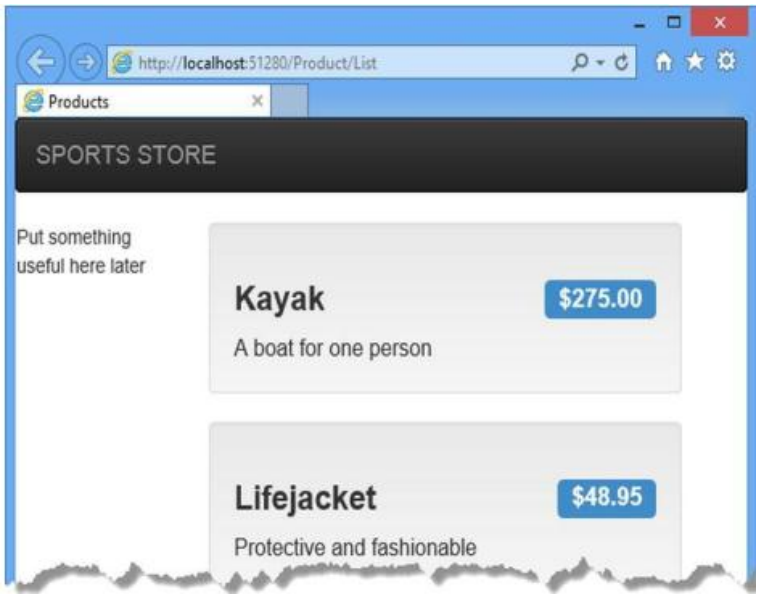
@Html.Partial("ProductSummary", p)

} <

div class="pager">

@Html.PageLinks(Model.PagingInfo, x => Url.Action("List", new {page
=

x}))



Reference: Pro Asp.Net MVC 5, Adam Freeman

11.3 Pass data values to the view from the controller

How to Pass data values to the view from the controller to view:

Through the various ways we can pass the data from the controller to view. In this chapter we will learn how to interact with Views and specifically Cover ways we can pass data from a controller to View to render a response to client.

View Bag:

View Bag uses dynamic feature that was added in C# 4.0, ViewBag=ViewData + Dynamic wrapper around the ViewData Dictionary.

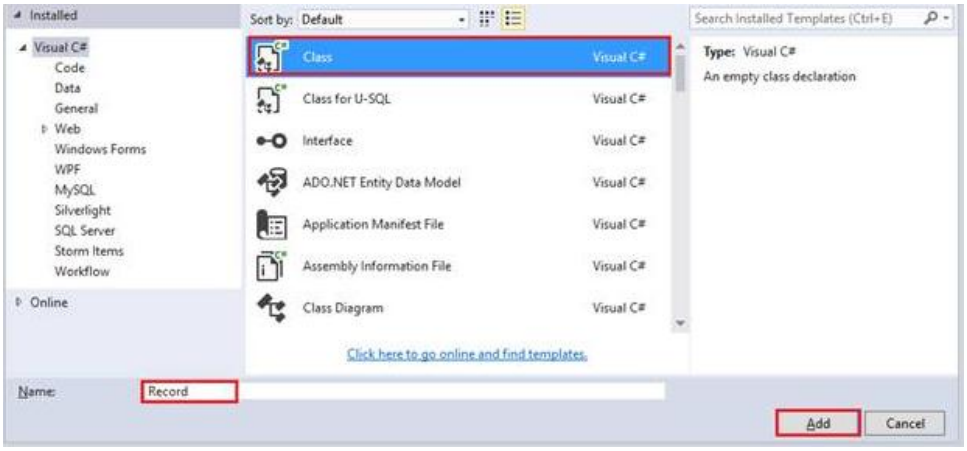
11.4 GENERATE DIFFERENT CONTENT BASED ON DATA VALUES

Pass Data between View and Controller:

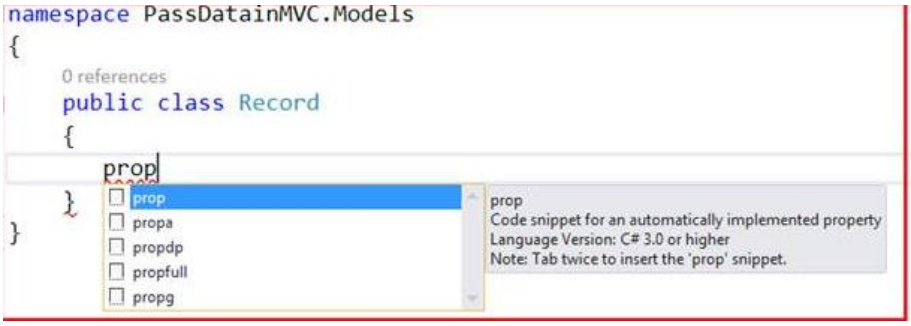
Steps:

1. Start the Microsoft Visual Studio.
2. Select the New-□Project□name to ASP.NET Web Application(.Net Framework)-□click on the OK button-□ due to the article is based on simple example just Select empty template by check to MVC
3. By right clicking on model folder add model class under solution explorer

4.



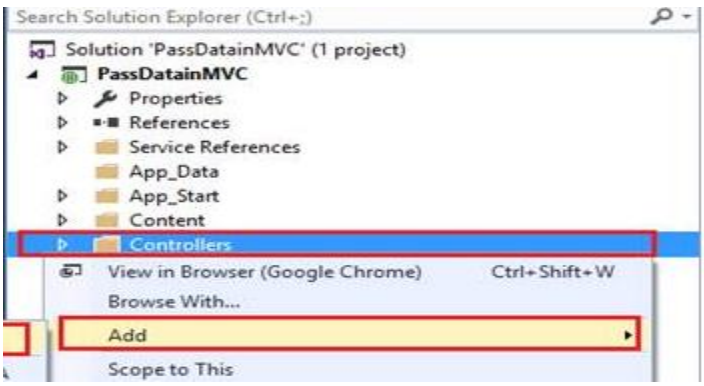
5. Just type code for implementing property automatically and press tab button which is available on the keyboard twice



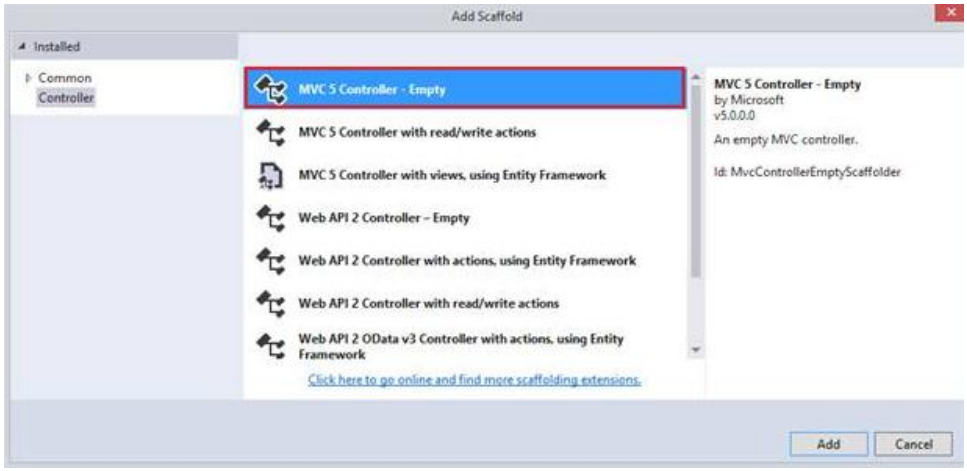
6. The model class contain only three properties:

```
Public Class Result
        {
         Public int id {get; set; }
         Public string ResultName { get; set; }
         Public string ResultRecord { get; set; }
        }
```

7. By right clicking on Controller folder add controller Class



8. Select an Empty Controller from the list of Scaffold. View



9. Give the name your Controller like ResultController . Click on Add.

10. By using ViewBag data can pass from Controller to View.

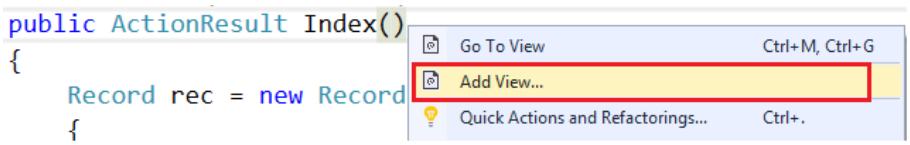
11. **ViewBag:** It gets the dynamic view data.

12. Now create object class and assign it to in ViewBag data dictionary. Assign the value property based on the class.

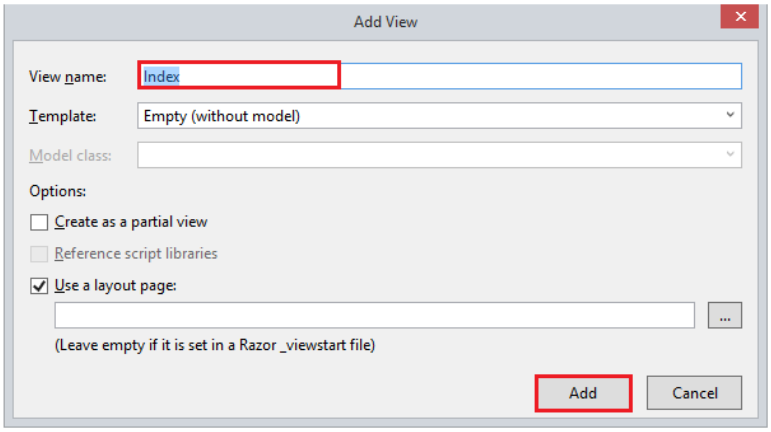
13. Public ActionResult Index()

```
{
   Result res= new Result;
    Id= '97';
ResultName= "M.Sc.IT";
ResultDetail= " The basic Sheet";
 }
ViewBag.Message=res;
}
```

14. Now add View for an Index by right clicking on it.



15.



Now assign the variable into ViewBag, all properties will be in place

16. **Insert the code:**

```
@using PassDetainMVC.Models
@{
   ViewBag.Title= “Index”;
}
<h3> Passing data from controller to View </h3>
@{
      Var data=ViewBag.Message;
    }
<h3> Id: @data.Id</h3>
<h3> ResultName:@data.ResultName</h3>
<h3> ResultDetail:@data.ResultDetail</h3>
```

Now build and execute/run the code you will get output.

2. Now passing the data from Controller to View

1. Create object same as above
2. Public Class ResultsController: Controller

```
{
 // Get: Result
Public ActionResult Index ()
          {
            Results res= new results
                 {
                  Id= 97
```

```
            Id= ‘97’;
            ResultName= “M.Sc.IT”;
            ResultDetail= “ The basic Sheet”;
                }
ViewData[“Message”]=res;
return View();
}

}
```

3. Access model class when you are using ViewData
4. @using PassDatainMVC.Models
5. @{
6. ViewBag.Title = "Index";
7. }
8. <h3>Passing Data From Controller To View using ViewData</h3>
9. @{
10. var data = (Record)ViewData["Message"];
11. }
12. <h3>Id: @data.Id</h3>
13. <h3>ResultName: @data.ResultName</h3>
14. <h3>ResultDetail: @data.ResultDetail</h3>

Then build and run the code you will get output.

11.5 ADD A NAMESPACE TO A VIEW

How to add namspaces in view:

Unlike many templating engines or interpreted view engines, Razor views are dynamically compiled at run time into classes and then executed. The compilation happens the first time the viewis requested, which incurs at a slight one-time performance cost. The benefit is that the next timethe view is used, it’s running fully compiled code. If the content of the view changes, ASP.NET willautomatically recompile the view.

The class that a view is compiled into derives from WebViewPage, which itself derives fromWebPageBase, which you saw earlier in the section

"Templated Razor Delegates." For long-time ASP.NET users, this kind of derivation should come as no surprise because it is similar to how an ASP.NET Web Forms page works with its Page base class as well.

You can change the base type for Razor views to a custom class, which makes it possible for you to add your own methods and properties to views. The base type for Razor views is defined within the Web.config file in the Views directory. The following section of Web.config contains the razor configuration.

<system.web.webPages.razor>

<host"
factoryType="System.web.Mvc.MvcWebRazorhostfactory,system.Eeb.Mvc"version=3.0.0.0,culture=neutral,PublicKeyToken=31BF3857E45/>

<pages pageBaseType="system.web.mvc.WebViewPage">

<namespaces>

<add namespace="System.Web.Mvc" />

<add namespace="System.Web.Mvc.Ajax" />

<add namespace="System.Web.Mvc.Html" />

<add namespace="System.Web.Routing" />

</namespaces>

</pages>

</system.web.webPages.razor>

11.6 SUMMARY

In this chapter we have leaned about the how to Reduce duplication in views, Specify a default layout, Pass data values to the

view from the controller, generate different content based on data values, add namespace to view in MVC application.

11.7 UNIT END EXERCISES

1. Create one MVC application and create code to reduce the duplication in view.
2. In the same application specify default layout.
3. Explain with example of steps pass data value to the view from the controller.
4. Write a steps with an example to add namespace to view in MVC application.

11.8 LIST OF REFERENCES

- Freeman, Adam. "Pro asp. netmvc 5 platform." Pro ASP. NET MVC 5 Platform. Apress, Berkeley, CA, 2014. ISBN: 1430265418
- Allen, K. Scott, et al. Professional ASP. NET MVC 5. Wrox Press, 2014. ISBN: 1118794753
- MVC Framework - First Application (tutorialspoint.com)

*****