1

INTRODUCING .NET

Unit Structure

1.0 OBJECTIVE

After going through this unit you will be able to,

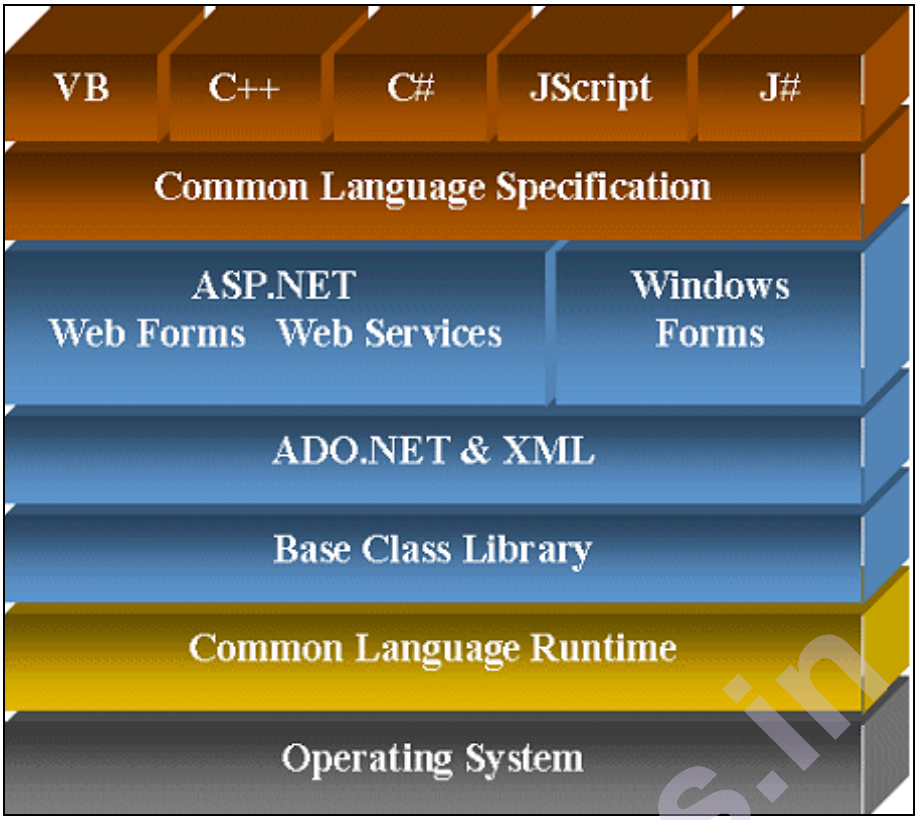
1. Create and Console Application with basics code.
2. Create the Application using different types of statements and loops.
3. Know about namespaces and assemblies and how to create the same.
4. Create console application using delegates and methods.

1.1 THE .NET FRAMEWORK

- Microsoft .NET is much more than XML Web services.
- At the heart of Microsoft .NET is the .NET Framework, consisting of the common language runtime and the class libraries.
- These two components provide the execution engine and programming APIs for building .NET applications.
- Applications compiled for the .NET Framework are not compiled directly to native code. Instead, they are compiled into an

intermediate language called Microsoft Intermediate Language (MSIL).

- When an application is run for the first time, the common language runtime just-in-time compiler compiles the MSIL code into native code before it is executed.
- The common language runtime is more than a simple JIT compiler; it is also responsible for providing low-level execution services, such as
  - garbage collection,
  - exception handling,
  - security services, and
  - Run time type-safety checking.
- Because of the common language runtime's role in managing execution, programs that target the .NET Framework are sometimes called "**managed**" applications.
- The .NET Framework also includes a set of classes for building applications that run on the common language runtime.
- These class libraries provide rich support for a wide range of tasks, including data access, security, file IO, XML manipulation, messaging, class reflection, XML Web services, user-interface construction, text processing, ASP.NET, and Microsoft Windows services.
- The most unique attribute of the .NET Framework is its support for multiple languages.
- It provides support for over 20 programming languages including Perl, Python, and COBOL.
- Relying on the common language runtime, code compiled with these compilers can interoperate.
- The .NET Framework is composed of the four extended applications named as four blue boxes—representing
  1. ASP.NET,
  2. Windows Forms,
  3. ADO.NET and
  4. XML, and subcomponents.



This book uses the Visual Basic language, which enables you to create readable, modern code. The .NET version of VB is similar in syntax to older flavors of VB that you may have encountered, including "classic" VB 6 and the Visual Basic for Applications (VBA) language often used to write macros in Microsoft Office programs.

1.1.1 C#, VB, and the .NET Languages

- This book uses the Visual Basic language, which enables you to create readable, modern code. The .NET version of VB is similar in syntax to older flavors of VB that you may have encountered, including "classic" VB 6 and the Visual Basic for Applications (VBA) language often used to write macros in Microsoft Office programs such as Word and Excel. However, you cannot convert classic VB into the .NET flavor of Visual Basic, just as you cannot convert C++ into C#.

- This book uses C#, Microsoft's .NET language of preference. C# resembles Java, JavaScript, and C++ in syntax, so programmers who have coded in one of these languages will quickly feel at home. Interestingly, VB and C# are quite similar. Though the syntax is different, both VB and C# use the .NET class library and are supported by the CLR. In fact, almost any block of C# code can be translated, line by line, into an equivalent block of VB code (and vice versa). An occasional language difference pops up, but for the most part, a developer who has learned one .NET language can move quickly and efficiently to another.

- There are even software tools that translate C# and VB code automatically (see **http://converter.telerik.com or http://tangiblesoftwaresolutions.com** for examples).

- In short, both VB and C# are elegant, modern languages that are ideal for creating the next generation of web applications.

1.1.2 Intermediate Language

- All the .NET languages are compiled into another lower-level language before the code is executed. This lower level language is the Common Intermediate Language (CIL, or just IL).

- The CLR, the engine of .NET, uses only IL code. Because all .NET languages are based on IL, they all have profound similarities. This is the reason that the VB and C# languages provide essentially the same features and performance.

- In fact, the languages are so compatible that a web page written with C# can use a VB component in the same way it uses a C# component, and vice versa.

- The .NET Framework formalizes this compatibility with something called the Common Language Specification (CLS).

- Essentially, the CLS is a contract that, if respected, guarantees that a component written in one .NET language can be used in all the others.

- One part of the CLS is the common type system (CTS), which defines the rules for data types such as strings, numbers, and arrays that are shared in all NET languages.

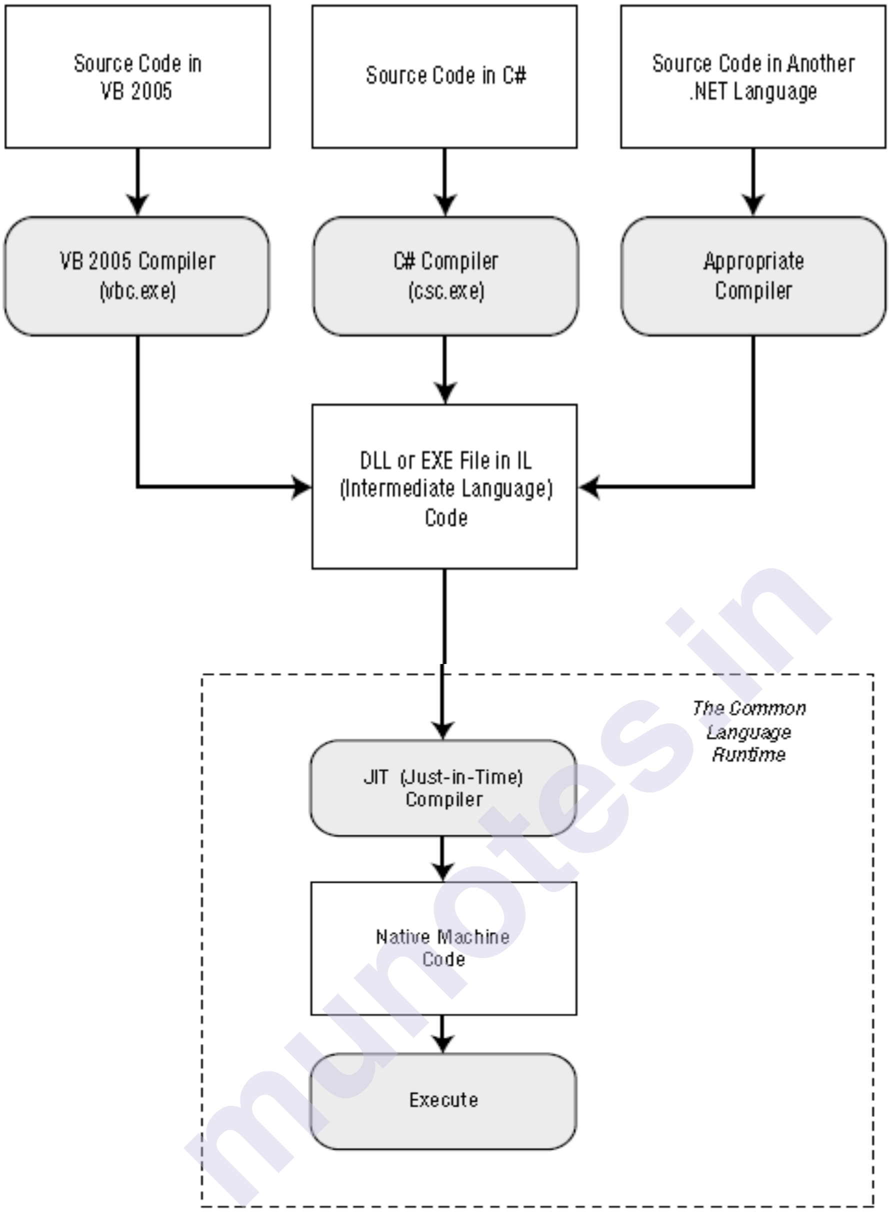
- The CLS also defines object-oriented ingredients such as classes, methods, events, and quite a bit more.

- For the most part, .NET developers don’t need to think about how the CLS works, even though they rely on it every day.

- Following Figure shows how the .NET languages are compiled to IL.

- Every EXE or DLL file that you build with a .NET language contains IL code.

- This is the file you deploy to other computers. In the case of a web application, you deploy your compiled code to a live web server.


Source Code in VB 2005
Source Code in C#
Source Code in Another .NET Language
VB 2005 Compiler (vbc.exe)
C# Compiler (csc.exe)
Appropriate Compiler
DLL or EXE File in IL (Intermediate Language) Code
The Common Language Runtime
JIT (Just-in-Time) Compiler
Native Machine Code
Execute

Language compilation in .NET

1.1.3 Components of .NET Framework

- The following pointers describe the components of the .Net framework 3.5 and the job they perform:

Common Language Runtime:

- It is built around CTS. It performs runtime tasks like memory management and garbage collection.

Base Class Libraries:

- It is a rich set of functional base classes.

Extended Class Libraries:

- Extended from base class libraries and designed to make it easier and faster to develop a specific application.

CLS: (Common Language Specification)

- It defines requirements for .net languages. It contains the specifications for the .Net supported languages and implementation of language integration.

CTS: (Common Type System)

- It provides guidelines for declaring, using and managing types at runtime and cross-language communication.

Metadata and Assemblies:

- Metadata is the binary information describing the program, which is either stored in a portable executable file(PE) or in the memory.
- Assembly is a logical unit consisting of the assembly manifest, type metadata, IL code and a set of resources like image files.

Multiple programming languages:

- It provides unified programming model for several languages.

Visual Studio .net:

- It is the IDE for coding with .net framework that spans the entire .net framework.

Windows & COM+ services:

- Today's requirements for today's .net framework SDK is Windows and COM+ services which provides facility to access the lower level system functionality.
- The class framework encapsulates the following functionality:
    - Data Access
    - Thread management
    - Interoperability with unmanaged code
    - Network protocol support
    - XML support
    - Web services support and Windows Forms support Access to assembly meta data

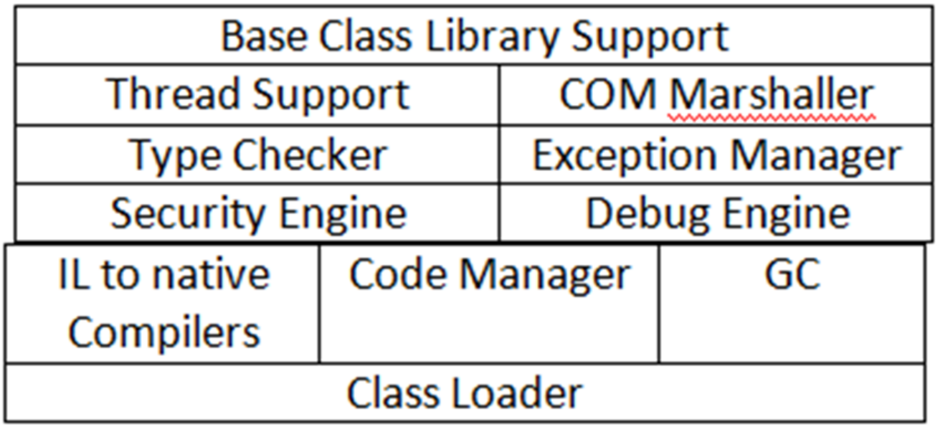
1.1.4 Common Language Runtime

- The CLR provides a rich level of support that simplifies application development and provides for better code reuse.
- The CLR provides a broad set of runtime services, including compilation, garbage collection and memory management.
- The CLR is built around the CTS, which defines standard, object-oriented data types that are used across all .NET programming languages.
- Code that runs under the control of the CLR is called managed code. Managed code allows the CLR to do the following.
  - Read meta data that describes the component interfaces and types
  - walk the code stack
  - handle exceptions
  - retrieve security information

Design Goals of the CLR:

1. **Simplify Development**
   - Define standards that promote code reuse
   - provide a broad range of services, including memory management and garbage collection
2. **Simplify application deployment**
   - Components use meta data instead of registration
   - support side-by-side, multiple component versions
   - command-line deployment (Xcopy) and uninstall(DEL)
3. **support development languages**
   - provide rich base classes for developer tools and languages
4. **support multiple languages**
   - define CTS that are used by all .NET languages
5. **enable convergence of programming models**
   - Build languages and tools on a common framework. For example, ASP .NET, VB .NET, and C# have access to the same base classes.

Structure of the CLR:



Base Class Library Support

Thread Support | COM Marshaller

Type Checker | Exception Manager

Security Engine | Debug Engine

IL to native Compilers | Code Manager | GC

Class Loader

1.1.5 The .NET Class Library

- The .NET class library is a giant repository of classes that provide prefabricated functionality for everything from reading an XML file to sending an e-mail message.

- If you've had any exposure to Java, you may already be familiar with the idea of a class library. However, the .NET class library is more ambitious and comprehensive than just about any other programming framework.

- Any .NET language can use the .NET class library's features by interacting with the right objects.

- This helps encourage consistency among different .NET languages and removes the need to install numerous components on your computer or web server.

- Some parts of the class library include features you'll never need to use in web applications (such as the classes used to create desktop applications with Windows interfaces). Other parts of the class library are targeted directly at web development.

- Still more classes can be used in various programming scenarios and aren't specific to web or Windows development.

- These include the base set of classes that define common variable types and the classes for data access, to name just a few.

1.1.6 Common Type System

- CTS defines standard, object oriented types and value types that are supported by all .NET programming languages.

- The CTS standards are what allow .NET to provide a unified programming model, and to support multiple languages.

- CTS is the first prerequisite for allowing languages to interoperate.
- This is easy to understand, if you consider that languages can only interoperate if they are based on the same system of types.
- In the past, type discrepancies have caused many interoperability problems, particularly for VB developers.
- So, CTS is an important new feature in the .NET framework. The CTS must support a range of languages, some of which are object-oriented, and some of which are not. Much has been made of the fact that COBOL is now a first class .NET language. COBOL is a procedural language, not an object-oriented one.
- The CTS provides two main types:
    - Value Types
    - Reference Types

      Value types are further classified into
        - Built-in types
    - User defined types

  Reference types are further classified into
    - Pointers
    - Objects
    - Interfaces
- Value types are simple data types that roughly correspond to simple bit patterns like integers and floats.
- In .NET, a value type derives from the System.Object namespace, and supports an interface that provides information about the kind of data that is stored, as well as the data value.
- They are useful for representing simple data types, and nay not-object user defined type, including enumerations.
- They are known as exact types which mean that they fully describe the value they hold.
- Reference types are also derived from the system.
- Object namespace, and may hold object references.
- They are self typing, which means that they describe their own interface.
- They are very specific to the type of object you are assigning.
- Once the reference is assigned, you expect to query the object reference according to what its interface provides.

Some of the primitive data types are:

- Bool
- Char
- int 8
- int 16
- float 32
- float 64
- unsigned int8
- unsigned int16

Type Safety:

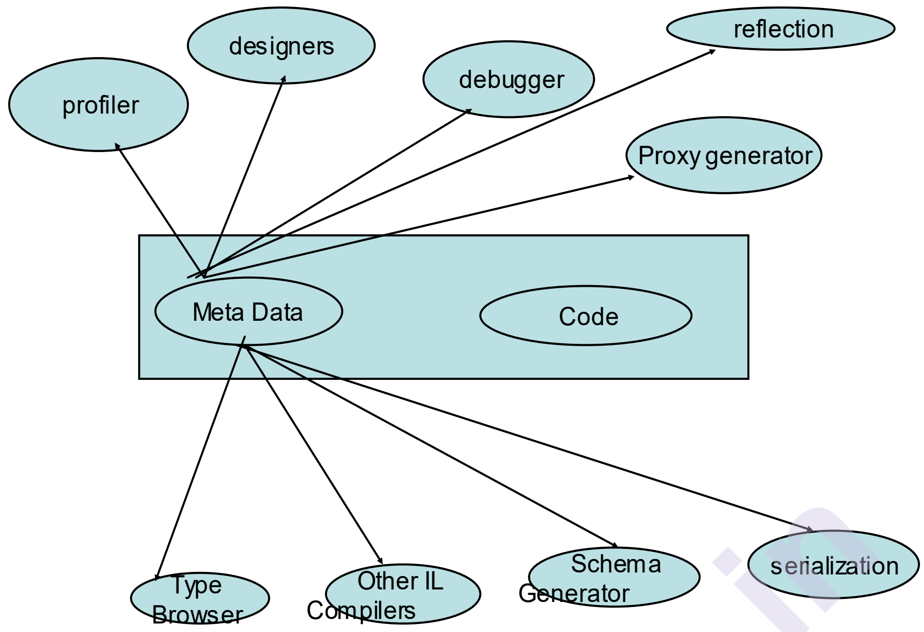
- The CTS promotes type safety, which in turn improves code stability.
- In .NET, type safety means that type definitions are completely known, and cannot be compromised.
- The CTS ensures that object references are strongly typed.
- It checks whether the array index out of range or not, whether the arithmetic exceptions are handled properly or not etc.

1.1.7 Metadata in .NET

Meta Data is organized information that the CLR uses to provide compile time and runtime services, including:

- Loading of class files
- Memory Management
- Debugging
- Object Browsing
- MSIL translation no Native Code
- NET components are self describing, because the Meta Data is stored as part of the compiled component known in .NET as an assembly.
- Combine this with the fact that .NET components do not require windows registry entries, and you can immediately appreciate why deployments are so much easier in .NET.

The figure below illustrates different Meta Data Consumers:



CONTENTS OF META DATA

- Description of the assembly (the deployment unit)
- identity: name, version and culture
- dependencies (other assemblies)
- security permission that the assembly requires to run
- Description of the Types
- Base classes and interfaces
- Custom attributes
- defined by the User
- defined by the Compiler
- defined by the Framework

1.1.8 Common Language Specification(CLS)

- The purpose of the NET framework is to define standards that makes it easier to write robust, secure and reusable code.

- The NET framework extends this concept by allowing any language to participate in the framework; so long as it conforms to the specifications embodied by the common Type System and the Common Language Specification.

- The common language Specification (CLS) defines conventions that languages must support in order to be interoperable within .NET.
- The CLS defines rules that range from naming conventions for interface members, to rules governing method overloading.
- In order to provide interoperation, a CLS-compliant language must obey the following conventions:
- Public identifiers are case- sensitive.
- Language must be able to resolve identifiers that are equivalent to their keywords.
- Stricter overloading rules; a given method name may refer to any number of methods, as long as each one differs in the number of parameters, or argument types.
- Properties and events must follow strict naming rules.
- All pointers must be managed, and reference must be typed; otherwise, they cannot be verified.

1.5 EXAMPLE PROGRAMS

Command Line Arguments

```
using System;
class Program
{
   public static void Main(string[] args)
   {
      Console.WriteLine("Enter The Name =");
      string name = Console.ReadLine();
      Console.WriteLine("Enter The Roll No.=");
      int rollno = Int32.Parse (Console.ReadLine());   // Convert.ToInt32()
      Console.WriteLine("Enter The Percentage :");
      double per = Double.Parse(Console.ReadLine());
      // Convert.ToDouble()
     // Console.WriteLine("Name=" + name);
    //  Console.WriteLine("Roll No.=" + rollno);
    //  Console.WriteLine("Percentage =" + per);
      Console.WriteLine("name={0}  \t  rollno={1}  \t  Percentage={2}",
name, rollno, per);
      Console.ReadKey(); // To hold the output
   }
}
```

Boxing and Unboxing

```
using System;
class Program
{
    public static void Main(string[] args)
    {
        int a = 10;   // Value Type
        object obj = a;   // Refence Type
        Console.WriteLine("a={0} ", a);
        Console.WriteLine("obj={0}", obj);
        int b = (int) obj ;
        Console.WriteLine("b={0}", b);
       Console.ReadKey(); // To hold the output
    }
}
```

Conditional Operator ?:

```
using System;
class Program
{
    public static void Main(string[] args)
    {

        Console.WriteLine("Enter The FOUR Numbers=");
        int a = Convert.ToInt32(Console.ReadLine());
        int b = Convert.ToInt32(Console.ReadLine());
        int c = Convert.ToInt32(Console.ReadLine());
        int d = Convert.ToInt32(Console.ReadLine());

        int large = a > b ? a : b;
        int larger = large > c ? large : c;
        int largest = larger > d ? larger : d;
        Console.WriteLine("Large={0}", largest);
        Console.ReadKey(); // To hold the output
    }
}
```

WAP to accept a number from the user and check whether it is positive, negative or zero.

```
using System;
class Program
{
   public static void Main(string[] args)
   {
      Console.WriteLine("Enter The Number");
      int num = Convert.ToInt32(Console.ReadLine());

      if (num > 0)
      {
         Console.WriteLine("Positive Number");
      }
      if (num < 0)
      {
         Console.WriteLine("Negative Number");
      }
      if (num == 0)
      {
         Console.WriteLine("Number is ZERO");
      }
      Console.ReadKey(); // To hold the output
   }
}
```

WAP to accept a number from the user and check whether it is even or odd.

```
using System;
class Program
{
   public static void Main(string[] args)
   {
      Console.WriteLine("Enter The Number");
      int num = Convert.ToInt32(Console.ReadLine());

      if (num % 2 == 0)
```

```
        {
            Console.WriteLine("Even Number");
        }
        else
        {
            Console.WriteLine("Odd Number");
        }
        Console.ReadKey(); // To hold the output
    }
}
```

Fall-through in Switch

```
using System;
class Program
{
    public static void Main(string[] args)
    {
        Console.WriteLine("Enter The Option");
        int opt = Convert.ToInt32(Console.ReadLine());

        switch (opt)
        {
            case 1:
                Console.WriteLine("ONE");
                goto case 3;
            case 2:
                Console.WriteLine("TWO");
                break;
            case 3:
                Console.WriteLine("THREE");
                break;
            case 4:
                Console.WriteLine("FOUR");
                break;
            default :
                Console.WriteLine("Invalid Option");
                break;
        }
        Console.ReadKey(); // To hold the output
    }
}
```

While Loop

```
using System;
class Program
{
    public static void Main(string[] args)
    {
        Console.WriteLine("Incremented Loop");
        int i = 1;
        while (i <= 10)
        {
            Console.Write(i + "\t");
            i++;
        }
        Console.WriteLine("\nDecremented Loop");
        int j = 10;
        while (j >= 1)
        {
            Console.Write(j + "\t");
            j--;
        }
        Console.ReadKey();
    }
}
```

do while Loop

```
int i = 1;
        do
        {
            Console.Write(i + "\t");
            i++;
        } while (i <= 10);

        int j = 10;
        do
        {
           Console.Write(j + "\t");
           j--;
        } while (j > 0);
```

For Loop

```
Console.WriteLine("Incremented Loop");
        for (int i = 1; i <= 10; i++)
        {
           Console.Write(i + "\t");
        }
        Console.WriteLine ("Decremented Loop")
        for(int j=10;j>=1;j--)
        {
           Console.Write (j+"\t");
        }
```

Foreach Loop

```
int[] num = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };


        for (int i = 0; i < 10; i++)
        {
           Console.Write(num[i] + "\t");
        }
        foreach(int s in num)
        {
           Console.Write(s + "\t");
        }
```

1.6 SUMMARY

This chapter 1 gives the basic syntax of C#. It discusses about variables, keywords, data types, creation of arrays, operators, control structures, methods debugging and few example programs. After learning the above topics, you can write many useful programs and built a strong foundation for larger programming projects.

1.7 EXERCISE: REVIEW QUESTIONS

Chapter 1

1) Write a note on .NET Framework.

2) Explain the data types in C#.

3) Explain the various operators in C#.

4) Discuss the various looping structures in C#.

5) Explain how arrays are created in C#.

6) What is a method? Explain its components.

7) How is debugging done in C#?

Program

1) WAP to accept a character from the user and check whether it is vowel or not.

2) WAP to accept two numbers from the user and display the greater number using if...else.

3) WAP to accept three numbers from the user and display the greater number.

4) WAP to accept a year from the user and display whether it is leap year or not.

5) WAP to accept a number from the user and display the factorial.

6) WAP to accept a number from the user and display sum of digits and reverse of that number.

7) WAP to accept a number from the user and check whether it is palindrom number or not.

8) WAP to accept a number from the user and check whether it is armstrong number or not.

9) WAP to accept a number from the user and check whether it is prime number or not.

10) WAP to accept two numbers from the user and display the GCD and LCM of that numbers.

11) WAP to accept a number from the user and check whether it is perfect number or not?

    a. Example : 6 -> 1+2+3 = 6   28 = 1 + 2 + 4 + 7 + 14 = 28

12) WAP to display all the prime numbers between 1 to 1000?

13) WAP to display all the armstrong numbers between 1 to 1000?

14) WAP to display the following output :

Numbers	Factorials
1	1
2	2
3	6
4	24
5	120
6	720

1.8 REFERENCE

1) The Complete Reference: C#

2) Visual C# 2012: How to program.

3) https://docs.microsoft.com/en-us/dotnet/csharp/

❄❄❄❄❄❄❄

2

THE C# LANGUAGE

Unit Structure

2.1 INTRODUCTION

- Before you can create an ASP.NET application, you need to choose a .NET language in which to program it.
- Both VB and C# are powerful, modern languages, and you won't go wrong using either of them to code your web pages.
- Often the choice is simply a matter of personal preference or your work environment. For example, if you've already programmed in a

language that uses C-like syntax (for example, Java), you'll probably be most comfortable with C#. Or if you've spent a few hours writing

Microsoft Excel macros in VBA, you might prefer the natural style of Visual Basic. Many developers become fluent in both. This chapter presents an overview of the C# language.

- You'll learn about the data types you can use, the operations you can perform, and the code you'll need to define functions, loops, and conditional logic.
- This chapter assumes that you have programmed before and are already familiar with most of these concepts—you just need to see how they're implemented in C#. If you've programmed with a similar language such as Java, you might find that the most beneficial way to use this chapter is to browse through it without reading every section.
- This approach will give you a general overview of C#.
- You can then return to this chapter later as a reference when needed. But remember, though you can program an ASP.NET application without mastering all the language details, this deep knowledge is often what separates the casual programmer from the true programming guru.

2.2 THE .NET LANGUAGES

- The .NET Framework ships with two core languages that are commonly used for building ASP.NET applications: C# and VB.
- These languages are, to a large degree, functionally equivalent. Microsoft has worked hard to eliminate language conflicts in the .NET Framework.
- These battles slow down adoption, distract from the core framework features, and makes it difficult for the developer community to solve problems together and share solutions.
- According to Microsoft, choosing to program in C# instead of VB is just a lifestyle choice and won't affect the performance, interoperability, feature set, or development time of your applications.
- Surprisingly, this ambitious claim is essentially true.
- .NET also allows other third-party developers to release languages that are just as feature-rich as C# or VB.
- These languages (which include Eiffel, Pascal, and even COBOL) "snap in" to the .NET Framework effortlessly.
- In fact, if you want to install another .NET language, all you need to do is copy the compiler to your computer and add a line to register it in a configuration file.

- Typically, a setup program would perform these steps for you automatically.
- Once installed, the new compiler can transform your code creations into a sequence of Intermediate Language (IL) instructions, just as the VB and C# compilers do with VB and C# code.
- IL is the only language that the Common Language Runtime (CLR) recognizes.
- When you create the code for an ASP.NET web form, it's changed into IL using the C# compiler (csc.exe) or the VB compiler (). Although you can perform the compilation manually, you're more likely to let ASP.NET handle it automatically when a web page is requested.

2.3 THE C# LANGUAGE

- New C# programmers are sometimes intimidated by the quirky syntax of the language, which includes special characters such as semicolons (;), curly braces ({}), and backward slashes (\).
- Fortunately, once you get accustomed to C#, these details will quickly melt into the background.
- In the following sections, you'll learn about four general principles you need to know about C# before you learn any other concepts.

Case Sensitivity

- Some languages are case-sensitive, while others are not. Java, C, C++, and C# are all examples of case-sensitive languages.
- VB is not. This difference can frustrate former VB programmers who don't realize that keywords, variables, and functions must be entered with the proper case.
- For example, if you try to create a conditional statement in C# by entering If instead of if, your code will not be recognized, and the compiler will flag it with an error when you try to build your application.

Commenting

- Comments are lines of descriptive text that are ignored by the compiler. C# provides two basic types of comments.
- The first type is the single-line comment. In this case, the comment starts with two forward slashes and continues for the entire current line:

```
// A single-line C# comment.
```

- Optionally, C# programmers can use /* and */ comment brackets to indicate multiple-line comments:

```
/* A multiple-line
C# comment. */
```

- This way, the code won't be executed, but it will still remain in your source code file if you need to refer to it or use it later.

Statement Termination

- C# uses a semicolon (;) as a statement-termination character.
- Every statement in C# code must end with this semicolon, except when you're defining a block structure. (Examples of such statements include methods, conditional statements, and loops, which are three types of code ingredients that you'll learn about later in this chapter.) By omitting the semicolon, you can easily split a statement of code over multiple physical lines.
- You just need to remember to put the semicolon at the end of the last line to end the statement.
- The following code snippet demonstrates four equivalent ways to perform the same operation (adding three numbers together):

```
// A code statement on a single line.
myValue = myValue1 + myValue2 + myValue3;
// A code statement split over two lines.
myValue = myValue1 + myValue2 +
myValue3;
// A code statement split over three lines.
myValue = myValue1 +
myValue2 +
myValue3;
// Two code statements in a row.
myValue = myValue1 + myValue2;
myValue = myValue + myValue3;
```

Blocks

- The C#, Java, and C languages all rely heavily on curly braces—parentheses with a little more attitude: {}.
- You can find the curly braces to the right of most keyboards (next to the P key); they share a key with the square brackets: [].
- Curly braces group multiple code statements together. Typically, you'll group code statements because you want them to be repeated in a loop, executed conditionally, or grouped into a function. These are all block structures, and you'll see all these techniques in this chapter.

- But in each case, the curly braces play the same role, which makes C# simpler and more concise than other languages that need a different syntax for each type of block structure.

```
{
// Code statements go here.
}
```

2.4 VARIABLES AND DATA TYPES

Variables

- As with all programming languages, you keep track of data in C# by using variables.
- Variables can store numbers, text, dates, and times, and they can even point to full-fledged objects.
- When you declare a variable, you give it a name and specify the type of data it will store.
- To declare a local variable, you start the line with the data type, followed by the name you want to use. A final semicolon ends the statement.

```
// Declare an integer variable named errorCode.
int errorCode;
// Declare a string variable named myName.
string myName;
```

The variables in C#, are categorized into the following types:

- Value types
- Reference types
- Object types

Value Type

- Value type variables can be assigned a value directly. They are derived from the class System.ValueType.
- The value types directly contain data. Some examples are int, char, and float, which stores numbers, alphabets, and floating point numbers, respectively. When you declare an int type, the system allocates memory to store the value.

For example if you type

Console.WriteLine(sizeof(int)),

you will get the output as 4, the bytes occupied by an integer.

Reference Type

- The reference types do not contain the actual data stored in a variable, but they contain a reference to the variables.
- In other words, they refer to a memory location. Using multiple variables, the reference types can refer to a memory location. If the data in the memory location is changed by one of the variables, the other variable automatically reflects this change in value.
- Example of **built-in** reference types are:
  - **object**,
  - **dynamic,** and
  - **string**.

Object Type

- The Object Type is the ultimate base class for all data types in C# Common Type System (CTS). Object is an alias for System.Object class.
- The object types can be assigned values of any other types, value types, reference types, predefined or user-defined types.
- However, before assigning values, it needs type conversion.

Data Types

- The types of data that a variable contain is called Datatype. A Datatype is a classification of things that share similar type of qualities or characteristics or behaviour .
- C# is strongly typed language so every variable and object must have a type.
- These are two types of data type in C#.

Primitive types or predefined

- Eg:-byte, short, int, float, double, long, char, bool, DateTime, string object etc.

Non-primitive types or user defined

- Eg:- class , struct, enum, interface, delegate, array.

Strings and Escaped Characters

- C# treats text a little differently than other languages such as VB. It interprets any embedded backslash (\) as the start of a special character sequence.
- For example, \n means add a new line (carriage return).
- The most useful character literals are as follows:
  - \" (double quote)

- \n (new line)
- \t (horizontal tab)
- \\ (backward slash)

You can also insert a special character based on its hex code by using the syntax \x250. This inserts a single character with hex value 250 (which is a character that looks like an upside-down letter a).

- Note that in order to specify the backslash character (for example, in a directory name), you require two slashes. Here's an example:

```
// A C# variable holding the path c:\MyApp\MyFiles
string path = "c:\\MyApp\\MyFiles";
```

Alternatively, you can turn off C# escaping by preceding a string with an @ symbol, as shown here: **string path = @"c:\MyApp\MyFiles";**

2.5 VARIABLES OPERATIONS

```
int number;
number = 4 + 2 * 3;// number will be 10.
number = (4 + 2) * 3;
// number will be 18.
```

- You can use all the standard types of variable operations in C#.
- When working with numbers, you can use various math symbols, as listed in Table below C# follows the conventional order of operations, performing exponentiation first, followed by multiplication and division and then addition and subtraction. You can also control order by grouping sub expressions with parentheses:

Operator Description Example

+ Addition 1 + 1 = 2

- Subtraction 5 - 2 = 3

*** Multiplication 2 * 5 = 10**

/ Division 5.0 / 2 = 2.5

% Gets the remainder left after integer division 7 % 3 = 1

- The operators above are designed for manipulating numbers. However, C# also allows you to use the addition operator (+) to join two strings:

```
// Join three strings together.
myName = firstName + " " + lastName;
In addition, C# provides special shorthand assignment operators.
Here are a few examples:
// Add 10 to myValue. This is the same as myValue = myValue + 10;
myValue += 10;
// Multiple myValue by 3. This is the same as myValue = myValue * 3;
myValue *= 3;
// Divide myValue by 12. This is the same as myValue = myValue / 12;
myValue /= 12;
```

2.6 KEYWORDS IN C#

- **Keywords** are predefined, reserved identifiers that have special meanings to the compiler.
- They cannot be used as identifiers in your program unless they include @ as a prefix. There are 77 keywords.
- Some of them are: is, base, checked, decimal, delegate, event, explicit, extern, fixed, for each, implicit, in, internal, is, lock ,object , override, params, read only, ref, sealed, stack, alloc, unchecked, unsafe, using.

Type Conversions

- Converting information from one data type to another is a fairly common programming task.
- For example, you might retrieve a user's text input that contains the number you want to use for a calculation.
- Or, you might need to take a calculated value and transform it into text you can display in a web page. Conversions are of two types: widening and narrowing. Widening conversions always succeed.
- For example, you can always convert a 32-bit integer into a 64-bit integer. You won't need any special code:

```
int mySmallValue;
long myLargeValue;
// Get the largest possible value that can be stored as a 32-bit integer.
// .NET provides a constant named Int32.MaxValue that provides this
number.
mySmallValue = Int32.MaxValue;
// This always succeeds. No matter how large mySmallValue is,
// it can be contained in myLargeValue.
myLargeValue = mySmallValue;
```

- On the other hand, narrowing conversions may or may not succeed, depending on the data.

- If you're converting a 32-bit integer to a 16-bit integer, you could encounter an error if the 32-bit number is larger than the maximum value that can be stored in the 16-bit data type.

- All narrowing conversions must be performed explicitly. C# uses an elegant method for explicit type conversion.

- To convert a variable, you simply need to specify the type in parentheses before the expression you're converting.

- The following code shows how to change a 32-bit integer to a 16-bit integer:

```
int count32 = 1000;

short count16;

// Convert the 32-bit integer to a 16-bit integer.

// If count32 is too large to fit, .NET will discard some of the

// information you need, and the resulting number will be incorrect.

count16 = (short)count32;
```

This process is called as Casting.

2.7 OBJECT-BASED MANIPULATION

- .NET is object-oriented to the core. In fact, even ordinary variables are really full-fledged objects in disguise.

- This means that common data types have the built-in smarts to handle basic operations (such as counting the number of characters in a string).

- Even better, it means you can manipulate strings, dates, and numbers in the same way in C# and in VB.

- You'll learn far more about objects in Chapter 3. But even now it's worth taking a peek at the object underpinnings in seemingly ordinary data types.

- For example, every type in the .NET class library includes a **ToString() method**.

- The default implementation of this method returns the class name.

- In simple variables, a more useful result is returned: the string representation of the given variable.

- The following code snippet demonstrates how to use the **ToString() method** with an integer:

```
string myString;

int myInteger = 100;

// Convert a number to a string. myString will have the contents
"100".

myString = myInteger.ToString();
```

2.7.1 The String Type

- One of the best examples of how class members can replace built-in functions is found with strings.

- In the past, every language has defined its own specialized functions for string manipulation.

- In .NET, however, you use the methods of the String class, which ensures consistency between all .NET languages.

- The following code snippet shows several ways to manipulate a string by using its object nature:

```
string myString = "This is a test string ";

myString = myString.Trim(); // = "This is a test string"

myString = myString.Substring(0, 4); // = "This"

myString = myString.ToUpper(); // = "THIS"

myString = myString.Replace("IS", "AT"); // = "THAT"

int length = myString.Length; // = 4
```

- The first few statements use built-in methods, such as Trim(), Substring(), ToUpper(), and Replace().

- These methods generate new strings, and each of these statements replaces the current myString with the new string object.

- The final statement uses a built-in Length property, which returns an integer that represents the number of characters in the string.

- Note that the Substring() method requires a starting offset and a character length.

- Strings use zero-based counting. This means that the first letter is in position 0, the second letter is in position 1, and so on.

- You'll find this standard of zero-based counting throughout .NET Framework for the sake of consistency.

Methods in System.String Class

Length() Returns the number of characters in the string (as an integer).

ToUpper() and ToLower()

Returns a copy of the string with all the characters changed to uppercase or lowercase characters.

Trim(), TrimEnd(), and TrimStart()

Removes spaces (or the characters you specify) from either end (or both ends) of a string.

PadLeft() and PadRight()

Adds the specified character to the appropriate side of a string as many times as necessary to make the total length of the string equal to the number you specify. For example, "Hi".PadLeft(5, '@') returns the string @@@Hi.

Insert()

Puts another string inside a string at a specified (zero-based) index position. For example, Insert(1, "pre") adds the string pre after the first character of the current string.

Remove()

Removes a specified number of characters from a specified position. For example, Remove(0, 1) removes the first character.

Replace()

Replaces a specified substring with another string. For example, Replace("a", "b") changes all a characters in a string into b characters.

Substring()

Extracts a portion of a string of the specified length at the specified location (as a new string). For example, Substring(0, 2) retrieves the first two characters.

StartsWith() and EndsWith()

Determines whether a string starts or ends with a specified substring. For example, StartsWith("pre") will return either true or false, depending on whether the string begins with the letters pre in lowercase.

2.7.2 The DateTime and TimeSpan Types

- The DateTime and TimeSpan data types also have built-in methods and properties.
- These class members allow you to perform three useful tasks:
  - Extract a part of a DateTime (for example, just the year) or convert a TimeSpan to a specific representation (such as the total number of days or total number of minutes).
  - Easily perform date calculations.
  - Determine the current date and time and other information (such as the day of the week or whether the date occurs in a leap year)
- For example, the following block of code creates a DateTime object, sets it to the current date and time, and adds a number of days.
- It then creates a string that indicates the year that the new date falls in (for example, 2012).

```
DateTime myDate = DateTime.Now;

myDate = myDate.AddDays(100);

string dateString = myDate.Year.ToString();
```

- The next example shows how you can use a TimeSpan object to find the total number of minutes between two DateTime objects:

```
DateTime myDate1 = DateTime.Now;

DateTime myDate2 = DateTime.Now.AddHours(3000);

TimeSpan difference;

difference = myDate2.Subtract(myDate1);

double numberOfMinutes;

numberOfMinutes = difference.TotalMinutes;
```

- The DateTime and TimeSpan classes also support the + and – arithmetic operators, which do the same work as the built-in methods. That means you can rewrite the example shown earlier like this:

```
// Adding a TimeSpan to a DateTime creates a new DateTime.

DateTime myDate1 = DateTime.Now;

TimeSpan interval = TimeSpan.FromHours(3000);

DateTime myDate2 = myDate1 + interval;

// Subtracting one DateTime object from another produces a TimeSpan.

TimeSpan difference;

difference = myDate2 - myDate1;
```

Now : Gets the current date and time.

Today: Gets the current date and leaves time set to 00:00:00.

Year, Date, Month, Hour, Minute, Second, and Millisecond

Returns one part of the DateTime object as an integer. For example, Month will return 12 for any day in December.

Add() and Subtract()

Adds or subtracts a TimeSpan from the DateTime. For convenience, these operations are mapped to the + and – operators, so you can use them instead when performing calculations with dates.

AddYears(), AddMonths(), AddDays(), AddHours(), AddMinutes(), AddSeconds(), AddMilliseconds()

Adds an integer that represents a number of years, months, and so on, and returns a new DateTime. You can use a negative integer to perform a date subtraction.

IsLeapYear()

Returns true or false depending on whether the specified year is a leap year.

ToString()

Returns a string representation of the current DateTime object. You can also use an overloaded version of this method that allows you to specify a parameter with a format string.

2.7.3 The Array Type

- Arrays also behave like objects in the world of .NET. (Technically, every array is an instance of the System.Array type.)
- For example, if you want to find out the size of a one-dimensional array, you can use the Length property or the GetLength() method, both of which return the total number of elements in an array:

```
int[] myArray = {1, 2, 3, 4, 5};
int numberOfElements;
numberOfElements = myArray.Length; // numberOfElements = 5
```

- You can also use the GetUpperBound() method to find the highest index number in an array. When calling GetUpperBound(), you supply a number that indicates what dimension you want to check.
- In the case of a one-dimensional array, you must always specify 0 to get the index number from the first dimension.
- In a two dimensional array, you can also use 1 for the second bound; in a three-dimensional array, you can also use 2 for the third bound; and so on.

The following code snippet shows GetUpperBound() in action:

```
int[] myArray = {1, 2, 3, 4, 5};
int bound;
// Zero represents the first dimension of an array.
bound = myArray. GetUpperBound(0); // bound = 4
```

- On a one-dimensional array, GetUpperBound() always returns a number that's one less than the length. That's because the first index number is 0.
- For example, the following code snippet uses GetUpperBound() to find the total number of rows and the total number of columns in a two-dimensional array:

```
// Create a 4x2 array (a grid with four rows and two columns).
int[,] intArray = {{1, 2}, {3, 4}, {5, 6}, {7, 8}};
int rows = intArray.GetUpperBound(0) + 1; // rows = 4
int columns = intArray.GetUpperBound(1) + 1; // columns = 2
```

Length

Returns an integer that represents the total number of elements in all dimensions of an array. For example, a 3 × 3 array has a length of 9.

GetLowerBound() and GetUpperBound()

Determines the dimensions of an array. As with just about everything in.NET, you start counting at zero (which represents the first dimension).

Clear()

Empties part or all of an array's contents, depending on the index values that you supply. The elements revert to their initial empty values (such as 0 for numbers).

IndexOf () and LastIndexOf ()

Searches a one-dimensional array for a specified value and returns the index number. You cannot use this with multidimensional arrays.

Sort()

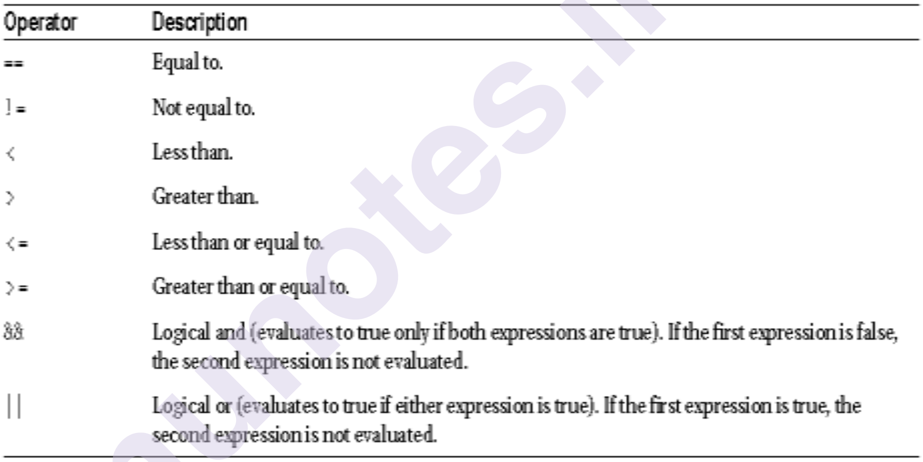
Sorts a one-dimensional array made up of comparable data such as strings or numbers.

Reverse ()

Reverses a one-dimensional array so that its elements are backward, from last to first.

2.8 CONDITIONAL LOGIC

- Conditional logic means deciding which action to take based on user input, external conditions, or other information—is the heart of programming.
- All conditional logic starts with a condition: a simple expression that can be evaluated to true or false.
- Your code can then make a decision to execute different logic depending on the outcome of the condition.
- To build a condition, you can use any combination of literal values or variables along with logical operators. Table below lists the basic logical operators.



Operator	Description
==	Equal to.
!=	Not equal to.
<	Less than.
>	Greater than.
<=	Less than or equal to.
>=	Greater than or equal to.
&&	Logical and (evaluates to true only if both expressions are true). If the first expression is false, the second expression is not evaluated.
||	Logical or (evaluates to true if either expression is true). If the first expression is true, the second expression is not evaluated.

- You can use all the comparison operators with any numeric types. With string data types, you can use only the equality operators (== and !=). C# doesn't support other types of string comparison operators.

```
int result;

result = String.Compare("apple", "attach"); // result = -1

result = String.Compare("apple", "all"); // result = 1

result = String.Compare("apple", "apple"); // result = 0

// Another way to perform string comparisons.

string word = "apple";

result = word.CompareTo("attach"); // result = -1
```

2.8.1 Conditional Statements

The if Statement

- The if statement is the powerhouse of conditional logic, able to evaluate any combination of conditions and deal with multiple and different pieces of data.
- Here's an example with an if statement that features two else conditions:
- An if block can have any number of conditions. If you test only a single condition, you don't need to include any else blocks.

```
if (myNumber > 10)
{
// Do something.
}
else if (myString == "hello")
{
// Do something.
}
else
{
// Do something.
}
```

- Keep in mind that the if construct matches one condition at most.
- For example, if myNumber is greater than 10, the first condition will be met.
- That means the code in the first conditional block will run, and no other conditions will be evaluated.
- Whether my String contains the text hello becomes irrelevant, because that condition will not be evaluated. If you want to check both conditions, don't use an else block—instead, you need two if blocks back-to-back, as shown here:

```
if (myNumber > 10)
{
// Do something.
}
if (myString == "hello")
{
// Do something.
}
```

The switch Statement

- C# also provides a switch statement that you can use to evaluate a single variable or expression for multiple possible values. The only limitation is that the variable you're evaluating must be an integer-based data type, a bool, a char, a string, or a value from an enumeration. Other data types aren't supported.

- In the following code, each case examines the myNumber variable and tests whether it's equal to a specific integer:

switch (myNumber)
{
case 1:
// Do something.
break;
case 2:
// Do something.
break;
default:
// Do something.
break;
}
	• You'll notice that the C# syntax inherits the convention of C/C++ programming, which requires that every branch in a switch statement be ended by a special break keyword.
	• If you omit this keyword, the compiler will alert you and refuse to build your application.
	• The only exception is if you choose to stack multiple case statements directly on top of each other with no intervening code.
	• This allows you to write one segment of code that handles more than one case. Here's an example:

switch (myNumber)
{
case 1:
case 2:
// This code executes if myNumber is 1 or 2.
break;
default:
// Do something.
break;
}
	• Unlike the if statement, the switch statement is limited to evaluating a single piece of information at a time.
	• However, it provides a cleaner, clearer syntax than the if statement when you need to test a single variable.

2.8 LOOPS IN C#

Introduction

- Loops allow you to repeat a segment of code multiple times. C# has three basic types of loops. You choose the type of loop based on the type of task you need to perform. Your choices are as follows:

- You can loop a set number of times with a for loop.
- You can loop through all the items in a collection of data by using a foreach loop.
- You can loop while a certain condition holds true with a while or do...while loop.
- The for and foreach loops are ideal for chewing through sets of data that have known, fixed sizes.
- The while loop is a more flexible construct that allows you to continue processing until a complex condition is met. The while loop is often used with repetitive tasks or calculations that don't have a set number of iterations.

2.8.1 The for Loop

- The for loop is a basic ingredient in many programs. It allows you to repeat a block of code a set number of times, using a built-in counter.
- To create a for loop, you need to specify a starting value, an ending value, and the amount to increment with each pass. Here's one example:

```
for (int i = 0; i < 10; i++)
{
// This code executes ten times.
System.Diagnostics.Debug.Write(i);
}
```

- You'll notice that the for loop starts with parentheses that indicate three important pieces of information. The first portion (int i = 0) creates the counter variable (i) and sets its initial value (0).
- The third portion (i++) increments the counter variable. In this example, the counter is incremented by 1 after each pass.
- That means i will be equal to 0 for the first pass, equal to 1 for the second pass, and so on. However, you could adjust this statement so that it decrements the counter (or performs any other operation you want).

```
string[] stringArray = {"one", "two", "three"};
for (int i = 0; i < stringArray.Length; i++)
{
System.Diagnostics.Debug.Write(stringArray[i] + " "); }
```

- The middle portion (i < 10) specifies the condition that must be met for the loop to continue. This condition is tested at the start of every pass through the block. If i is greater than or equal to 10, the condition will evaluate to false, and the loop will end.

2.8.2 The foreach Loop

- C# also provides a foreach loop that allows you to loop through the items in a set of data.

- With a foreach loop, you don't need to create an explicit counter variable.

- Instead, you create a variable that represents the type of data for which you're looking. Your code will then loop until you've had a chance to process each piece of data in the set.

- The foreach loop is particularly useful for traversing the data in collections and arrays.

- For example, the next code segment loops through the items in an array by using foreach.

```
string[] stringArray = {"one", "two", "three"};

foreach (string element in stringArray)

{

// This code loops three times, with the element variable set
to

// "one", then "two", and then "three".

System.Diagnostics.Debug.Write(element + " ");
```

- This code has exactly the same effect as the example in the previous section, but it's a little simpler:

- In this case, the foreach loop examines each item in the array and tries to convert it to a string. Thus, the foreach loop defines a string variable named element.

- If you used a different data type, you'd receive an error.

- The foreach loop has one key limitation: it's read-only.

- For example, if you wanted to loop through an array and change the values in that array at the same time, foreach code wouldn't work.

```
int[] intArray = {1,2,3};

foreach (int num in intArray)

{

num += 1;

}
```

- Here’s an example of some flawed code:

2.8.3 The While Loop

- Finally, C# supports a while loop that tests a specific condition before or after each pass through the loop.
- When this condition evaluates to false, the loop is exited. Here’s an example that loops ten times.
- At the beginning of each pass, the code evaluates whether the counter (i) is less than some upper limit (in this case, 10). If it is, the loop performs iteration.

```
int i = 0;
while (i < 10)
{
i += 1;
// This code executes ten times. }
```

- You can also place the condition at the end of the loop by using the do...while syntax. In this case, the condition is tested at the end of each pass through the loop:

```
int i = 0;
do
{
i += 1;
// This code executes ten times.
}
while (i < 10);
```

2.9 METHODS IN C#

- Methods are the most basic building block you can use to organize your code.
- Essentially, a method is a named grouping of one or more lines of code.
- Ideally, each method will perform a distinct, logical task.
- By breaking down your code into methods, you not only simplify your life, but also make it easier to organize your code into classes and step into the world of object-oriented programming.
- When you declare a method in C#, the first part of the declaration specifies the data type of the return value, and the second part indicates the method name.
- If your method doesn't return any information, you should use the void keyword instead of a data type at the beginning of the declaration.

```
// This method doesn't return any information.
void MyMethodNoReturnedData()
{
// Code goes here.
}
// This method returns an integer.
int MyMethodReturnsData()
{
// As an example, return the number 10.
return 10;
}
```

- Notice that the method name is always followed by parentheses. This allows the compiler to recognize that it's a method.
- In this example, the methods don't specify their accessibility. This is just a common C# convention. You're free to add an accessibility keyword (such as public or private), as shown here:

```
private void MyMethodNoReturnedData()
{
// Code goes here.
}
```

- The accessibility determines how different classes in your code can interact.
- Private methods are hidden from view and are available only locally, whereas public methods can be called by all the other classes in your application.
- To really understand what this means, you'll need to read the next chapter, which discusses accessibility in more detail.
- Invoking your methods is straightforward—you simply type the name of the method, followed by parentheses.
- If your method returns data, you have the option of using the data it returns or just ignoring it:

```
// This call is allowed.

MyMethodNoReturnedData();

// This call is allowed.

MyMethodReturnsData();

// This call is allowed.

int myNumber;

myNumber = MyMethodReturnsData();

// This call isn't allowed.

// MyMethodNoReturnedData() does not return any
information.

myNumber = MyMethodNoReturnedData();
```

2.9.1 Parameters

- Methods can also accept information through parameters. Parameters are declared in a similar way to variables.
- By convention, parameter names always begin with a lowercase letter in any language.

- Here’s how you might create a function that accepts two parameters and returns their sum:

```
private int AddNumbers(int number1, int number2)
{
        return number1 + number2;

}
```

- When calling a method, you specify any required parameters in parentheses or use an empty set of parentheses if no parameters are required:

```
// Call a method with no parameters.
MyMethodNoReturnedData();
// Call a method that requires two integer parameters.
MyMethodNoReturnedData2(10, 20);
// Call a method with two integer parameters and an
integer return value.
int returnValue = AddNumbers(10, 10);
```

2.9.2 Method Overloading

- C# supports method overloading, which allows you to create more than one method with the same name but with a different set of parameters.
- When you call the method, the CLR automatically chooses the correct version by examining the parameters you supply.
- This technique allows you to collect different versions of several methods together.
- For example, you might allow a database search that returns an array of Product objects representing records in the database.
- Rather than create three methods with different names depending on the criteria, such as GetAllProducts(), GetProductsInCategory(), and GetActiveProducts(), you could create three versions of the GetProducts() method.
- Each method would have the same name but a different signature, meaning it would require different parameters.
- This example provides two overloaded versions for the GetProductPrice() method:

```
private decimal GetProductPrice(int ID)
{
// Code here.
}
private decimal GetProductPrice(string name)
{
// Code here.
}
// And so on...
```

- Now you can look up product prices based on the unique product ID or the full product name, depending on whether you supply an integer or string argument:

```
decimal price;
// Get price by product ID (the first version).
price = GetProductPrice(1001);
// Get price by product name (the second version).
price = GetProductPrice("DVD Player");
```

- You cannot overload a method with versions that have the same signature that is, the same number of parameters and parameter data types because the CLR will not be able to distinguish them from each other.

- When you call an overloaded method, the version that matches the parameter list you supply is used. If no version matches, an error occurs.

Optional and Named Parameters

- Method overloading is a time-honored technique for making methods more flexible, so you can call them in a variety of ways.
- C# also has another feature that supports the same goal: optional parameters.
- An optional parameter is any parameter that has a default value.
- If your method has normal parameters and optional parameters, the optional parameters must be placed at the end of the parameter list.
- Here's an example of a method that has a single optional parameter:

2.11 DELEGATES

- Delegates allow you to create a variable that “points” to a method. You can then use this variable at any time to invoke the method.
- Delegates help you write flexible code that can be reused in many situations.
- They’re also the basis for events, an important .NET concept that you’ll consider in the next chapter.
- The first step when using a delegate is to define its signature.
- The signature is a combination of several pieces of information about a method: its return type, the number of parameters it has, and the data type of each parameter.

```
private string GetUserName(int ID, bool useShortForm = false)
{
// Code here.
}
// Explicitly set the useShortForm parameter.
name = GetUserName(401, true);
// Don't set the useShortForm parameter, and use the default
value (false).
name = GetUserName(401);
```

- A delegate variable can point only to a method that matches its specific signature.
- In other words, the method must have the same return type, the same number of parameters, and the same data type for each parameter as the delegate.
- For example, if you have a method that accepts a single string parameter and another method that accepts two string parameters, you’ll need to use a separate delegate type for each method.
- To consider how this works in practice, assume that your program has the following method:

```
private string TranslateEnglishToFrench(string english)

{

// Code goes here.

}
```

- This method accepts a single string argument and returns a string.

- With those two details in mind, you can define a delegate that matches this signature. Here's how you would do it:

- Notice that the name you choose for the parameters and the name of the delegate don't matter.

```
private delegate string StringFunction(string inputString);
```

- The only requirement is that the data types for the return value and parameters match exactly.

- Once you've defined a type of delegate, you can create and assign a delegate variable at any time. Using the String Function delegate type, you could create a delegate variable like this:

- Using your delegate variable, you can point to any method that has the matching signature. In this example, the StringFunction delegate type requires one string parameter and returns a string. Thus, you can use the functionReference variable to store a reference to the TranslateEnglishToFrench() method

```
string frenchString;
frenchString = functionReference("Hello");
```

- Now that you have a delegate variable that references a method, you can invoke the method through the delegate. To do this, you just use the delegate name as though it were the method name:

- In the previous code example, the method that the function Reference delegate points to will be invoked with the parameter value "Hello", and the return value will be stored in the French String variable.

```
StringFunction functionReference;
functionReference = TranslateEnglishToFrench;
```

- The following code shows all these steps—creating a delegate variable, assigning a method, and calling the method—from start to finish:

2.12 SUMMARY:

This chapter 2 gives the basic syntax of OOP in C#. It discusses about class, methods, constructors, destructor, method overloading and few example programs. After learning the above topics, you can write many useful programs and built a strong foundation for larger programming projects.

2.13 QUESTIONS

1) Explain OOP in C#.
2) Explain class and its member in C#.
3) Explain the methods in C#.
4) Explain constructor with example in C#.
5) Explain method overloading with example in C#.
6) Explain properties and indexer in C#?
7) Explain inheritance with example.
8) Explain method overriding with example.
9) Explain abstract class.
10) Explain Interface with example.
11) Explain structure with example.

2.14 REFERENCES:

1) The Complete Reference: C#

2) Visual C# 2012: How to program.

3) https://docs.microsoft.com/en-us/dotnet/csharp/

❄❄❄❄❄❄

3

TYPES, OBJECTS, AND NAMESPACES

Unit Structure

3.1 INTRODUCTION

- In this chapter, you'll learn how objects are defined and how you manipulate them in your code. Taken together, these concepts are the basics of what's commonly called object-oriented programming.
- This chapter explains objects from the point of view of the .NET Framework.
- It doesn't rehash the typical object-oriented theory, because countless excellent programming books cover the subject. Instead, you'll see the types of objects .NET allows, how they are constructed, and how they fit into the larger framework of namespaces and assemblies.

3.2 THE BASICS ABOUT CLASSES

- Class and Object are the basic concepts of Object-Oriented Programming which revolve around the real-life entities.
- A class is a user-defined blueprint or prototype from which objects are created.
- Basically, a class combines the fields and methods (member function which defines actions) into a single unit.
- In C#, classes support polymorphism, inheritance and also provide the concept of derived classes and base classes.
- Generally, a class declaration contains only keyword class, followed by an identifier(name) of the class.
- But there are some optional attributes that can be used with class declaration according to the application requirement.
- In general, class declarations can include these components, in order:

Modifiers: A class can be public or internal etc. By default modifier of class is internal.

Keyword class: A class keyword is used to declare the type class.

Class Identifier: The variable of type class is provided. The identifier(or name of class) should begin with an initial letter which should be capitalized by convention.

Base class or Super class: The name of the class's parent (superclass), if any, preceded by the : (colon). This is optional.

Interfaces: A comma-separated list of interfaces implemented by the class, if any, preceded by the : (colon). A class can implement more than one interface. This is optional.

Body: The class body is surrounded by { } (curly braces).

Note:

Constructors in class are used for initializing new objects. Fields are variables that provide the state of the class and its objects, and methods are used to implement the behavior of the class and its objects.

Syntax :

```
//[access modifier] - [class] - [identifier]
public class Customer
{
  // Fields, properties, methods and events go here...
}
```

3.2.1 Objects in C#

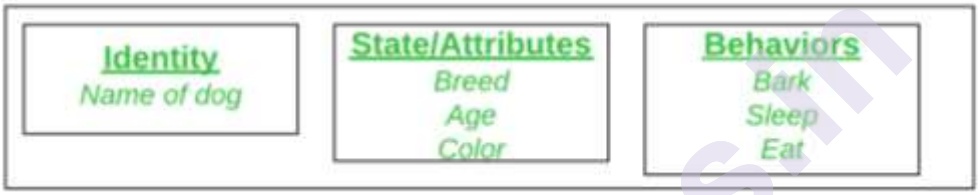
- It is a basic unit of Object-Oriented Programming and represents the real-life entities.
- A typical C# program creates many objects, which as you know, interact by invoking methods. An object consists of :

State: It is represented by attributes of an object. It also reflects the properties of an object.

Behavior: It is represented by methods of an object. It also reflects the response of an object with other objects.

Identity: It gives a unique name to an object and enables one object to interact with other objects.

Consider Dog as an object and see the below diagram for its identity, state, and behavior.

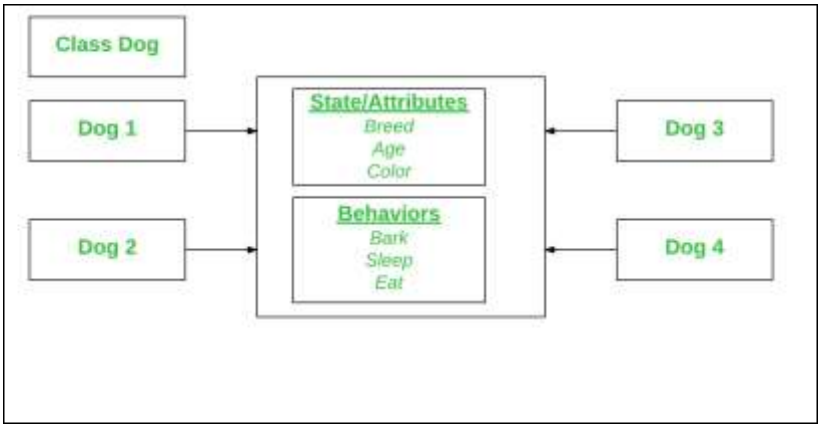


- Objects correspond to things found in the real world. For example, a graphics program may have objects such as “circle”, “square”, “menu”.
- An online shopping system might have objects such as “shopping cart”, “customer”, and “product”.

- **Declaring Objects (Also called instantiating a class)**

- When an object of a class is created, the class is said to be instantiated.

- All the instances share the attributes and the behaviour of the class. But the values of those attributes, i.e. the state are unique for each object. A single class may have any number of instances.



Syntax:

```
Classname objectname = new classname();
```

Example 1:

```
using System;
public class Student
{
int id;//data member (also instance variable)
String name;//data member(also instance variable)
   public static void Main(string[] args)
      {
         Student s1 = new Student();//creating an object of Student
         s1.id = 101;
         s1.name = "Sonoo Jaiswal";
         Console.WriteLine(s1.id);
         Console.WriteLine(s1.name);
       }  }
```

Example 2: Initialize and Display data through method

```
using System;
  public class Student
   {
      public int id;
      public String name;
      public void insert(int i, String n)
      {
         id = i;
         name = n;
      }
      public void display()
      {
         Console.WriteLine(id + " " + name);
      }
  }
  class TestStudent{
      public static void Main(string[] args)
       {
          Student s1 = new Student();
          Student s2 = new Student();
          s1.insert(101, "Ajeet");
          s2.insert(102, "Tom");
          s1.display();
          s2.display();
        }   }
```

3.2.2 Constructors

- Constructors are methods that are called when the object is first created. To create an object, the constructor call is preceded by the keyword “new”.
- The process of doing this is called instantiation.
- An object is then referred to as an instance of its class. They are often used to initialize the data of an object.
- A constructor has the same name as the name of its type (name of class).
- Its method signature includes only the method name and its parameter list; it does not include a return type.
- Objects are allocated on the heap (a memory region allocated for the program).
- Objects must be created with new Eg. **Stack stk = new Stack(50);**
- If you don't provide a constructor for your class, C# creates one by default that instantiates the object and sets member variables to the default values.
- If a constructor was declared, no default constructor is generated.

How constructors are different from a normal member function?

- A constructor is different from normal functions in following ways:
- Constructor has same name as the class itself
- Constructors don’t have return type
- A constructor is automatically called when an object is created.
- It must be placed in public section of class.
- If we do not specify a constructor, C++ compiler generates a default constructor for object (expects no parameters and has an empty body).

In C#, constructors can be divided into 5 types

1) Default Constructor
2) Parameterized Constructor
3) Copy Constructor
4) Static Constructor
5) Private Constructor

1) Default Constructor in C#

- A constructor without any parameters is called a default constructor; in other words, this type of constructor does not take parameters.
- The drawback of a default constructor is that every instance of the class will be initialized to the same values and it is not possible to initialize each instance of the class with different values.
- The default constructor initializes:

All numeric fields in the class to zero.
All string and object fields to null.

```
using System;
namespace DefaultConstructor
{
  class addition
  {
    int a, b;
    public addition()   //default constructor
    {
      a = 100;
      b = 175;
    }
    public static void Main()
    {
      addition obj = new addition(); //an object is created , constructor is called
      Console.WriteLine(obj.a);
      Console.WriteLine(obj.b);
      Console.Read();
    }
  }
}
```

2) Parameterized Constructor in C#

- A constructor with at least one parameter is called a parameterized constructor.
- The advantage of a parameterized constructor is that you can initialize each instance of the class with a different value.

```
using System;
namespace Constructor
{
   class paraconstrctor
   {
     public  int a, b;
     public paraconstrctor(int x, int y)   // declaring Parameterized
Constructor with int x, y parameter
      {
         a = x;
         b = y;
      }
  }
   class MainClass
   {
      static void Main()
      {
         paraconstrctor v = new paraconstrctor(100, 175);   // Creating
object of Parameterized Constructor and int values
         Console.WriteLine("-----------parameterized          constructor
example by vithal wadje--------------");
         Console.WriteLine("\t");
         Console.WriteLine("value of a=" + v.a );
         Console.WriteLine("value of b=" + v.b);
         Console.Read();
      }
   }
}
```

3) Copy Constructor in C#

- The constructor which creates an object by copying variables from another object is called a copy constructor.
- The purpose of a copy constructor is to initialize a new instance to the values of an existing instance.

```
public employee(employee emp)

{

   name=emp.name;

   age=emp.age;

}
```

The copy constructor is invoked by instantiating an object of type employee and bypassing it the object to be copied. **Example**

```
employee emp1=new  employee (emp2);
```

```
using System;

namespace copyConstructor

{

   class employee

   {

      private string name;

      private int age;

      public employee(employee emp)   // declaring Copy constructor.

      {

         name = emp.name;

         age = emp.age;

      }

      public employee(string name, int age)  // Instance constructor.

      {

         this.name = name;

         this.age = age;

      }

      public string Details     // Get details of employee

      {
          get
          {
             return  " The age of " + name +" is "+ age.ToString();
          }
       }
    }
    class empdetail
    {
       static void Main()
       {
          employee emp1 = new employee("Vithal", 23);  // Create a new
employee object.
          employee emp2 = new employee(emp1);           // here is emp1
details is copied to emp2.
          Console.WriteLine(emp2.Details);
          Console.ReadLine();
       }
    }}
```

4) Static Constructor in C#

- When a constructor is created using a static keyword, it will be invoked only once for all of the instances of the class and it is invoked during the creation of the first instance of the class or the first reference to a static member in the class.
- A static constructor is used to initialize static fields of the class and to write the code that needs to be executed only once.

Some key points of a static constructor are:

- A static constructor does not take access modifiers or have parameters.
- A static constructor is called automatically to initialize the class before the first instance is created or any static members are referenced.
- A static constructor cannot be called directly.
- The user has no control over when the static constructor is executed in the program.
- A typical use of static constructors is when the class is using a log file and the constructor is used to write entries to this file.

```
using System;
namespace staticConstructor
{
   public class employee
   {
      static employee() // Static constructor
      declaration{Console.WriteLine("The static constructor ");
   }
   public static void Salary()
   {
      Console.WriteLine();
      Console.WriteLine("The Salary method");
   }
}
class details
{
   static void Main()
   {
      Console.WriteLine("----------Static constructor example by vithal
wadje----------");
      Console.WriteLine();
      employee.Salary();
      Console.ReadLine();
   }
  }
}
```

5) Private Constructor in C#

- When a constructor is created with a private specifier, it is not possible for other classes to derive from this class, neither is it possible to create an instance of this class.
- They are usually used in classes that contain static members only. Some key points of a private constructor are:
- One use of a private constructor is when we have only static members.
- It provides an implementation of a singleton class pattern

- Once we provide a constructor that is either private or public or any, the compiler will not add the parameter-less public constructor to the class.

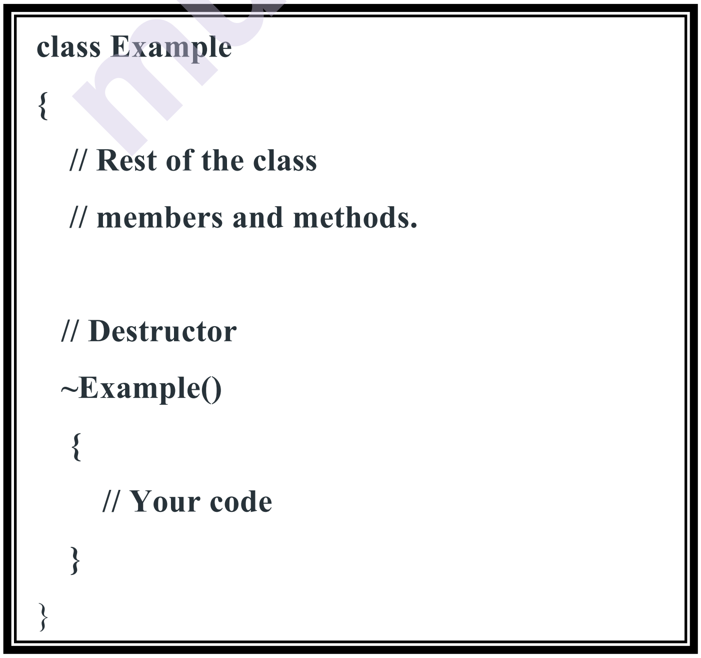
```
using System;
namespace defaultConstructor
{
   public class Counter
   {
      private Counter()   //private constructor declaration
      {
      }
      public static int currentView;
      public static int visitedCount()
      {
         return ++ currentView;
      }
   }
   class viewCountedetails
   {
      static void Main()
      {
         // Counter aCounter = new Counter();   // Error
         Console.WriteLine("-------Private constructor example by
vithal wadje----------");
         Console.WriteLine();
         Counter.currentView = 500;
         Counter.visitedCount();
         Console.WriteLine("Now the view count is: {0}",
Counter.currentView);
         Console.ReadLine();
      }
   }
}
```

3.2.3 Destructors

- Destructors in C# are methods inside the class used to destroy instances of that class when they are no longer needed.
- The Destructor is called implicitly by the .NET Framework's Garbage collector and therefore programmer has no control as when to invoke the destructor.
- An instance variable or an object is eligible for destruction when it is no longer reachable.

Important Points:

- A Destructor is unique to its class i.e. there cannot be more than one destructor in a class.
- A Destructor has no return type and has exactly the same name as the class name (Including the same case).
- It is distinguished apart from a constructor because of the Tilde symbol (~) prefixed to its name.
- A Destructor does not accept any parameters and modifiers.
- It cannot be defined in Structures. It is only used with classes.
- It cannot be overloaded or inherited.
- It is called when the program exits.
- Internally, Destructor called the Finalize method on the base class of object.



```
class Example
{
   // Rest of the class
   // members and methods.


  // Destructor
  ~Example()
   {
      // Your code
   }
}
```

3.2.4 Static Data Members and Member Functions

- We can define class members as static using the static keyword.
- When we declare a member of a class as static, it means no matter how many objects of the class are created, there is only one copy of the static member.
- The keyword static implies that only one instance of the member exists for a class. Static variables are used for defining constants because their values can be retrieved by invoking the class without creating an instance of it.
- Static variables can be initialized outside the member function or class definition. You can also initialize static variables inside the class definition.

```
using System;
namespace StaticVarApplication {
  class StaticVar {
    public static int num;

    public void count() {
      num++;
    }
    public int getNum() {
      return num;
    }
  }
  class StaticTester {
    static void Main(string[] args) {
      StaticVar s1 = new StaticVar();
      StaticVar s2 = new StaticVar();
      s1.count();
      s1.count();
      s1.count();
      s2.count();
      s2.count();
      s2.count();
      Console.WriteLine("Variable num for s1: {0}", s1.getNum());
      Console.WriteLine("Variable num for s2: {0}", s2.getNum());
      Console.ReadKey();
    }
  }
}
```

Static Method

- A static method in C# is a method that keeps only one copy of the method at the Type level, not the object level.
- That means, all instances of the class share the same copy of the method and its data. The last updated value of the method is shared among all objects of that Type.
- Static methods are called by using the class name, not the instance of the class.

```
class StaticDemo
{
    public static void withoutObj()
    {
        Console.WriteLine("Hello");
    }
     static void Main()
    {
        Program. withoutObj();
        Console.ReadKey();
    }
}
```

Using Static Method

- Usually we define a set of data members for a class and then every object of that class will have a separate copy of each of those data members.

```
class Program
  {
      public int myVar;  //a non-static field
        static void Main()
       {
          StaticDemo p1 = new StaticDemo();  //an object of class
          p1.myVar = 10;
          p1.withoutObj();
          Console.WriteLine(p1.myVar);
          Console.ReadKey();
       }
  }
```

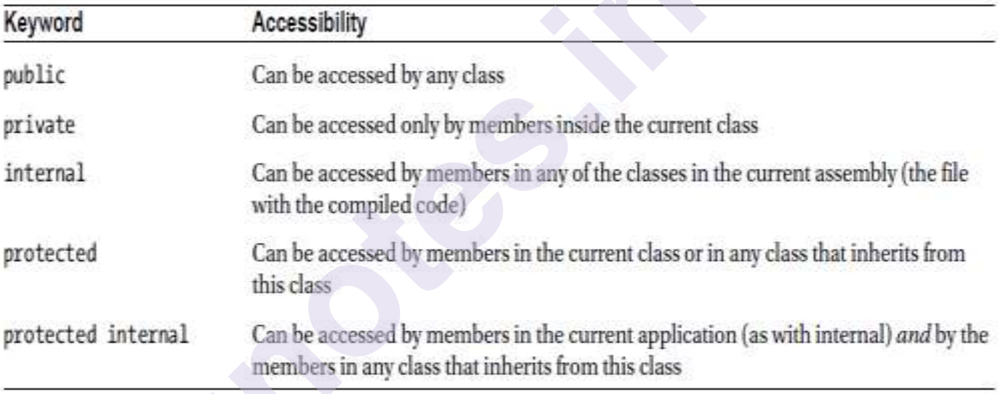
3.3.2 this Object in C#

- The “this” keyword in C# is used to refer to the current instance of the class. It is also used to differentiate between the method parameters and class fields if they both have the same name.
- Another usage of “this” keyword is to call another constructor from a constructor in the same class.
- Here, for an example, we are showing a record of Students i.e: id, Name, Age, and Subject. To refer to the fields of the current class, we have used the “this” keyword in C#.

```
public Student(int id, String name, int age, String subject) {
  this.id = id;
  this.name = name;
  this.subject = subject;
  this.age = age;
}
```

```
using System.IO;
using System;

class Student {
  public int id, age;
  public String name, subject;

  public Student(int id, String name, int age, String subject) {
    this.id = id;
    this.name = name;
    this.subject = subject;
    this.age = age;
  }
  public void showInfo() {
    Console.WriteLine(id + " " + name+" "+age+ " "+subject);
  }
}

class StudentDetails {
  public static void Main(string[] args) {
    Student std1 = new Student(001, "Jack", 23, "Maths");
    Student std2 = new Student(002, "Harry", 27, "Science");
    Student std3 = new Student(003, "Steve", 23,
"Programming");
    Student std4 = new Student(004, "David", 27, "English");


    std1.showInfo();
    std2.showInfo();
    std3.showInfo();
    std4.showInfo();
  }
}
```

3.3.3 Access Specifier



Keyword	Accessibility
public	Can be accessed by any class
private	Can be accessed only by members inside the current class
internal	Can be accessed by members in any of the classes in the current assembly (the file with the compiled code)
protected	Can be accessed by members in the current class or in any class that inherits from this class
protected internal	Can be accessed by members in the current application (as with internal) *and* by the members in any class that inherits from this class

3.3.4 Adding Properties in Class

- Property in C# is a member of a class that provides a flexible mechanism for classes to expose private fields. Internally, C# properties are special methods called accessors.
- A C# property have two accessors, get property accessor and set property accessor.
- A get accessor returns a property value, and a set accessor assigns a new value. The value keyword represents the value of a property.
- Properties in C# and .NET have various access levels that is defined by an access modifier.
- Properties can be read-write, read-only, or write-only. The read-write property implements both, a get and a set accessor.
- A write-only property implements a set accessor, but no get accessor. A read-only property implements a get accessor, but no set accessor.

```
class Person
{
  private string name; // field

  public string Name   // property
  {
    get { return name; }   // get method
    set { name = value; }  // set method
  }
}
```

Example:

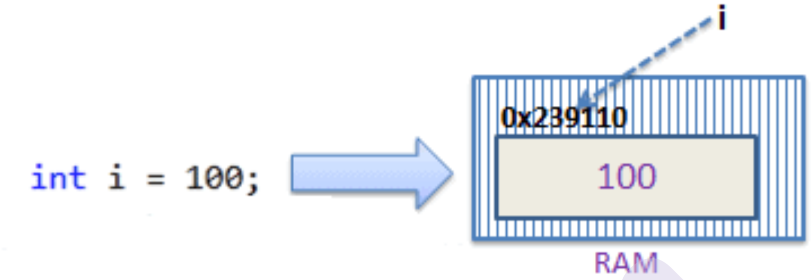
```
class Person
{
  private string name; // field
  public string Name   // property
  {
    get { return name; }
    set { name = value; }
  }
}

class Program
{
  static void Main(string[] args)
  {
    Person myObj = new Person();
    myObj.Name = "Liam";
    Console.WriteLine(myObj.Name);
  }
}
```

3.4 VALUE TYPES AND REFERENCE TYPES

- In C#, these data types are categorized based on how they store their value in the memory. C# includes the following categories of data types:
- Value type
- Reference type

Value Type

- A data type is a value type if it holds a data value within its own memory space. It means the variables of these data types directly contain values.
- For example, consider integer variable int i = 100;
- The system stores 100 in the memory space allocated for the variable i.
- The following image illustrates how 100 is stored at some hypothetical location in the memory (0x239110) for 'i':



The following data types are all of value type:

- bool
- byte
- char
- decimal
- double
- enum
- float
- int
- long etc.

Passing Value Type Variables

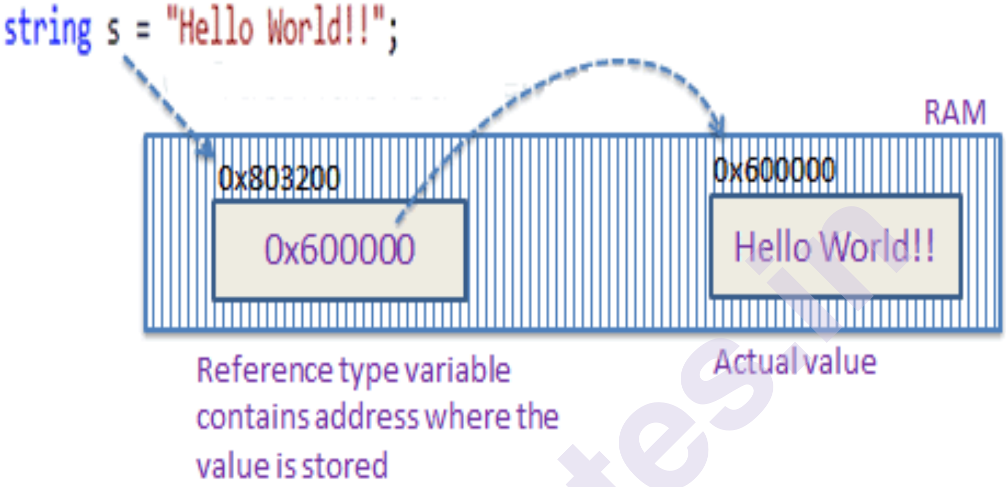
- When you pass a value-type variable from one method to another, the system creates a separate copy of a variable in another method.
- If value got changed in the one method, it wouldn't affect the variable in another method

```
static void ChangeValue(int x)
{
   x =  200;
   Console.WriteLine(x);
}
static void Main(string[] args)
{
   int i = 100;
   Console.WriteLine(i);
   ChangeValue(i);
   Console.WriteLine(i);
}
```

Reference Type

- Unlike value types, a reference type doesn't store its value directly. Instead, it stores the address where the value is being stored.
- In other words, a reference type contains a pointer to another memory location that holds the data.
- For example, consider the following string variable:

  string s = "Hello World!!";
- The following image shows how the system allocates the memory for the above string variable.


string s = "Hello World!!";
RAM
0x803200
0x600000
0x600000
Hello World!!
Reference type variable contains address where the value is stored
Actual value

- As you can see in the above image, the system selects a random location in memory (0x803200) for the variable s.
- The value of a variable s is 0x600000, which is the memory address of the actual data value.
- Thus, reference type stores the address of the location where the actual value is stored instead of the value itself.

The followings are reference type data types:

String

Arrays (even if their elements are value types)

Class

Delegate

Passing Reference Type Variables

- When you pass a reference type variable from one method to another, it doesn't create a new copy; instead, it passes the variable's address.
- So, If we change the value of a variable in a method, it will also be reflected in the calling method.

```
static void ChangeReferenceType(Student std2)
{
   std2.StudentName = "Steve";
}
static void Main(string[] args)
{
   Student std1 = new Student();
   std1.StudentName = "Bill";
   ChangeReferenceType(std1);
   Console.WriteLine(std1.StudentName);
}
```

3.5 UNDERSTANDING NAMESPACES AND ASSEMBLIES

Namespaces in C#

- Namespaces are used to organize the classes.
- It helps to control the scope of methods and classes in larger .Net programming projects.
- The biggest advantage of using namespace is that the class names which are declared in one namespace will not clash with the same class names declared in another namespace.
- It is also referred as named group of classes having common features.
- The members of a namespace can be namespaces, interfaces, structures, and delegates.
- There are two types of namespaces.
  1. User Defined Namespace
  2. Build in Namespaces.

Defining a User-Defined Namespace

- To define a namespace in C#, we will use the namespace keyword followed by the name of the namespace and curly braces containing the body of the namespace as follows:

Example:

```
// defining the namespace name1
namespace name1
{
   // C1 is the class in the namespace name1
   class C1
   {
      // class code
   }
}
```

```
using System;
namespace first_space {
  class namespace_cl {
    public void func() {
      Console.WriteLine("Inside first_space");
    }
  }
}
namespace second_space {
  class namespace_cl {
    public void func() {
      Console.WriteLine("Inside second_space");
    }
  }
}
class TestClass {
  static void Main(string[] args) {
    first_space.namespace_cl fc = new first_space.namespace_cl();
    second_space.namespace_cl sc = new
second_space.namespace_cl();
    fc.func();
    sc.func();
    Console.ReadKey();
  }
}
```

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

```
Inside first_space
Inside second_space
```

3.5.1 The Using Keyword

- The using keyword states that the program is using the names in the given namespace. For example, we are using the System namespace in our programs. The class Console is defined there. We just write -

```
Console.WriteLine ("Hello there");
```

We could have written the fully qualified name as -

```
System.Console.WriteLine("Hello there");
```

- You can also avoid prepending of namespaces with the using namespace directive.

- This directive tells the compiler that the subsequent code is making use of names in the specified namespace.
- The namespace is thus implied for the following code –

```
using System;
using first_space;
using second_space;

namespace first_space {
  class abc {
    public void func() {
      Console.WriteLine("Inside first_space");
    }
  }
}
namespace second_space {
  class efg {
    public void func() {
      Console.WriteLine("Inside second_space");
    }
  }
}
class TestClass {
  static void Main(string[] args) {
    abc fc = new abc();
    efg sc = new efg();
    fc.func();
    sc.func();
    Console.ReadKey();
  }
}
```

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

```
Inside first_space
Inside second_space
```

3.5.2 Nested Namespace

You can define one namespace inside another namespace as follows -

```
namespace namespace_name1 {
  // code declarations
  namespace namespace_name2 {
    // code declarations
  }
}
```

```
using System;
using first_space;
using first_space.second_space;
namespace first_space {
  class abc {
    public void func() {
      Console.WriteLine("Inside first_space");
    }
  }
  namespace second_space {
    class efg {
      public void func() {
        Console.WriteLine("Inside second_space");
      }
    }
  }
}
class TestClass {
  static void Main(string[] args) {
    abc fc = new abc();
    efg sc = new efg();
    fc.func();
    sc.func();
    Console.ReadKey();
  }
}
```

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

```
Inside first_space
Inside second_space
```

3.5.3 Assemblies in C#

- An assembly is a collection of types and resources that are built to work together and form a logical unit of functionality.
- Assemblies take the form of executable (.exe) or dynamic link library (. dll) files, and are the building blocks of .NET applications.
- An Assembly is a basic building block of .Net Framework applications. It is basically a compiled code that can be executed by the CLR.
- An assembly is a collection of types and resources that are built to work together and form a logical unit of functionality. An Assembly can be a DLL or exe depending upon the project that we choose.

Assemblies have the following properties:

- Assemblies are implemented as .exe or .dll files.
- For libraries that target the .NET Framework, you can share assemblies between applications by putting them in the global assembly cache (GAC). You must declare strong-name assemblies before you can include them in the GAC.
- Assemblies are only loaded into memory if they are required. If they aren't used, they aren't loaded. This means that assemblies can be an efficient way to manage resources in larger projects.

Assemblies are basically the following two types:

1. Private Assembly
2. Shared Assembly

1. Private Assembly

- It is an assembly that is being used by a single application only.
- Suppose we have a project in which we refer to a DLL so when we build that project that DLL will be copied to the bin folder of our project.
- That DLL becomes a private assembly within our project. Generally, the DLLs that are meant for a specific project are private assemblies.

2. Shared Assembly

- Assemblies that can be used in more than one project are known to be a shared assembly.
- Shared assemblies are generally installed in the GAC.
- Assemblies that are installed in the GAC are made available to all the .Net applications on that machine.

GAC(Global Assembly Cache)

- GAC stands for Global Assembly Cache. It is a memory that is used to store the assemblies that are meant to be used by various applications.
- Every computer that has CLR installed must have a GAC. GAC is a location that can be seen at “C:\Windows\assembly” for .Net

applications with frameworks up to 3.5. For higher frameworks like 4 and 4.5 the GAC can be seen at: “C:\Windows\Microsoft.NET\assembly\GAC_MSIL”.

Components of Assembly:

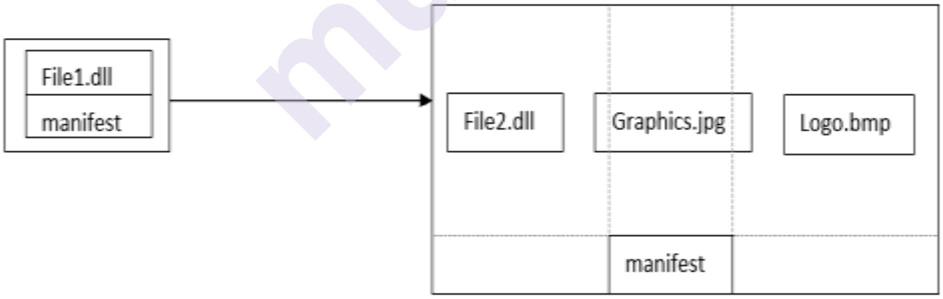
A static assembly consist of 4 elements:-

1. The assembly manifest, which contains assembly metadata.
2. Type metadata
3. Microsoft Intermediate Language(MSIL) code that implement the type.
4. A set of resources.

Assembly manifest :-

- Every assembly whether static or dynamic contain a collection of data that describes how the element in assembly relates to each other.
- It contains its assembly metadata such as assembly version requirement & security identity & all metadata needed to define this scope of the assembly & resolve references to resources & classes.
- It can be stored in either PE file(portable exe file) or .dll file with Microsoft intermediate language(MSIL) code or in a standalone PE file that contains only assembly manifest information.

The following illustration show the different ways to manifest can be stored.



The following table shows the information contain in the assembly manifest.

Information	**Description**
Assembly name	A text string specifying the assembly name.
Version number	A major & minor version number & a revision & build number. The common language runtime uses this no. to enforce version policy.

Culture	Information on the culture or language the assembly supports the information should be used only to designate an assembly as a satellite assembly containing culture or language specific information.
Strong name Information	The public key from the publisher if the assembly has been given a strong name.

3.6 ADVANCED CLASS PROGRAMMING

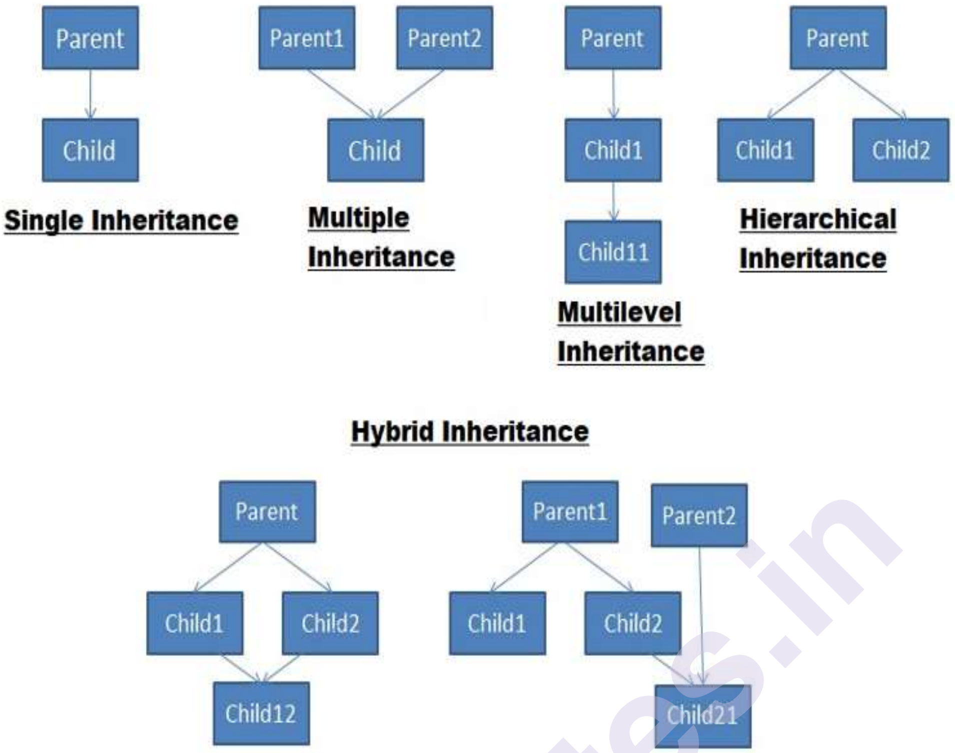
- Part of the art of object-oriented programming is determining object relations.
- For example, you could create a Product object that contains a ProductFamily object or a Car object that contains four Wheel objects.
- To create this sort of object relationship, all you need to do is define the appropriate variable or properties in the class. This type of relationship is called containment (or aggregation).

3.6.1 Inheritance in C#

- One of the most important concepts in object-oriented programming is inheritance.
- Inheritance allows us to define a class in terms of another class, which makes it easier to create and maintain an application.
- This also provides an opportunity to reuse the code functionality and speeds up implementation time.
- The process of creating new class from an existing class is called as inheritance.
- When creating a class, instead of writing completely new data members and member functions, the programmer can designate that the new class should inherit the members of an existing class.
- This existing class is called the base class, and the new class is referred to as the derived class.

```
<acess-specifier> class <base_class> {

  ...

}



class <derived_class> : <base_class> {

  ...

}
```

Types of Inheritance



OOPs support the six different types of inheritance as given below :

1. Single inheritance
2. Multi-level inheritance
3. Multiple inheritance
4. Hierarchical Inheritance
5. Hybrid Inheritance

Single inheritance

- In this inheritance, a derived class is created from a single base class.
- In the given example, Class A is the parent class and Class B is the child class since Class B inherits the features and behavior of the parent class A.

Multi-level inheritance

- **In this inheritance, a derived class is created from another derived class.**
- In the given example, class c inherits the properties and behavior of class B and class B inherits the properties and behavior of class B. So, here A is the parent class of B and

- class B is the parent class of C. So, here class C implicitly inherits the properties and behavior of class A along with Class B i.e there is a multilevel of inheritance.

Multiple inheritance

- In this inheritance, a derived class is created from more than one base class. This inheritance is not supported by .NET Languages like C#, F# etc. and Java Language.
- In the given example, class c inherits the properties and behavior of class B and class A at same level. So, here A and Class B both are the parent classes for Class C.

Hierarchical Inheritance

- In this inheritance, more than one derived classes are created from a single base class and further child classes act as parent classes for more than one child classes.
- In the given example, class A has two children class B and class D. Further, class B and class C both are having two children - class D and E; class F and G respectively.

Hybrid inheritance

- This is combination of more than one inheritance. Hence, it may be a combination of Multilevel and Multiple inheritance or Hierarchical etc.
- Example Of Multilevel Inheritance

```
using System;
class A
{
   public void displayA()
   {
      Console.WriteLine("Base Class Function");
   }
}
class B : A
{
   public void displayB()
   {
      Console.WriteLine("Derived Class Function");
   }
}
```

```
class C : B
{
   public void displayC()
   {
      Console.WriteLine("Derived Class Function");
   }
}
class Programs
{
   public static void Main(string[] args)
   {
      C obj = new C();
      obj.displayA();
      obj.displayB();
      obj.displayC();
      Console.ReadKey();
   }
}
```

Hierarchical Inheritance

```
using System;
class A
{
   public void displayA()
   {
      Console.WriteLine("Base Class Function");
   }
}
class B : A
{
   public void displayB()
   {
      Console.WriteLine("Derived Class Function");
   }
}
class C : A
```

```
{
   public void displayC()
   {
      Console.WriteLine("Derived Class Function");
   }
}
class Programs
{
   public static void Main(string[] args)
   {

      B obj1 = new B();
      C obj = new C();

      obj.displayA();
      obj.displayC();

      obj1.displayA();
      obj1.displayB();
      Console.ReadKey();
   }
}
```

3.6.2 Interfaces in C#

- Interface is like a contract. In the human world, the contract between the two or more humans binds them to act as per the contract.
- In the same way, the interface includes the declaration of one or more functionalities.
- Entities that implement the interface must define functionalities declared in the interface. In C#, a class or a struct can implement one or more interfaces
- In C# an interface can be defined using the interface keyword.
- Interfaces can contain methods, properties, indexers, and events as members.
- You cannot use any access modifier for any member of an interface. All the members by default are public members.

```
}
```

Implementing an Interface

A class or a Struct can implement one or more interfaces using colon (:).
Syntax: <Class or Struct Name> : <Interface Name>

```
using System;
interface MyInterface
{
   void display();
}
class CallInterface : MyInterface
{
   public void display()
   {
      Console.WriteLine("Interface is called");
   }
}
class CallInterface1 : MyInterface
{
   public void display()
   {
      Console.WriteLine("Interface Method");
   }
}
class Programs
{
   public static void Main(string[] args)
   {
      CallInterface obj = new CallInterface();
      CallInterface1 obj1 = new CallInterface1();

      obj.display();
      obj1.display();

      Console.ReadKey();
   }
}
```

```
using System;
interface Addition
{
   void add(int n1, int n2);
}
interface Subtraction
{
   void sub(int n1, int n2);
}
class CallInterface : Addition,Subtraction
{
   public void add(int n1, int n2)
   {
      Console.WriteLine("Addition :" + (n1 + n2));
   }
   public void sub(int n1, int n2)
   {
      Console.WriteLine("Subtraction :" + (n1 - n2));
   }
}
class Programs
{
   public static void Main(string[] args)
   {
      CallInterface obj = new CallInterface();

      obj.add(10,10);
      obj.sub(100,10);
      Console.ReadKey();
   }
}
```

Explicit Interface Implementation

- Explicit implementation is useful when class is implementing multiple interface thereby it is more readable and eliminates the confusion.
- It is also useful if interfaces have same method name coincidently.

```
using System;
interface I1
{
   void show();
}
interface I2
{
   void show();
}
class CallInterface : I1,I2
{
   void I1.show()
   {
      Console.WriteLine("I1 interface Method");
   }
   void I2.show()
   {
      Console.WriteLine("I2 Interface Method");
   }
}
class Programs
{
   public static void Main(string[] args)
   {
      CallInterface obj = new CallInterface();
      I1 i1 = (I1) obj;
      i1.show();
      I2 i2 = (I2)obj;
      i2.show();
      Console.ReadKey();
   }
}
```

3.6.3 Delegates in C#

- A function can have one or more parameters of different data types, but what if you want to pass a function itself as a parameter?
- How does C# handle the callback functions or event handler? The answer is - delegate.

- Delegate is like a pointer to a function.
- It is a reference type data type and it holds the reference of a method.
- All the delegates are implicitly derived from System.Delegate class.
- A delegate can be declared using delegate keyword followed by a function signature as shown below.

```
<accessmodifier>delegate<returntype><delegate_name>(<parameters>);
```

Steps to used Delegate

1) Declaration of delegate

<accessmodifier> delegate <returntype> <delegate_name>(<paramets);

public delegate void show();

2) **Delegate method declaration**

```
class DelegateMethod
{
  public void display()
   {
      Console.WriteLine("Delegate is called");
   }
}
```

3) **Delegate Instantiation(delegate object creation)**

DelegateMethod dm = new DelegateMethod();

MyDelegate d = new MyDelegate(dm.display);

4) **Calling Delegate**

d();

```
Example
using System;
public delegate void MyDelegate();   // Step1
class DelegateMethod
{
  public void display()
   {
      Console.WriteLine("Delegate is called");
   }
}
```

```
class Programs
{
    public static void Main(string[] args)
    {

        DelegateMethod dm = new DelegateMethod();
        MyDelegate d = new MyDelegate(dm.display);

        d();

        Console.ReadKey();
    }
}
```

Types of Delegate

1) Simple Delegate

- When delegate uses only one method to execute on the behalf of the delegate then it is called as simple delegate.

2) Multicast Delegate

- The delegate can points to multiple methods. A delegate that points multiple methods is called a multicast delegate.
- The "+" operator adds a function to the delegate object and the "-" operator removes an existing function from a delegate object.

```
using System;
public delegate void MathOptr(int n1, int n2);
class DelegateMethod
{
    public void add(int n1, int n2)
    {
        Console.WriteLine("Addition =" + (n1 + n2));
    }
    public void sub(int n1, int n2)
    {
        Console.WriteLine("Subtraction =" + (n1 - n2));
    }
    public void mult(int n1, int n2)
    {
        Console.WriteLine("Mult =" + (n1 * n2));
    }
    public void div(int n1, int n2)
    {
        Console.WriteLine("Div =" + (n1 / n2));
    }
}
class Programs
{
    public static void Main(string[] args)
    {

        DelegateMethod dm = new DelegateMethod();

        MathOptr d1 = new MathOptr(dm.add);
        MathOptr d2 = new MathOptr(dm.sub);
        MathOptr d3 = new MathOptr(dm.mult);
        MathOptr d4 = new MathOptr(dm.div);

        d1(10, 20);
        d2(20, 10);
        d3(10, 10);
        d4(20, 5);

        MathOptr d5 = d1 + d2 + d3 + d4;
        d5(10,2);

        Console.ReadKey();
    }
}
```

3.7 SUMMARY:

This chapter 3 gives the basic syntax of exception handling in C#. It discusses about exception, Assembly, Components of Assembly, Private and Shared Assembly, Garbage Collector, JIT compiler, Namespaces and few example programs. After learning the above topics, you can write many useful programs and built a strong foundation for larger programming projects.

3.8 QUESTIONS:

1. Write a short note on Assembly.
2. What is the significance of Assemblies in .NET?
3. List and Explain the Components of assemblies.
4. How Garbage collector works?
5. Explain JIT compiler.
6. What is namespace? Explain System namespace.

3.9 REFERENCE:

1) The Complete Reference: C#

2) Visual C# 2012: How to program.

3) https://docs.microsoft.com/en-us/dotnet/csharp/

❄❄❄❄❄❄

4

WEB FORM FUNDAMENTALS

4.0 OBJECTIVES

After going through this unit, you will be able to:

- To gain knowledge about what is web form and why it is used.
- How to design a webform and its applications
- How to write a code in C#.
- Explain about the controls and its uses.
- Various elements of Navigation.

4.1 INTRODUCTION

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

4.2 AN OVERVIEW

4.2.1 Writing Code

To start with writing a code i.e. dynamic and operational coding we need to switch to code-behind class. To switch **up and down** the code we use View code or View Designer buttons which appears just above the Solution Explorer window. Another approach is by double clicking either on .aspx page in the solution explorer or .aspx.cs page. Here we will be discussing C# code not the HTML code .

4.2.2 Using the Code-Behind class

Code-Behind class refers to the code that is written in separate class file that has extension .aspx.cs or .aspx.vb depending on the language used. When we switch to code view we will see the page class for our webpage.

For example when we create a webpage named SamplePage.aspx we will see a code behind class looks like this :

```
Using System;
Using.System.Collection.Generic;
Using.System.Linq;
Using.System.Web;
Using.System.Web.UI;
Using.System.Web.UI.WebControls;
Public partial class SamplePage: System.Web.UI.Page
{
   Protected void Page_Load(object sender , EventArgs e)
   {
    }
}
```

4.2.3 Adding Event Handlers

ASP.NET allows to implement event based model for our application.

To create an event handler in properties window we do the following

1) Double-click to create a new event handler for that event.

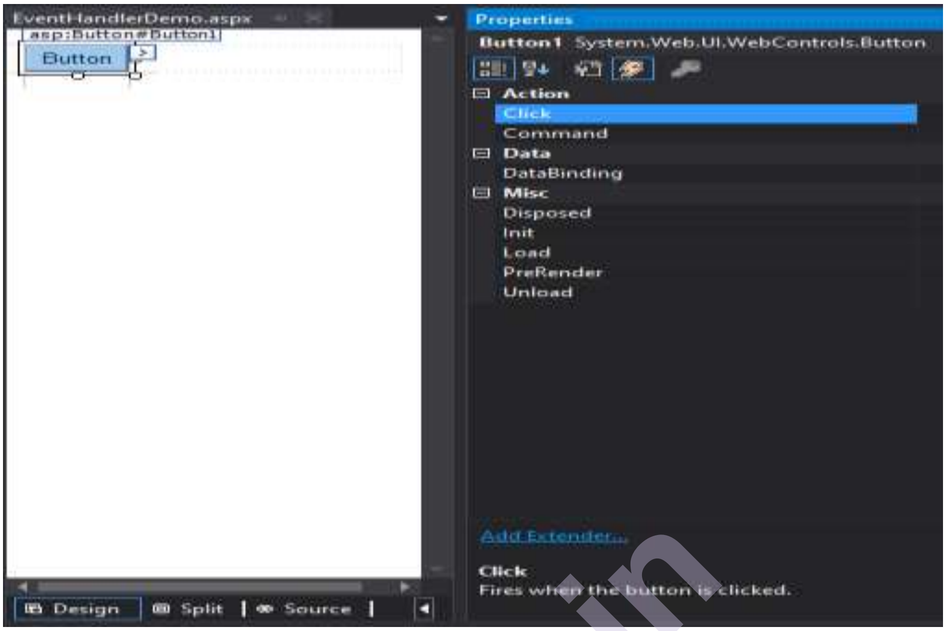
2) Type the name of the handler to create.

3) In the drop-down list select the name of existing handler.

For e.g:- The following diagram suggests that

Select the Button and right click and click on properties. In the properties window click the yellow icon event

Or simple double click the button it will show code behind file where we can write the code if the user click button what action has to be performed.



Click
Fires when the button is clicked.

```
public partialclass_EventHandlerDemo:System.Web.UI.Page
{
  protected void Page_Load(object sender, Event Args e)
  {
  }
  protected void Button_Click(object sender, EventArgs e)
  {
   }
}
```

Where object sender represents the object raising the event.

EventArgs represents the event arguments.

4.3 UNDERSTANDING THE ANATOMY OF AN ASP.NET APPLICATION

ASP.NET pages are divided into multiple webpages. Every webpage shares a common resource and configuration. Each webpage is executed in separate application domain.

If error occurs in application domain it does not affect other applications running on the same computer. Each web application is a combination of files, pages, handlers, modules and executable code that can be invoked from a virtual directory on the web server.

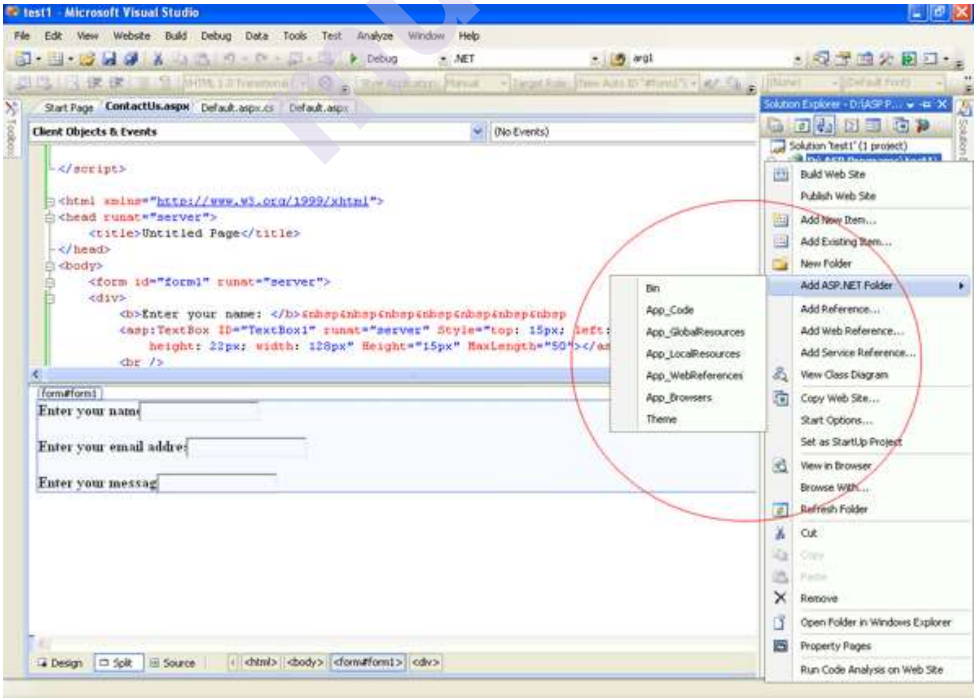
4.3.1 ASP.NET File Types

1) .aspx- This file is basically an extension of ASP.NET Web Pages. It is used for web form pages.
2) .ascx – This extension is basically used for web form user controls. The .ascx extension is often used for consistent parts of a website like headers, footers etc.
3) .asmx – This extension is used for files that implement ASP.NET Web Services.
4) .vb – This extension is used for VisualBasic.NET code modules. These files are code-behind files and are used to separate code from user interface.
5) .resx – These files are basically used for Windows Form Application for storing resources such as text strings for internationalization of applications.
6) Global.asax – These files are basically used to define Application-and-Session level variables and start up procedures.

4.3.2 ASP.NET Web Folders

ASP.NET defines several special folders. These special folders can be added to a Web site from Visual Studio menu system. We can add it by right clicking the Website Application Project and selecting Add ASP.NET folder.

The diagram shows the following :-



```
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Untitled Page</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <b>Enter your name: </b>&nbsp&nbsp&nbsp&nbsp&nbsp&nbsp&nbsp
        <asp:TextBox ID="TextBox1" runat="server" Style="top: 15px; left:
            height: 22px; width: 128px" Height="15px" MaxLength="50"></a
        <br />
```

Enter your name

Enter your email addre

Enter your messag

ASP.Net folders are as follows:-

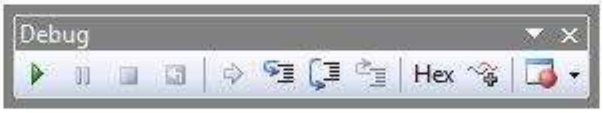
1) **App_Browsers -** ASP.net reserve this folder name for storing browser definition files. These files are used to determine the client browser capabilities and have .browser extension.
2) **App_Code –** App_code folder can contain source code for utility classes as well business objects (i.e .cs, .vb and .jsl files).The Classes that are present in App_Code folder are automatically compiled when your web application compiled.
3) **App_Data -** .App_Data folder is used by ASP.NET application for storing ASP.NET application local database. App_Data is used to store files that can be used as database files (.mdf and xml files). Developers can also use this folder for storing data for their ASP.NET Application.
4) **App_GlobalResources -** App_GlobalResources folder contains resources (.resx and .resources files) that are compiled into assemblies and have a global scope. These files are used to externalize text and images from our application code. This also helps us to support multiple languages and design-time changes without recompilation of your source code. These files are strongly typed and can be accessed programmatically.
5) **App_LocalResources -** App_LocalResources folder contains resources (.resx and .resources files). These files are available to a particular project, web page, master page or web user control.
6) App_Themes -These files contain subfolders that defines a specific theme or look and feel for your Web site. These consist of files (such as .skin, .css and image files) that defines the appearance of Web pages and controls.
7) **App_WebReferences –** It contains Web references files (.wsdl, .xsd, .disco, and .discomap files) that define references to Web services.
8) **Bin -** Bin folder contains compiled assemblies (.dll files) for code that we want to reference in our application. Assemblies in Bin folder are automatically reference in our application.

4.3.3 Debugging

Debugging enables programmers to see how the Code works step-by-step, how the variables values change, how objects are created and destroyed, and so on. Debugging is the application's method of inserting breakpoints. Such breakpoints are used to pause running a program from running.

We start the debugging session using F5(Debug/Start Debugging). This command starts our app with the debugger attached.

The Green arrow also starts the debugger



When running code in the debugger we often realize that we don’t need to see what happens to a particular function. We use some commands to skip through code

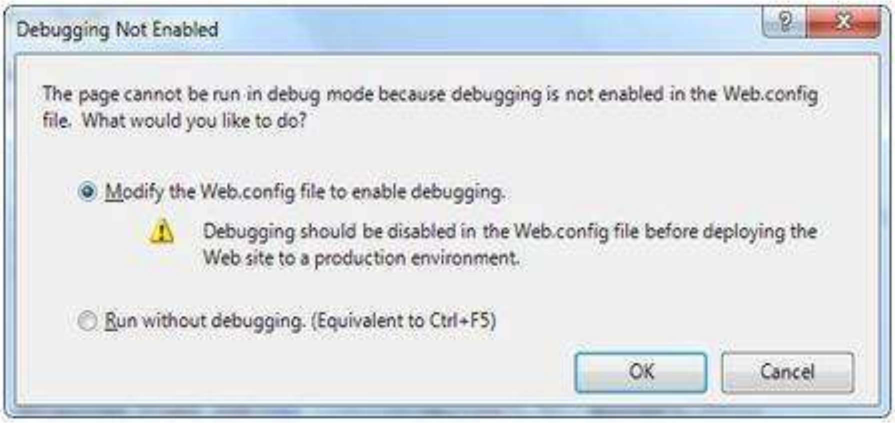
Keyboard Command	Menu Command	Description
F10	Step Over	If a current line contains a function call, Step Over runs the code then suspends execution at the first line of code after the called function returns.
Shift+F11	Step Out	Step Out continues running code and suspends execution when the current function returns.

Run to a specific location or function

These methods are useful when we know exactly what code we want to inspect and where we want to start debugging.

- **Set breakpoints in the code** :- To set a simple breakpoint in our code, open the source file in the Visual Studio editor. Set the cursor at the line of code where we want to suspend execution and then right-click in the code window to see the context menu and choose Breakpoint/Insert Breakpoint(or Press F9)
- **Run to the cursor location**:- To run to the cursor location, place the cursor on an executable line of code in a source window. On the editors context menu, choose Run to Cursor.
- **Manually break into the code** :- To break into the line of code in an executing app, choose Debug, Break All (Keyboard:Ctrl+Alt+Delete)
- **Run to a function on call stack** :- In the Call Stack Window, select the function, right-click and choose Run to Cursor.

When the site is executed for the first time, Visual Studio displays a prompt asking whether it should be enabled for debugging:



Debugging Not Enabled

The page cannot be run in debug mode because debugging is not enabled in the Web.config file. What would you like to do?

● Modify the Web.config file to enable debugging.

Debugging should be disabled in the Web.config file before deploying the Web site to a production environment.

○ Run without debugging. (Equivalent to Ctrl+F5)

4.4 INTRODUCTION TO SERVER CONTROLS

The server controls are the heart of ASP.NET pages which represents dynamic elements that our user can interact with. Server controls are tags that are understood by the server. There are three kinds of server controls

1) HTML Server Controls – Traditional HTML tags.
2) Web Server Controls – New ASP.NET tags.
3) Validation server Controls – For input validation.

Advantages of Server Controls

- ASP .NET Server Controls can detect the target browser's capabilities and render themselves accordingly.
- Newer set of controls that can be used in the same manner as any HTML control like Calender controls.
- ASP .NET Server Controls have an object model different from the traditional HTML and even provide a set of properties and methods that can change the outlook and behavior of the controls.
- ASP .NET Server Controls have higher level of abstraction.

4.4.1 HTML Server Controls :-

The HTML elements are considered as text in ASP.NET file. They cannot be referred as server side code. These controls can be treated as server control by adding **runat= "server"** attribute. The **id** attribute in the element can be added as reference to the control. All the HTML server controls are written in the <form> tag.

Some of the HTML Controls are as follows:-

- HtmlAnchor – Controls an <a> HTML element
- HtmlButton – Controls a <button> HTML element
- HtmlForm – Controls a <form> HTML element
- HtmlGeneric – Controls other HTML element not specified by a specific HTML server control.
- HtmlImage – Controls an <image> HTML element
- HtmlInputButtonControl - The HtmlInputButton control is used to control **<input type= "button">, <input type= "reset"> and <input type= "submit">** elements.
- HTMlInputCheckbox control - The HtmlInputCheckBox control is used to control as **<input type="checkbox">** element.
- HtmlInputFile control - The HtmlInputFile control is used to control an **<input type= "file">** element.
- HtmlInputHidden control - The HtmlInputHidden control is sued to control an **<input type= "hidden">** element.

- HtmlInputRadioButton -The HtmlInputRadioButton control is used to control an **<input type= “radio”>** element.
- HtmlSelect control - The HtmlSelect control is used to control **<select>** element.
- HtmlTextArea control – The HtmlTextArea control is used to control <textarea> element
- HtmlTable control - The HtmlTable control is used to control <table> element.

HTML Control Events

Events	Controls
ServerClick	HtmlAnchor, HtmlButton, HtmlInputButton, HtmlInputImage, HtmlInputReset
ServerChange	HtmlInputText, HtmlInputCheckBox, HtmlInputRadioButton, HtmlInputHidden, HtmlSelect, HtmlTextArea

For example:- This example has one button and one textbox. When user clicks the Button the result is shown in textbox

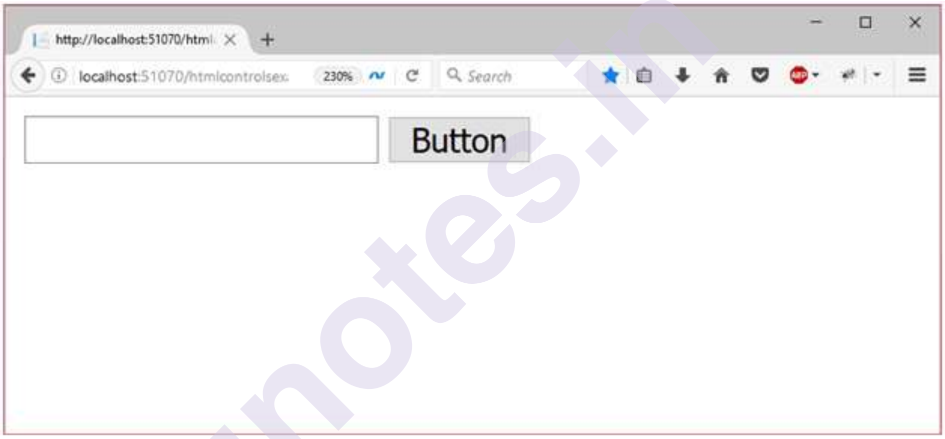
htmlcontrolsexample.aspx

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="html
controlsexample.aspx.cs"
Inherits="asp.netexample.htmlcontrolsexample" %>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title></title>
</head>
<body>
<form id="form1" runat="server">
<div>
<input id="Text1" type="text" runat="server"/>
<asp:Button ID="Button1" runat="server" Text="Button" OnClick="Butto
n1_Click"/>
</body>
</html>
```

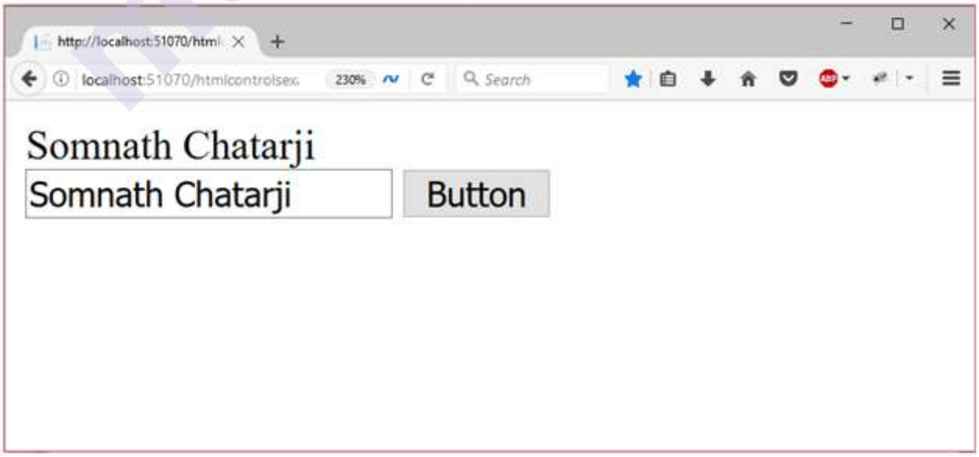
// htmlcontrolsexample.aspx.cs

```
using System;
namespace asp.netexample
{
  public partial class htmlcontrolsexample : System.Web.UI.Page
  {
      protected void Page_Load(object sender, EventArgs e)
       {
       }
 protected void Button1_Click(object sender, EventArgs e)
  {
      string a = Request.Form["Text1"];
       Response.Write(a);
  }
 }
}
```

OUTPUT :-



When we click the button after entering text, it responses back to client.



```
Somnath Chatarji
```

4.4.2 View State

View State is used to store user data on page at the time of post back of web page. It does not hold the controls, it holds the values of controls. It does not restore the value to control after page post back.

View State can hold the value on single web page, if we go to other page using response.redirect then View State will be null.

Syntax:-

Store the value in viewstate

```
ViewState[“name”]=”ASP.NET Programming”;
```

Retrieve information from viewstate

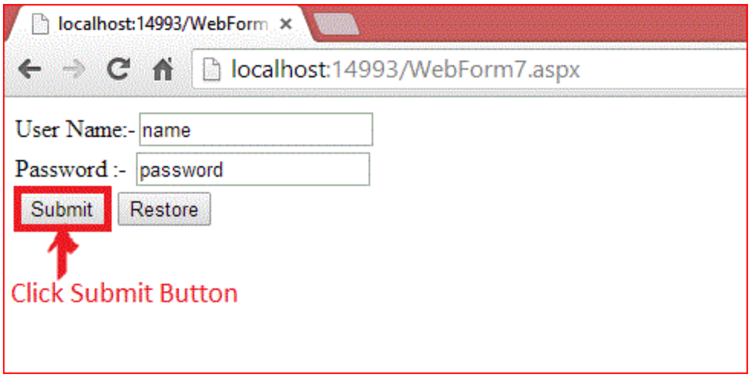
```
string value=ViewState[“name”].ToString();
```

Example of viewstate

//**aspprogramming.aspx**

```
protected void Button1_click(Object sender,EventArgs e)
{
//value of Textbox1 and TextBox2 is assigned on Viewstate
Viewstate[“name”]=TextBox1.Text;
Viewstate[“password”]=TextBox2.Text;
//after clicking on Button TextBox Value will be cleared
TextBox1.Text-TextBox2.Text=string.Empty;
}
protected void Button3_Click(object sender,EventArgs e)
{
//If Viewstate value is not null then value of viewstate is assign to TextB
If (Viewstate[“name”] != null)
   {
       TextBox1.Text=Viewstate[“name”].ToString();
    }
    If(Viewstate[“password’’] != null)
    {
      TextBox2.Text=Viewstate[“password”].ToString();
    }
}
```

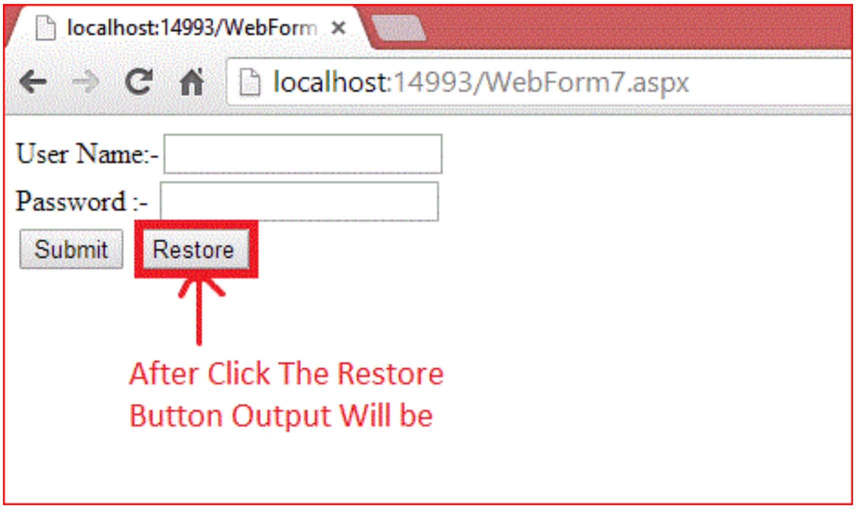
Output:-



User Name:- name

Password :- password

Click Submit Button



4.4.3 Using Page Class

Every web Page is a custom class that inherits from the system. web.UI.Page control. By inheriting form this class, your page class acquires a number of properties that our code can use. These include properties for enabling caching, validation and tracing.

Some fundamental properties of page class are as follows:-

Properties	Description
Application	Instance of the HttpApplicationState class; represents the state of the application. It is functionally equivalent to the ASP intrinsic Application object.
Cache	Instance of the *Cache* class; implements the cache for an ASP.NET application. More efficient and powerful than Application, it supports item priority and expiration.
Request	Instance of the HttpRequest class; represents the current HTTP request. It is functionally equivalent to the ASP intrinsic *Request* object.
Response	Instance of the HttpResponse class; sends HTTP response data to the client. It is functionally equivalent to the ASP intrinsic *Response* object.
Server	Instance of the HttpServerUtility class; provides helper methods for processing Web requests. It is functionally equivalent to the ASP intrinsic *Server* object.
Session	Instance of the HttpSessionState class; manages user-specific data. It is functionally equivalent to the ASP intrinsic *Session* object

Trace	Instance of the TraceContext class; performs tracing on the page.
User	the Principal object that represents the user making the request.
IsPostBack	Indicates whether the page is being loaded in response to a client postback or whether it is being loaded for the first time.

Sending the User to the New page

Sending the user to the new page means page navigation. It is the technique to navigate between webforms in asp.net

1. Hyperlink control – It is used to navigate to another page. Using hyperlink, you can navigate to another page within same application or to an external website. The hyperlink control is rendered as an HTML anchor<a> tag. The Hyperlink control does not expose server side events, so when the user clicks on a hyperlink there is no server side event to intercept the click.

   Example:- <a href=”adds.aspx”>Accounting</a>

2. Response.Redirect - It provides a method Redirect() which is used to redirects the user to another webpage which may or may not be on the same server. It can redirect the user to an external website on different server. This method updates the address bar and adds the updated URL to the browser history.

   Syntax:- Response.Redirect(path) Here the path is required which is the location of the ASP file to which control should be transferred.

3. Server.Transfer – The Server object provides method Transfer() which cause to quit current execution of web page and redirects the user to another web page on the same server. It reduces the server request and conserves server resources.

   Synatx :- Server.Transfer(path) Here the path is required which is the location of the ASP file to which control should be transferred.

4.4.4 Using Application Events

Application events are important in an ASP.NET application as the events are fired by server controls, we use them to perform additional processing tasks. For example, by using application events, we can write logging code that runs every time a request is received, no matter what page is being requested. Basic ASP.NET features such as session state and authentication use application events to plug into the ASP.NET processing pipeline.

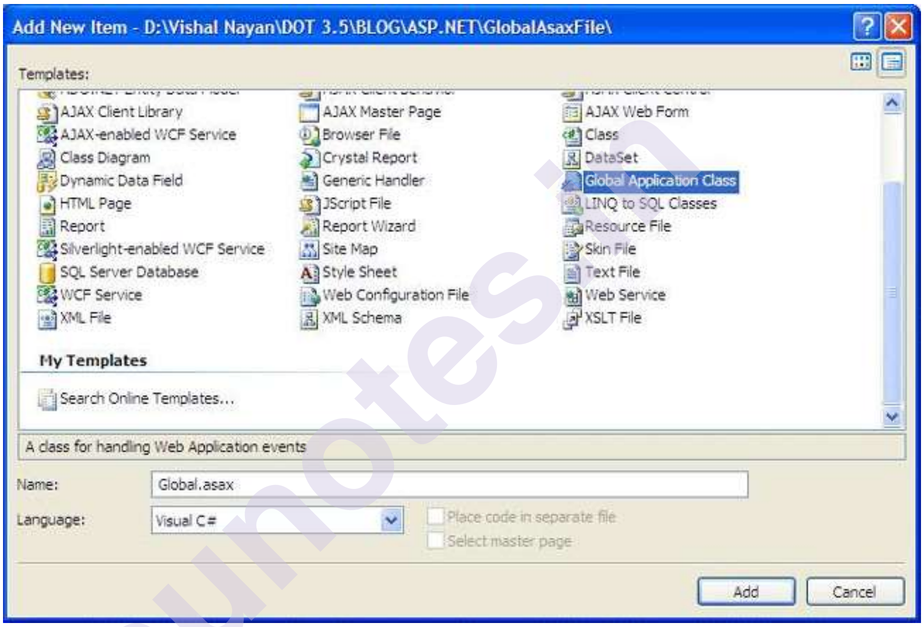
Global.asax allows us to write event handlers that react to global events in web applications. Global.asax files are never called directly by the user, rather they are called automatically in response to application events.

Some points to remember about global.asax:-

1. They do not contain any HTML or ASP.NET tags.
2. Contain methods with specific predefined names.
3. They defined methods for a single class, application class.
4. They are optional, but a web application has no more than one global.asax file.

How to add global.asax file:

Select Website >>Add New Item (or Project >> Add New Item if you're using the Visual Studio web project model) and choose the Global Application Class template.



Add New Item - D:\Vishal Nayan\DOT 3.5\BLOG\ASP.NET\GlobalAsaxFile\

A class for handling Web Application events

Name: Global.asax

Language: Visual C#

□ Place code in separate file
□ Select master page

After you have added the global.asax file, you will find that Visual Studio has added Application Event handlers:

The Event Handling Methods are as follows:-

Event Handling Method	Description
Application_Start()	This event is fired when the first instance of the HTTP Application class is created. It allows us to create objects that are accessible by all HTTP Application instances
Application_End()	This event is fired when the last instance of an HTTP Application class is destroyed. It's fired only once during an application lifetime.
Application_BeginRequest()	This event is fired when an application request is received. It's first event fired for a request which is often a page request

	(URL) that a user enters
Application_EndRequest()	This is the last Event fired for an application request
Session_Start()	This event is fired when a new user visits the application Website
Session_End()	This event is fired when a user's session times out, ends, or they leave the application website
Application_Error()	Application_Error() event occurs in response to an unhandled error

4.4.5 Configuring in ASP.NET Application

A configuration file (web.config) is used to manage various settings that define a website. The settings are stored in XML files that are separate from your application code. In this way you can configure settings independently from your code. Generally a website contains a single Web.config file stored inside the application root directory. However there can be many configuration files that manage settings at various levels within an application.

There are number of important settings that can be stored in the configuration file. Some of the most frequently used configurations, stored conveniently inside Web.config file are:

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

Configuration file looks like this

```
 <configuration>
      <connectionStrings>
           <add    name=”myCon”    connectionstring=”server=MyServer;
database=apeksha;uid=apekshashirke;pwd=mydata1223" />
      </connectionStrings>
</configuration/>
```

Different types of configuration files

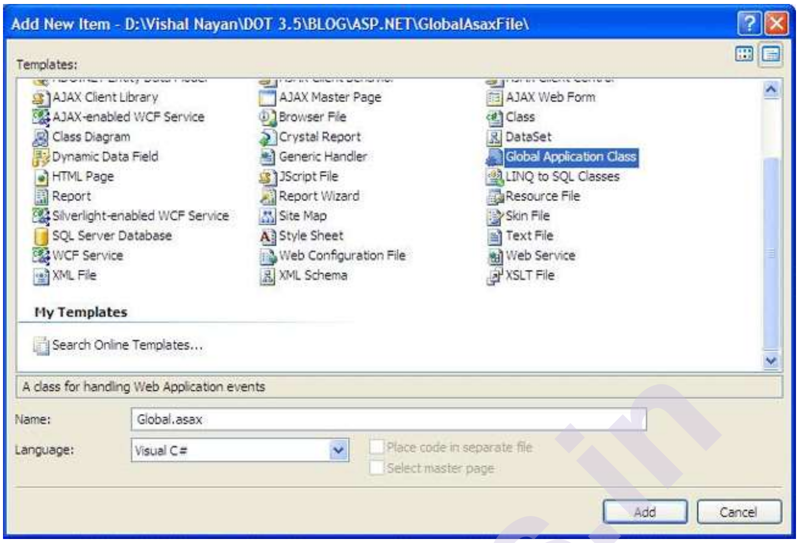
1) Machin.config :- Server or machine-wide configuration file.
2) Web.config :- Application configuration files which deal with a single application.

Example :- Program to display the number of visitors currently browsing your website

To Create a website and adding a webform to it

Adding a Global.asax to web applications

Add New Item>Global Application Class >Add



A class for handling Web Application events

Name: Global.asax

Language: Visual C#

□ Place code in separate file

□ Select master page

Global.asax

```
<%@ Application Language="C#" %>
<script runat=”server”>
 void Application_Start(object sender, EventArgs e)
  {
     Application[“OnlineUsers’]=0; //application variable
  }
  void Application_End(object sender, EventArgs e)
  {
    //Code that runs on application shutdown
  }
  Void Application_Error(object sender, EventArgs e)
  {
    //Code that runs when an unhandled error occurs
  }
  Void Session_Start(object sender, EventArgs e)
  {
     Application.Lock();
     Application[“OnlineUsers’]=(int) Application[“OnlineUsers’]-1;
     Application.UnLock();
  }
</script>
```

Web.config

```
<?xml version="1.0">
<configuration>
  <system.web>
    <sessionState mode="InProc" cookieless="false" timeout="1"/>
    <compilation debug="true" targetFramework="4.5.2"/>
    <httpruntime targetFramework="4,5.2"/>
  </system.web>
</configuration>
```

Default.aspx

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeFile="Default.aspx.cs" Inherits="_Default" %>
<html xmlns=http://www.w3.org/1999/xhtml>
<head runat="server">
<title></title>
</head>
<body>
 <form id="form1" runat="server">
    <div>
     Visitors Count :<%=Application["OnlineUsers"].ToString()%>
    </div>
  </form>
</body>
</html>
```

Where

- Mode :- The mode setting supports three options : inproc, sqlserver and stateserver.ASP.NET supports two modes : in process and out of process.
- Cookieless :- The cookieless option for ASP.NET is configured with this simple Boolean setting(true/false)
- Timeout :- This option controls the length of time a session is considered valid. The session timeout is a sliding value, on each request the timeout period is set to the current time plus the timeout value.

4.5 QUESTIONS:-

1. What is Code behind class in ASP.Net?
2. How to add Event Handlers in ASP.Net?
3. What are different types of ASP.Net files?
4. What are ASP.Net Folders?
5. Write a short note on Debugging.
6. Explain in detail the concept of HTMLServer Controls.
7. What do you mean by View state?
8. What are the Properties of Page class?
9. What are Event handling methods in ASP.Net?
10. Explain in detail Web Control hierarchy.

4.6 REFERENCES

https://developer.mozilla.org/en-US/docs/Learn/Forms/Your_first_form

https://www.oreilly.com/library/view/designing-web-navigation/9780596528102/ch01.html

https://flylib.com/books/en/2.370.1.28/1/ (Anatomy of ASP.NET application)

https://flylib.com/books/en/2.321.1.16/1/ (ASP.Net File Types)

https://www.c-sharpcorner.com/uploadfile/puranindia/special-folders-in-asp-net/ (ASP.Net Web folders)

https://www.tutorialspoint.com/asp.net/asp.net_debugging.htm (Debugging)

https://www.wideskills.com/aspnet/html-server-controls

https://www.c-sharpcorner.com/UploadFile/225740/what-is-view-state-and-how-it-works-in-Asp-Net53/

❄❄❄❄❄❄❄

5

FORM CONTROLS

Unit Structure

5.0 FORM CONTROLS

ASP.NET is Fully Based on Controls. the use of controls can easily make any Application without any problem in Asp.Net.There are some Types of controls which are used in Asp.Net.

1. Web Forms Standard controls.
2. Navigation Controls

3. Validation Controls
4. Web Parts controls
5. HTML Controls

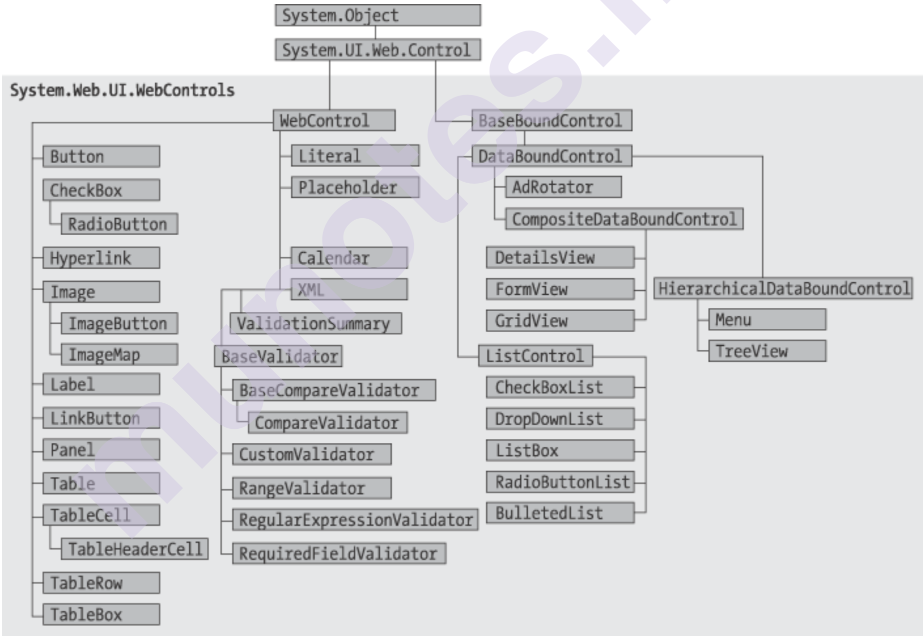
5.1 STEPPING UP TO WEB CONTROLS

Controls are building blocks in web form. Server controls are tags that are understood by the server. Web server controls are special ASP.NET tags understood by the server. Web server controls are also created on the server and they require runat=”server” attribute to work. Basic Web control include additional methods, events and properties

5.2 WEB CONTROL CLASSES

Web control classes are the basic control classes used in ASP.Net. Web control classes are defined in System.Web.UI.WebControls namespace.

The following diagram shows the web control hierarchy



Webserver controls can be divided into following categories

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

5) **Login and security controls** - These controls provide user authentication.

6) **Master pages** - These controls provide consistent layout and interface throughout the application.

7) **Navigation controls** - These controls help in navigation. For example, menus, tree view etc.

8) **Rich controls** - These controls implement special features. For example, AdRotator, FileUpload, and Calendar control.

The syntax for creating web server control is:

<asp:control_name id="some_id" runat="server" />

Example :-

<asp:Button id="btn1" Text="Click Here" runat="server"/>

Properties of web controls

1. AccessKey- Pressing this key with the Alt key moves focus to the control.
2. BackColor, ForeColor – Used to change the color of the background (BackColor) and text (ForeColor) of the control
3. BorderColor, BorderStyle, BorderWidth – Used to change the border of control in the browser. Each of these three ASP.NET properties maps directly to its CSS counterpart
4. CssClass – It is used to define the HTML class attribute for the control in the browser.
5. Enabled - Indicates whether the control is grayed out.
6. Font – Used to define different font related settings such as Font-Size, Font-Names and Font-Bold
7. Height, Width – It determines the height and width of the control in the browser
8. TabIndex - Gets or sets the tab index of the Web server control.
9. ToolTip - Gets or sets the text displayed when the mouse pointer hovers over the web server control.

10 Visible - It indicates whether a server control is visible.

11. UniqueID - Unique identifier.
12. SkinID - Gets or sets the skin to apply to the control.
13. Style - Gets a collection of text attributes that will be rendered as a style attribute on the outer tag of the Web server control.

5.2.1 Units

Many properties of the controls use measurements such as Border width, Height and Width. The values here requires the unit structure which combines a numeric value with a type of measurement(pixels, percentage and so on)

Example :- <asp:Label ID="Label1" runat="server" Height="10px" Text="Label"> </asp:Label>

5.2.2 Enumerations

Enumerations are used in heavily in the .NET class library to group a set of related constants, which may use for collection type values for the controls such as font names, colours or styles. For example, When you set a controls BorderStyle property, we can choose one of the several predefined values from the Borderstyle enumeration

Example :-

Button1.BorderStyle=BorderStyle.Dashed; //C#

<asp:Label BorderStyle=”Dashed” Text=”Border Test” ID=”lbl” runat=”server” />

5.2.3 Colors

The color property refers to a color object from System.Drawing namespace. we can create color objects in several ways:-

- **ARGB(alpha,red,gren,blue) color value** : we specify each value as an integer from 0 to 255. The alpha component represents the transparency of a color.
- **predefined .NET color name :** we choose the corresponding name read-only property from color structure. These properties include the 140 HTML color names.
- **HTML color name :** we specify this value as a string by using the ColorTranslator class.

Example :-

Label1.ForeColor=System.Drawing,Color.Red

Label1.ForeColor=System.Drawing.ColorTranslator.FromHtml(“#FF0000”)

Label1.ForeColor=Color.Red

Label1.ForeColor= ColorTranslator.FromHtml(“#FF0000”)

5.2.4 Fonts

The Font Property actually references a full Font Info Object which defines in the System. Web. UI. Web Controls namespace

Property	Description
Name	A String indicating the font name (such as Verdana)
Names	An array of strings with font names in the order of preference. The browser will use the first matching font that’s installed on the user’s computer.
Size	The size of the font as a FontUnit object. This can represent an absolute or relative size
Bold, Italic, Overline, Strikeout, Underline	Boolean properties that apply the given style attribute.

5.2.5 Default Button

Default Button property ensures that whenever user presses Enter key the button that is set as default will be triggered. we can set the default button at the form level meaning of all the buttons in the form the one that is set as default will be triggered whenever user presses Enter key.

Example :-

```
<form id="form1" runat="server" defaultbutton="Button1">
<asp:Button ID="Button1" runat="server" Text="Button"
OnClick = "Button1_Click" />
<asp:Button ID="Button2" runat="server" Text="Button"
   OnClick = "Button2_Click" />
<asp:Button ID="Button3" runat="server" Text="Button"
   OnClick = "Button3_Click" />
<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
</form>
```

5.3 LIST CONTROLS

These controls are used to display the list with some list items in the page. These controls include List Box, Drop Down List, Check Box List, Radio Button List and Bulleted List. To add items to the list we have to define <asp:ListItem> elements between the opening and closing tags of the control.

1. ListBox

We can select multiple items from ListBox at a time.

Basic syntax of list box control:

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

2. DropDownList

DropDownList control is used select single option from multiple listed items.

Basic syntax of DropDown List :

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

Common properties of List Box and Drop Down List are as follows

Properties	Description
SelectedValue	Get the value of the selected item from the dropdown list.
SelectedIndex	Gets the index of the selected item from the dropdown box.
SelectedItem	Gets the text of selected item from the list.
Items	Gets the collection of items from the dropdown list.
DataTextField	Name of the data source field to supply the text of the items. Generally this field came from the data source.
DataValueField	Name of the data source field to supply the value of the items. This is not visible field to list controls, but you can use it in the code.
DataSourceID	ID of the data source control to provide data.

Common properties of each list item objects:

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

3. CheckBoxList

The CheckBox control allows the user to set true/false or yes/no type options. The user can select or deselect it. When a check box is selected it has the value True, and when it is cleared, it holds the value False.

Properties of CheckBox Control

Properties	Description
Appearance	Gets or sets a value determining the appearance of the check box.
AutoCheck	Gets or sets a value indicating whether the Checked or CheckedState value and the appearance of the control automatically change when the check box is selected.
CheckAlign	Gets or sets the horizontal and vertical alignment of the check mark on the check box.
Checked	Gets or sets a value indicating whether the check box

	is selected.
CheckState	Gets or sets the state of a check box.
Text	Gets or sets the caption of a check box.
ThreeState	Gets or sets a value indicating whether or not a check box should allow three check states rather than two.

4. RadioButtonList

Radio Button List Control is same as Drop Down List but it displays a list of radio buttons that can be arranged either horizontally or vertically. We can select only one item from the given Radio Button List of options.

Properties of Radio Button List

Properties	Description
RepeatColumns	It displays the number of columns of radio buttons.
RepeatDirection	It pacifies the direction in which the controls to be repeated. The values available are Horizontal and Vertical. Default value is vertical.
RepeatLayout	Determines whether the radio buttons display in an HTML table.

5. Bulleted List

The bulleted list control creates bulleted lists or numbered lists.

Properties	Description
Bulletstyle	Determines the type of list. We can choose from Numbered(1,2,3 …);Lower Alpha (a, b, c,…); Upper Alpha(A,B,C,…);Lower Roman(i, ii, iii….)and Upper Roman(I.II,III,…) and the bullet symbols Disc, Circle, Square or Custom Image
BulletImageUrl	If the Bullet Style is set to Custom Image, this points to the image that is placed to the left to each item as a bullet.
FirstBulletNumber	In a ordered list (using (using Numbered, Lower Alpha, Upper Alpha, Lower Roman and Upper Roman) this sets the first value. For example if you set First Bullet Number to 3, the list might read 3,4,5…
DisplayMode	It determines whether the text of each item is rendered as a text or hyperlink

5.4 TABLE CONTROLS

In .NET Framework the Table class enables us to build an HTML Table. The System.Web.UI Controls namespace defines the Table class along with other web controls

Control	Code	Description
Table	<asp:table>	Parent control for Table Row controls. The Rows property of the Table object is a collection of Table Row objects.
TableRow	<asp:TableRow>	Parent control for Table Cell controls. The Cells property of the Table Row object contains a collection of Table Cell objects.
TableCell	<asp:TableCell>	Contains content to be displayed. The Text property contains HTML text. The Controls collection can contain other controls.
TableRow Collection	<asp:TableRowCollection>	It encapsulates Table Row Collection and is used to manage a collection of table or removing a row from it.
TableCell Collection	<asp:TableCellCollection>	It manages collection of table cells such as adding a cell to a row or removing a cell from it.
TableHeader Collection	<asp:TableHeadercell>	It encapsulates a table header cell.

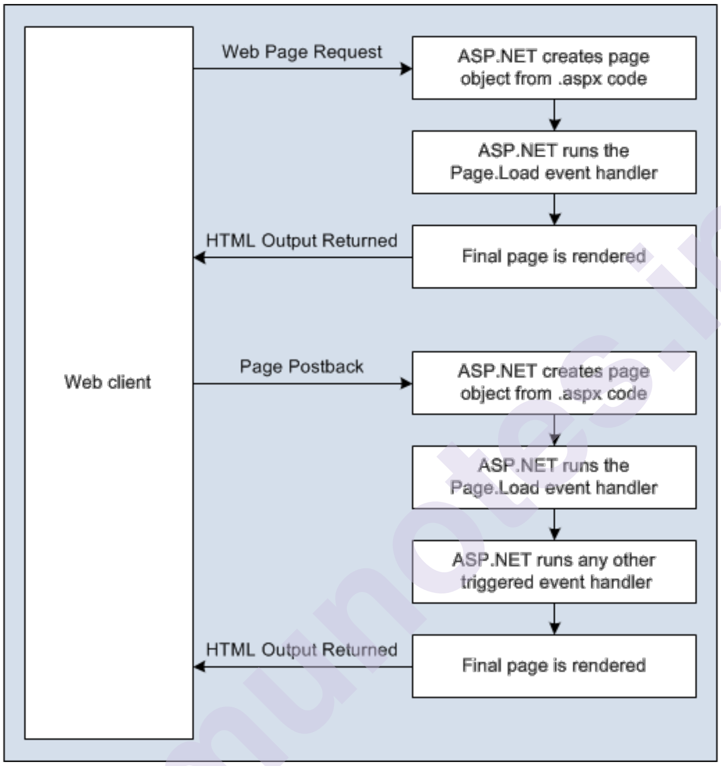
Properties of Table class

Property	Description
BackImageUrl	An URL to an image is used as background for the table
Caption	The caption of the table.
CaptionAlign	The alignment of the caption text.
CellPadding	The space between the cell walls and content.
CellSpacing	The space between cells.
GridLines	The gridline format in the Table.

HorizontalAlign	The horizontal alignment of the table in the page
Rows	A collection of rows in the table
Runat	Specifies that the control is a server control. Must be set to "server"

5.5 WEB CONTROL EVENTS AND AUTO POST BACK

The following diagram shows the order of event in ASP.NET page processing:


Web client
Web Page Request
ASP.NET creates page object from .aspx code
ASP.NET runs the Page.Load event handler
HTML Output Returned
Final page is rendered
Page Postback
ASP.NET creates page object from .aspx code
ASP.NET runs the Page.Load event handler
ASP.NET runs any other triggered event handler
HTML Output Returned
Final page is rendered

Sometimes we need to write the server code that will react immediately to an event that occurs on the client. Some events, such as Click event of a button take place immediately, because when clicked, the button posts back the page. However, other actions do cause events but don't trigger a post back.

For example when user chooses a new item in a list (which triggers SelectedIndexChanged event) or changes the text in a text box (the TextChangedevent). In these cases without post back your code has no way to run.

ASP.NET handles this by giving you two options:

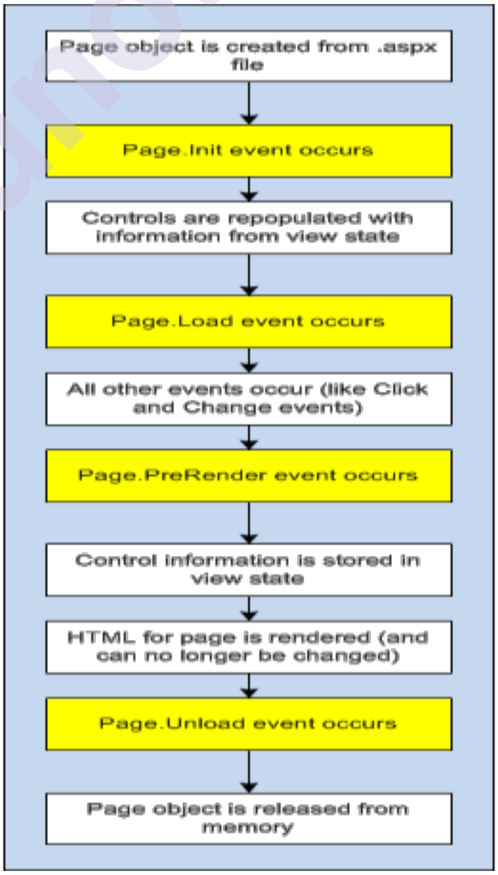
– To wait until the next post back to react to the event. For example imagine you want to react to the SelectedIndexChanged event in a list. If the user selects an item in a list, nothing happens immediately. But, if the user clicks a button to post back the page, two events fire: ButtonClick followed by ListBox.SelectedIndexChanged. And if you have several controls, it’s quite possible for a single post back to result in several change events, which fire one after the other, in an undetermined order.

– To use the automatic post back feature to force a control to post back the page immediately when it detects a specific user action. In this scenario, when the user clicks a new item in the list, the page is posted back, your code executes, and a new version of the page is returned.

Event	Web controls	AlwaysPostBack
Click	Button, ImageButton	True
TextChanged	TextBox	False
CheckedChanged	CheckBox, RadioButton	False
SelectedIndexChanged	DropDownList, ListBox, CheckBoxList, RadioButtonList	False

If you want to capture a change event (such as TextChanged, CheckedChanged, or SelectedIndexChanged) immediately, by setting the control’s AutoPostBack property to true. This way the page will be submitted automatically when the user interacts with the control.

When the page is posted back, ASP.NET examines the page, loads all the current information, and then allows your code to perform some extra processing before returning the page back to the user. The next picture illustrates this:


Page object is created from .aspx file
Page.Init event occurs
Controls are repopulated with information from view state
Page.Load event occurs
All other events occur (like Click and Change events)
Page.PreRender event occurs
Control information is stored in view state
HTML for page is rendered (and can no longer be changed)
Page.Unload event occurs
Page object is released from memory

AutoPostBack in ASP.NET

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

What is AutoPostBack Property in ASP.NET?

The web Pages with one or more web controls are configured to use AutoPostBack, the ASP.NET adds a special JavaScript function to the rendered HTML page. This function is named _doPostBack(). When called, it triggers a PostBack sending data back to the web server.

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

```
<script language= “text/javascript”>
 Function _doPostBack (event Target, event Argument)
  {
    If (!theForm.onsubmit || (theForm.onsubmit()!=false))
    the Form._EVENTTARGET.value=eventTarget;
    theForm._EVENTTARGET.value=eventTarget;
    theForm.submit();
  }
</script>
```

ASP.NET generates the _doPostBack() function automatically provided atleast one control on three page uses automatic postbacks.

Life Cycle of a Web Page

To work with ASP.NET web Controls events, we need to understand the webpage lifecycle. The following actions will be taken place when a user changes the control that has Auto Post Back property set to true:

1 On the client side the Javascript_doPostBack function is invoked and the page is submitted to the server.

2. ASP.NET recreates the Page object using the .aspx file.

3. ASP.NET retrieves state information from the hidden view state field and updates the control accordingly.

4. The Page Load event is fired.

5. The appropriate change event is fired for the control.

6. The Page.PreRender event fired and the page is rendered.

7. Finally, the Page.Unload event is fired.

8. The new page is sent to the client.

5.5 VALIDATION CONTROLS

Validation controls are used to implement presentation logic, to validate user input data and it is also used for data format, data type and data range. Validation consist of two types client side and server side.

Client Side – Dependent on browser and scripting language support.

Sever Side – It is not dependent on browser and scripting language support.

Validation controls in ASP.NET

There are six types of validation controls in ASP.NET

1. Required Field Validation Control
2. Compare Validator Control
3. Range Validator Control
4. Regular Expression Validator Control
5. Custom Validator Control
6. Validation Summary

1. RequiredFieldValidator Control

The RequiredFieldValidator control is simple validation control, which checks to see if the data is entered for the input control. We can have a RequiredFieldValidator control for each form element on which you wish to enforce Mandatory Field rule.

Syntax:-

```
<asp:RequiredFieldValidator ID="RequiredFielsdValidator3" runat="server" Style="top: 98px;left: 367px; position: absolute; height: 26px;width: 162px"
ErrorMessage="password required" ControlToValidate="TextBox2" >
</asp: RequiredFieldValidator>
```

Example:-

```
Your Name  : <br/>
<asp:TextBox runat="server" id="txtName"/>
<asp:RequiredFieldValidator runat="server" id="reqName" ForeColor="Red" controltovalidate"txtName" errormessage="Please enter your name !"/>
```

2. CompareValidator Control

The CompareValidator control allows us to make comparison to compare data entered in an input control with a constant value or a value in a different control.

Syntax:-

```
<asp:RequiredFieldValidator ID=”RequiredFieldValidator3”
runat=”server” Style=”top: 145px; left: 367px; position: absolute; height:
26px; width: 162px” ErrorMessage=”password required”
ControlToValidate=”TextBox3” ></asp: RequiredFieldValidator>
```

Example :-

```
Password:<br />
<asp:TextBox runat=”server” id=”txt11” TextMode=”Password”/>
<br/>ReEnter password :<br/>
<asp:TextBox runat=”server” id=”txt11” TextMode=”Password”/><br/>
<asp:CompareValidator runat=”server” id=”cmpNumbers”
ForeColor=”Red” Controltovalidate=”txt12” controltocompare=”txt11”
operator=”LessThan” type=”Integer” errormessage=”Password should
match !”> Password should match!</asp:CompareValidator>
```

3. RangeValidator Control

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

Syntax:-

```
<asp:RangeValidatorID=”rvclass” runat=”server”
ControlToValidate=”txtclass” Errormessage=”Enter your class (6-12)”
Maximum Value=”12” Minimum Value=”6” Type=”Integer”/>
```

Example:-

```
Enter age :<br/>
<asp:TextBox runat=”server” id=”txt1”/>
<asp:RangeValidator ID=”RangeValidator2” Type=”Integer”
runat=”server” ForeColor=”Red” ControlToValidate=”txt1”
MinimumValue=”18” MaximumValue=”100” Error Message=”Not valid
age”>Not valid age</asp:RangeValidator>
```

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

4. Regular Expression Validator Control

The RequiredExpressionValidator allows validating the input text by matching against a pattern of regular expression. The regular expression is set in the ValidationExpression property. The following table summarizes the commonly used syntax construct for regular expressions.

Character Escapes	**Description**
\b	Matches a backspace.
\t	Matches a tab.
\r	Matches a carriage return.
\v	Matches a vertical tab
\f	Matches a form feed
\n	Matches a new line
\	Escape character

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

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

Syntax:-

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

</asp:RegularExpressionValidator>
```

Example :-

Email ID :<br/>

<asp:TextBox runat=”server” id=”txtnumber”/>

```
<asp:RegularExpressionvalidator runat=”server” ForeColor=”Red” id=”rexNumber” controlToValidate =”txtnumber” errormessage=”Please enter valid email address!”

ValidationExpression=”\w=([-+.’]\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*”>Please enter valid email address!</asp:regularExpressionValidator>
```

5. CustomValidator

The CustomValidator control allows writing application specific custom validation routines for both the client side and the server side validation. The client side validation routine should be written in a scripting language, such as JavaScript or VBScript, which the browser can understand. The server side validation routine must be called from the control's ServerValidate event handler. The server side validation routine should be written in any .Net language, like C# or VB.Net.

Syntax:-

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

Example:-

```
Custom text :<br>
<asp:TextBox runat=”server” id=”txtCustom”/>
<asp:CustomValidator id=”CustomValidator1” ControlToValidate=”txtCustom” ClientValidationFunction=”ServerValidation” Display=”Static” ForeColor=”Red” runat=”server”/>
```

5.7 RICH CONTROLS

Rich controls are built with multiple HTML elements and contain rich functionality. Examples of rich controls are the calendar control and AdRotator.

5.7.1 The Calendar Control

The calendar control is used to display a calendar in the browser. This control displays a one month calendar that allows the user to select dates and move to the next and previous months. It is a functionality rich web control which provides displaying one month at a time, selecting a day, a week or a month, selecting a range of days moving from month to month and controlling the display of the days programmatically.

Syntax:-

```
<asp:Calendar ID=”Calendar1” runat=”server”>
</asp:Calendar>
```

Properties and events of calendar control

Properties	Description
Caption	Gets or sets the caption for the calendar control.
CaptionAlign	Gets or sets the alignment for the caption.
CellPadding	Gets or sets the number of spaces between the data and the cell border
CellSpacing	Gets or sets the space between cells.
DayHeaderStyle	Gets the style properties for the section that displays the day of the week.
DayNameFormat	Gets or sets format of days of the week.
DayStyle	Gets the style properties for the days in the displayed month.
FirstDayOfWeek	Gets or sets the day of week to display in the first column.
NextMonthText	Gets or sets the text for next month navigation control. The default value is >.
NextPrevFormat	Gets or sets the format of the next and previous month navigation control.
OtherMonthDayStyle	Gets the style properties for the days on the Calendar control that are not in the displayed month.
PrevMonthText	Gets or sets the text for previous month navigation control. The default value is <.
SelectedDate	Gets or sets the selected date.
SelectedDates	Gets a collection of Date Time objects representing the selected dates.
SelectedDayStyle	Gets the style properties for the selected dates.
SelectionMode	Gets or sets the selection mode that specifies whether the user can select a single day, a week or an entire month.
SelectMonthText	Gets or sets the text for the month selection element in the selector column.
SelectorStyle	Gets the style properties for the week and month selector column.
SelectWeekText	Gets or sets the text displayed for the week selection element in the selector column.
ShowDayHeader	Gets or sets the value indicating whether the heading for the days of the week is displayed.
ShowGridLines	Gets or sets the value indicating whether the gridlines would be shown.
ShowNextPrevMonth	Gets or sets a value indicating whether next and previous month navigation elements are shown in

	the title section.
ShowTitle	Gets or sets a value indicating whether the title section is displayed.
TitleFormat	Gets or sets the format for the title section.
Titlestyle	Get the style properties of the title heading for the Calendar control.
TodayDayStyle	Gets or sets the value for today's date.
UseAccessibleHeader	Gets or sets a value that indicates whether to render the table header <th> HTML element for the day headers instead of the table data <td> HTML element.
VisibleDate	Gets or sets the date that specifies the month to display.
WeekendDayStyle	Gets the style properties for the weekend dates on the Calendar control.

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

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

5.7.2 The AdRotator

The AdRotator is to provide a graphic on a page that is chosen randomly from a group of possible images. Every time the page is requested an image is selected at random and displayed which is the "rotation" indicated by the name Adrotator. One use of Adrotator is to show banner-style advertisements on a page but we can use it any time we want to vary an image randomly.

The Advertisment file

The AdRotator stores its list of image files in an XML file. This file uses the format shown here:

```
<Advertisements>
<Ad>
<ImageUrl>proetech.jpg<ImageUrl>
<NavigateUrl>http://www.prosetech.com<NavigateUrl>
<AlternateText><ProseTech Site</Alternate text>
<Impressions>1</Impressions>
```

```
<Keyword>Computer</Keyword>
</Ad>
</Advertisements>
```

Advertisement File Elements

1. ImageUrl - The path of image that will be displayed.
2. NavigateUrl - The link that will be followed when the user clicks the ad.
3. AlternateText - The text that will be displayed instead of the picture if it cannot be displayed.
4. Keyword - Keyword identifying a group of advertisements. This is used for filtering.
5. Impressions - The number indicating how often an advertisement will appear.

The AdRotator class

The actual AdRotator class provides a limited set of properties

1. _blank :- The link opens a new unframed window.
2. _parent :- The link opens in the parent of the current frame.
3. _self :- The link opens in the current frame
4. _top :- The link opens in the topmost frame of the current window.

5.7.3 Pages with Multiple Views

MultiView and View controls allow you to divide the content of a page into different groups, displaying only one group at a time. Each View control manages one group of content and all the View controls are held together in a MultiView control.

The MultiView control is responsible for displaying one View control at a time. The View displayed is called the active view.

Syntax of Multiview

```
<asp:Multview ID=”MultiView1” runat=”server”>
</asp:MultiView>
```

Syntax of View

```
<asp:View ID=“View1” runat=”server”>
</asp:View>
```

Properties of View and MultiView Controls

Both View and MultiView controls are derived from Control class and inherit all its properties, methods, and events. The most important property of the View control is Visible property of type Boolean, which sets the visibility of a view.

The MultiView control has the following important properties:

Properties	Description
Views	Collection of View controls within the Multi View.
ActiveViewIndex	A zero based index that denotes the active view. If no view is active, then the index is -1.

The Command Name attribute of the button control associated with the navigation of the MultiView control are associated with some related field of the MultiView control. For example, if a button control with CommandName value as NextView is associated with the navigation of the multi view, it automatically navigates to the next view when the button is clicked.

The following table shows the default command names of the above properties:

Properties	**Description**
NextViewCommandName	Next View
PreviousViewCommandName	Prev View
SwitchViewByIDCommandName	Switch View By ID
SwitchViewByIndexCommandName	Switch View By Index

The important methods of the multi view control are:

Methods	**Description**
SetActiveView	Sets the active view
GetActiveView	Retrieves the active view

Every time a view is changed, the page is posted back to the server and a number of events are raised. Some important events are:

Events	**Description**
ActiveViewChanged	Raised when a view is changed
Activate	Raised by the active view
Deactivate	Raised by the inactive view

5.8 USER CONTROLS AND GRAPHICS

User controls are used to have code which is used multiple times in an application. The user control can then be reused across the application. The user control needs to be registered on the ASP.Net page before it can be used. To use user control across all pages in an application, register it into the web.

5.8.1 User Controls

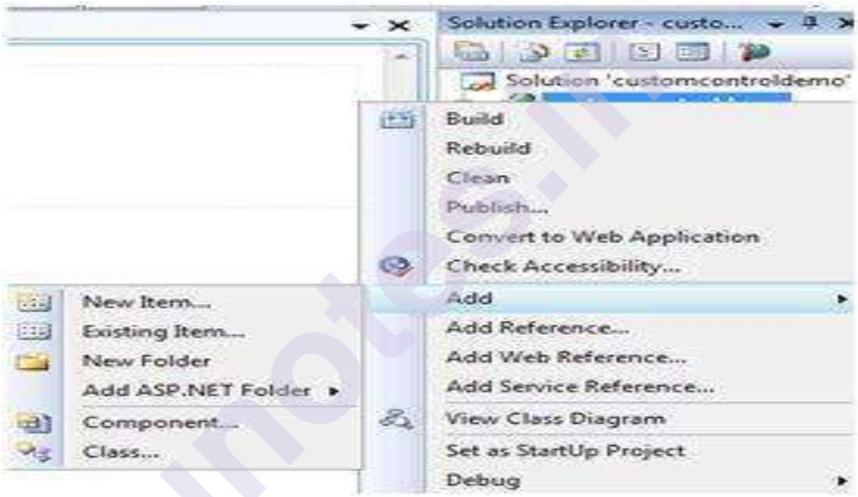
User controls behaves like a miniature ASP.NET pages and webforms which could be used by many other pages. These are derived from the

System.Web.UI.UserControl Class. These controls have the following characteristics:

- They have an .ascx extension.
- They may not contain any <html>, <body>, or <form> tags.
- They have a Control directive instead of a Page directive.

To understand the concept, let us create a simple user control, which will work as footer for the web pages. To create and use the user control, take the following steps:

- Create a new web application.
- Right click on the project folder on the Solution Explorer and choose Add New Item.



- Select Web User Control from the Add New Item dialog box and name it footer.ascx. Initially, the footer.ascx contains only a Control directive.

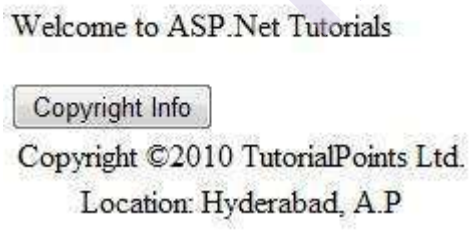
footer.ascx

```
<%@ Control Language="C#" AutoEventWireup="true"
CodeBehind="footer.ascx.cs" Inherits="customcontroldemo.footer" %>
<table>
  <tr>
    <td align="center"> Copyright ©2010 TutorialPoints Ltd.</td>
  </tr>
  <tr>
    <td align="center"> Location: Hyderabad, A.P </td>
  </tr>
</table>
```

To add the user control to your web page, you must add the Register directive and an instance of the user control to the page. The following code shows the content file:

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeBehind="Default.aspx.cs" Inherits="customcontroldemo._Default"
%>
  <%@ Register Src="~/footer.ascx" TagName="footer"
TagPrefix="Tfooter" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
  <head runat="server">
    <title>
      Untitled Page
    </title>
  </head>
  <body>
    <form id="form1" runat="server">
      <div>
        <asp:Label ID="Label1" runat="server" Text="Welcome to
ASP.Net Tutorials "></asp:Label>
        <br />  <br />
        <asp:Button ID="Button1" runat="server"
onclick="Button1_Click"  Text="Copyright Info" />
      </div>
      <footer:footer ID="footer1" runat="server" />
    </form>
  </body>
</html>
```

When executed, the page shows the footer and this control could be used in all the pages of your website.



Welcome to ASP.Net Tutorials

Copyright ©2010 TutorialPoints Ltd.
Location: Hyderabad, A.P

5.8.2 Dynamic Graphics

Graphics device interface resides in System.Drawing.dll assembly. Graphics classes are System.Drawing, System.Text, System.Printing, System.Internal, System.imaging, System.Drawing2D and System.Design name spaces.

The Graphics Class

The Graphics class encapsulates Graphics Device interface drawing surfaces. Before drawing any object we have to create surface using Graphics class

Methods	Description
DrawArc	Draws an arc from the specified ellipse
DrawBeizer	Draws a cubic beizer curve
DrawBeizers	Draws a series of cubic Beizer curves
DrawClosedCurve	Draw closed curve defined by array of points
DrawCurve	Draws a curve defined by an array of points
DrawEllipse	Draws an ellipse
DrawImage	Draws an Image
DrawLine	Draws a line
DrawPath	Draws the lines and curves defined by a GraphicsPath
DrawPie	Draws the outline of a pie section
FillEllipse	Fills the interior of an ellipse defined by bounding rectangle.
FillPath	Fills the interior of a path
FillPie	Fills the interior of a pie section
FillPolygon	Fills the interior of a polygon defined by an array of points.
FillRectangle	Fills the interior of a series of rectangles with a Brush
FillRegion	Fills the interior of a Region

Graphics Objects

After creating a Graphics object, You can use it draw lines, fill shapes and draw text so on, The major objects are as follows:-

Methods	Description
Brush	Used to fill enclosed surfaces with patterns, colors or bitmaps
Pen	Used to draw lines and polygons including rectangles, arcs and pies
Font	Used to describe the font to be used to render text
Color	Used to describe the color used to render a particular object

5.8.3 The Chart control

The chart control can create chart images of different types with many formatting options and labels. It can create standard charts like area charts, bar charts, column charts, line charts and pie charts along with more specialized charts like stock charts with the provided data.

To fill the data for chart we can use datasource option for the database values with the help of visual studio or we can provide the data from file or collection.

The Chart.Series collection contains all data series(Series objects) in the chart control. Each series is assigned the following:

- A chart type(the Series.ChartType property)
- A chart area(the Series.ChartArea property)
- A legend(the Series.Legend property),if applicable
- An X axis(the Series.XaxisType property)
- A Y axis(the Series.XaxisType property)

Each series contains a collection of Datapoint objects(the Series.Points collection property). Each data point contains :

- An X value(the DataPoint.Xvalue property)
- One or more Y values(the DataPoint.Yvalues property)
-

Chart Properties and Methods

- New Chart - Creates a new chart object and sets its width and height.
- AddTitle() – This method specifies the chart title
- AddSeries() – This method adds data to the chart
- chartType – This parameter defines the type of chart
- xValues – This parameter defines x-axis names
- yValues - This parameter defines the y-axis values
- Write() - method displays the chart

5.9 WEBSITE NAVIGATION

Maintaining the menu of a large web site is difficult and time consuming.

In ASP.NET the menu can be stored in a file to make it easier to maintain. This file is normally called **web.sitemap**, and is stored in the root directory of the web.

Different Navigation Controls in ASP.NET

5.9.1 SiteMapPath Control:-

Site maps are XML files which are mainly used to describe the logical structure of the web application. It defines the layout of all pages in web application and how they relate to each other. Whenever you want you can add or remove pages to your site map there by managing navigation of website efficiently. Site map files are defined with .sitemap extension. <sitemap> element is the root node of the sitemap file.

It has three attributes:

- **Title:** It provides textual description of the link.
- **URL:** It provides the location of the valid physical file.
- **Description:** It is used for tooltip of the link.

Properties of SiteMapPath Control:

- **PathSeparator:** This property is to get or set the out separator text.
- **NodeStyle:** This property is used to set the style of all nodes that will be displayed.
- **RootNodeStyle:** This property is used to set the style on the absolute root node.
- **PathDirection:** This property is used to set the direction of the links generated in the output.
- **CurrentNodeStyle:** This property is used to set the style on node that represent the current page.
- **ShowToolTips:** This property is used to set the tooltip for the control. Default value is true.
- **PathSeparatorStyle:** This property is used to set the style of path separator.
- **pathSeparator:** This property is to get or set the out separator text.
- **NodeStyle:** This property is used to set the style of all nodes that will be displayed.
- **RootNodeStyle:** This property is used to set the style on the absolute root node.

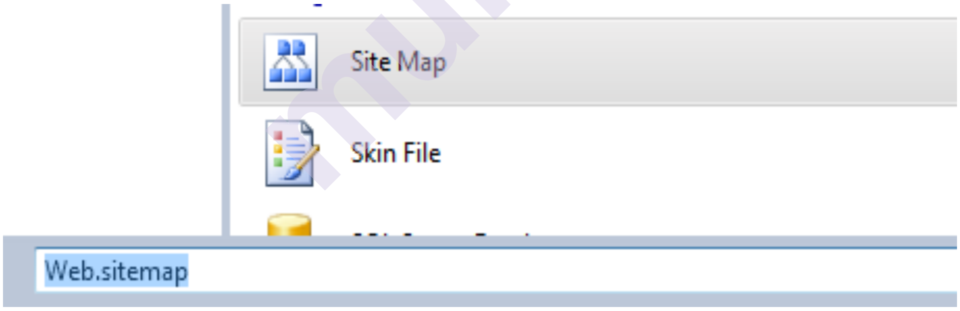
- **PathDirection:** This property is used to set the direction of the links generated in the output.
- **CurrentNodeStyle:** This property is used to set the style on node that represent the current page.
- **ShowToolTips:** This property is used to set the tooltip for the control. Default value is true.
- **PathSeparatorStyle:** This property is used to set the style of path separator.

Creation of Site Map

Below is the HTML Markup of the Master Page Main.Master that contains the SiteMapPath control as well as the SiteMapDataSource control.

```
<asp:SiteMapDataSource ID="SiteMapDataSource1" runat="server" Sho
wStartingNode="true" />

<asp:SiteMapPath ID="SiteMapPath1" runat="server" PathSeparator=">"
RenderCurrentNodeAsLink="false">

</asp:SiteMapPath>

<hr />

<asp:ContentPlaceHolder ID="ContentPlaceHolder1" runat="server">

</asp:ContentPlaceHolder>
```

Sitemap can be added using AddNewItemDialog of Visual Studio as shown below.



5.9.2 The Menu Control

The Menu control is used to create a menu of hierarchical data that can be used to navigate through the pages. The Menu control conceptually contains two types of items.

- StaticMenu that is always displayed on the page
- DynamicMenu that appears when opens the parent item.

Its properties like BackColor, ForeColor, BorderColor, BorderStyle, BorderWidth, Height etc. are implemented through style properties of <table, tr, td/> tag.

Following are some important properties that are very useful.

Properties	Description
DataSourceID	Indicates the data source to be used (You can use .sitemap file as datasource).
Text	Indicates the text to display in the menu.
Tooltip	Indicates the tooltip of the menu item when you mouse over.
Value	Indicates the nondisplayed value (usually unique id to use in server side events
NavigateUrl	Indicates the target location to send the user when menu item is clicked. If not set you can handle MenuItemClick event to decide what to do.
Target	If NavigationUrl property is set, it indicates where to open the target location (in new window or same window).
Selectable	true/false. If false, this item can't be selected. Usually in case of this item has some child.
ImageUrl	Indicates the image that appears next to the menu item.
ImageToolTip	Indicates the tooltip text to display for image next to the item.
PopOutImageUrl	Indicates the image that is displayed right to the menu item when it has some subitems
Target	If NavigationUrl property is set, it indicates where to open the target location (in new window or same window).

Styles of Menu Control

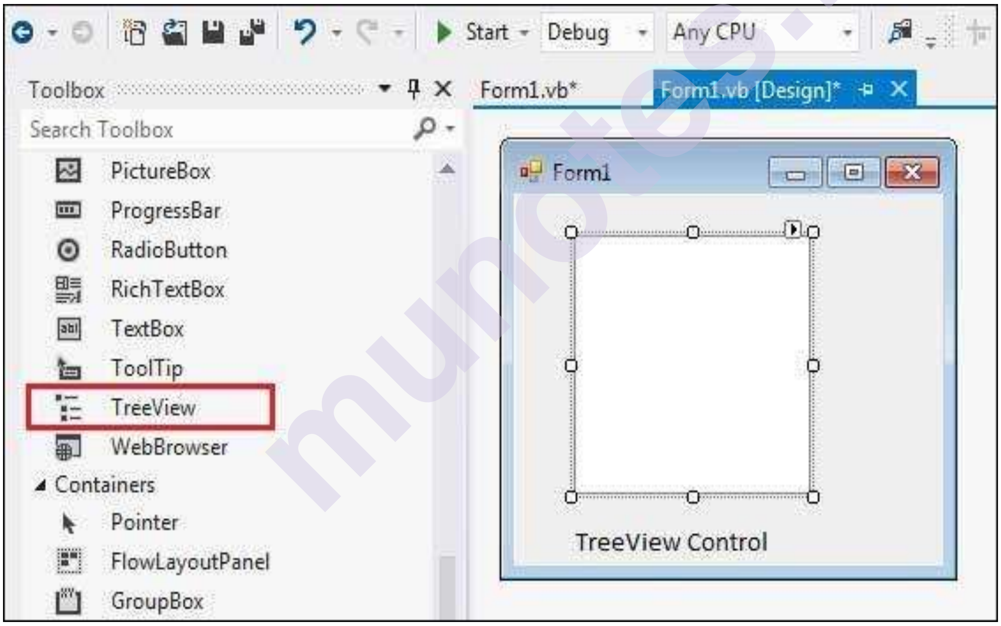
Properties	Description
StaticMenuStyle	Sets the style of the parent box in which all menu items appears.
DynamicMenuStyle	Sets the style of the parent box in which dynamic menu items appears.
StaticMenuItemStyle	Sets the style of the individual static menu items.

DynamicMenuItemStyle	Sets the style of the individual dynamic menu items
StaticSelectedStyle	Sets the style of the selected static items.
DynamicSelectedStyle	Sets the style of the selected dynamic items.
StaticHoverStyle	Sets the mouse hovering style of the static items.
DynamicHoverStyle	Sets the mouse hovering style of the dynamic items (subitems).

5.9.3 The Tree Control

The TreeView control is used to display hierarchical representations of items similar to the ways the files and folders are displayed in the left pane of the Windows Explorer. Each node may contain one or more child nodes.

Let's click on a TreeView control from the Toolbox and place it on the form.



Now we have to set Data Source property of this control to SiteMapdataSource1.

Properties of Tree View Control

Properties	**Description**
BackColor	Gets or sets the background color for the control.
BackgroundImage	Gets or set the background image for the TreeView control.

BackgroundImageLayout	Gets or sets the layout of the background image for the TreeView control.
BorderStyle	Gets or sets the border style of the tree view control.
CheckBoxes	Gets or sets a value indicating whether check boxes are displayed next to the tree nodes in the tree view control.
DataBindings	Gets the data bindings for the control.
Font	Gets or sets the font of the text displayed by the control.
FontHeight	Gets or sets the height of the font of the control.
ForeColor	The current foreground color for this control, which is the color the control uses to draw its text.

5.10 QUESTIONS:-

1. What are different types of WebServer controls?
2. Write short notes on
   a) Units
   b) Enumeration
   c) Colors
   d) Fonts
3. Explain in detail List controls.
4. What are Table controls?
5. What is an Autopostback method in ASP.Net?
6. What is Life Cycle of a Web Page?
7. Write a short note on Validation Controls.
8. Explain in detail Calendar control.
9. What are User Controls?
10. Explain in detail Graphic Class .
11. What are different Navigation controls in ASP.Net?

5.11 REFERENCES

https://www.tutorialride.com/asp-net/list-controls-in-asp-net.htm

https://w3schools.sinsixx.com/aspnet/control_table.asp.htm

https://www.tutorialspoint.com/asp.net/asp.net_multi_views.htm

https://www.dotnetfunda.com/tutorials/controls/menu.aspx

❄❄❄❄❄❄❄

6

ERROR HANDLING, LOGGING AND TRACING

Unit Structure

6.0 OBJECTIVE

- To learn about errors and how to handle the errors.
- To learn about exceptions and how to handle the exceptions.

Error handling refers to the anticipation, detection, and resolution of programming errors.

Exception Handling in C# is *a process to handle runtime errors*. We perform exception handling so that normal flow of the application can be maintained even after runtime errors.

In C#, exception is an event or object which is thrown at runtime. All exceptions the derived from *System.Exception* class. It is a runtime error which can be handled. If we don't handle the exception, it prints exception message and terminates the program.

6.1 INTRODUCTION

- Handling mechanism consists of try-catch blocks.
- A **try** statement specifies a code block subject to error-handling or cleanup code.
- The try block must be followed by a **catch** block, a finally block, or both. The catch block executes when an error occurs in the try block.
- The **finally** block executes after execution leaves the try block (or if present, the catch block), to perform cleanup code, whether or not an error occurred.
- A catch block has access to an Exception object that contains information about the error. You use a catch block to either compensate for the error or **rethrow** the exception.
- You rethrow an exception if you merely want to log the problem, or if you want to rethrow a new, higher-level exception type.
- A finally block adds determinism to your program, by always executing no matter what. It's useful for cleanup tasks such as closing network connections.

6.2 AN OVERVIEW

An exception is a problem that arises during the exception of a program. A C# exception is a response to an exceptional circumstance that arises while a program is running, such as an attempt to divide by zero. Exceptions provide a way to transfer control from one part of a program to another. C# exception handling is built upon four keywords: try, catch, finally and throw. –

- **try -** A try block identifies a block of code for which particular exceptions is activated. It is followed by one or more catch blocks.
- **catch -** A program catches an exception with an exception handler at the place in a program where you want to handle the problem. The catch keyword indicates the catching of an exception.
- **finally -** The finally block is used to execute a given set of statements, whether an exception is thrown or not thrown. For example, if you open a file, it must be closed whether an exception is raised or not.
- **throws -** A program throws an exception when a problem shows up. This is done using a throw keyword.

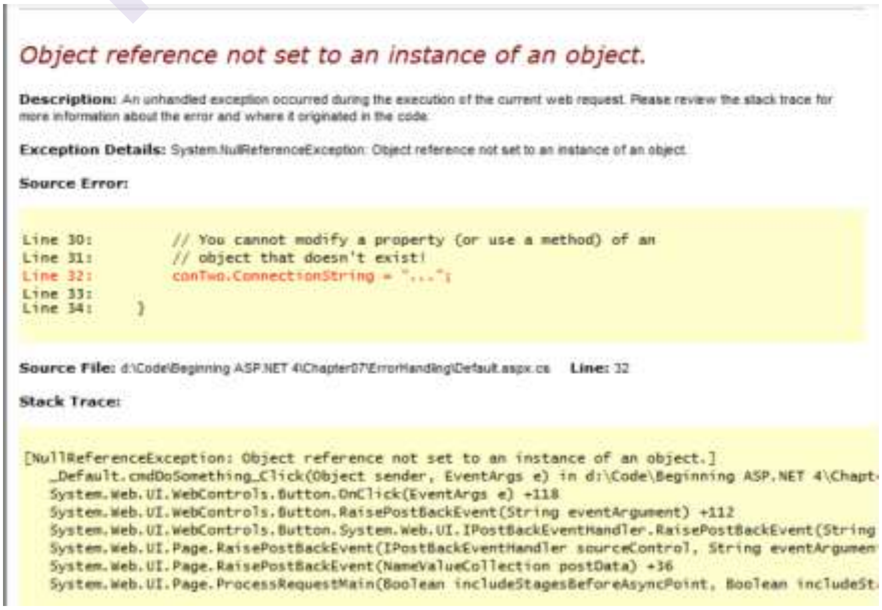
6.3 AVOIDING COMMON ERRORS

- Errors can occur in a variety of situations. Some of the most common causes of errors include attempts to divide by zero (usually caused by invalid input or missing information) and attempts to connect to a limited resource such as a file or a database (which can fail if the file doesn’t exist, the database connection times out, or the code has insufficient security credentials)

- One infamous type of error is the null reference exception, which usually occurs when a program attempts to use an uninitialized object. As a .NET programmer, you’ll quickly learn to recognize and resolve this common but annoying mistake.
- The following code example shows the problem in action, with two SqlConnection objects that represent database connections:

```
// Define a variable named conOne and create the object.
private SqlConnection conOne = new SqlConnection();
 // Define a variable named conTwo, but don't create the object.
private SqlConnection conTwo;
public void cmdDoSomething_Click(object sender, EventArgs e)
{
// This works, because the object has been created with the new keyword.
conOne.ConnectionString = "...";
// The following statement will fail and generate a null reference exception.
// You cannot modify a property (or use a method) of an object that doesn't exist!
conTwo.ConnectionString = "...";
}
```

When an error occurs in your code, .NET checks to see whether any error handlers appear in the current scope. If the error occurs inside a method, .NET searches for local error handlers and then checks for any active error handlers in the calling code. If no error handlers are found, the page processing is aborted.



Object reference not set to an instance of an object.

Description: An unhandled exception occurred during the execution of the current web request. Please review the stack trace for more information about the error and where it originated in the code.

Exception Details: System.NullReferenceException: Object reference not set to an instance of an object.

Source Error:

```
Line 30:         // You cannot modify a property (or use a method) of an
Line 31:         // object that doesn't exist!
Line 32:         conTwo.ConnectionString = "...";
Line 33:
Line 34:     }
```

Source File: d:\Code\Beginning ASP.NET 4\Chapter07\ErrorHandling\Default.aspx.cs **Line:** 32

Stack Trace:

```
[NullReferenceException: Object reference not set to an instance of an object.]
   _Default.cmdDoSomething_Click(Object sender, EventArgs e) in d:\Code\Beginning ASP.NET 4\Chapt
   System.Web.UI.WebControls.Button.OnClick(EventArgs e) +118
   System.Web.UI.WebControls.Button.RaisePostBackEvent(String eventArgument) +112
   System.Web.UI.WebControls.Button.System.Web.UI.IPostBackEventHandler.RaisePostBackEvent(String
   System.Web.UI.Page.RaisePostBackEvent(IPostBackEventHandler sourceControl, String eventArgumen
   System.Web.UI.Page.RaisePostBackEvent(NameValueCollection postData) +36
   System.Web.UI.Page.ProcessRequestMain(Boolean includeStagesBeforeAsyncPoint, Boolean includeSt
```

- Even if an error is the result of invalid input or the failure of a third-party component, an error page can shatter the professional appearance of any application.
- The application users end up feeling that the application is unstable, insecure, or of poor quality—and they're at least partially correct.

6.4 UNDERSTANDING EXCEPTION HANDLING

- Most .NET languages support structured exception handling. Essentially, when an error occurs in your application, the .NET Framework creates an exception object that represents the problem.
- You can catch this object by using an exception handler. If you fail to use an exception handler, your code will be aborted, and the user will see an error page.
- If you catch the exception, you can notify the user, attempt to resolve the problem, or simply ignore the issue and allow your web page code to keep running.

Structured exception handling provides several key features:

- **Exceptions are object-based:** Each exception provides a significant amount of diagnostic information wrapped into a neat object, instead of a simple message and error code. These exception objects also support an Inner Exception property that allows you to wrap a generic error over the more specific error that caused it. You can even create and throw your own exception objects.
- **Exceptions are caught based on their type:** This allows you to streamline error-handling code without needing to sift through obscure error codes.
- **Exception handlers use a modern block structure:** This makes it easy to activate and deactivate different error handlers for different sections of code and handle their errors individually.
- **Exception handlers are multilayered:** You can easily layer exception handlers on top of other exception handlers, some of which may check for only a specialized set of errors. As you'll see, this gives you the flexibility to handle different types of problems in different parts of your code, thereby keeping your code clean and organized.
- **Exceptions are a generic part of the .NET Framework:** This means they're completely cross-language compatible. Thus, a .NET component written in C# can throw an exception that you can catch in a web page written in VB.

6.4.1 The Exception Class

Every exception class derives from the base class System.Exception. The .NET Framework is full of predefined exception classes, such as NullReferenceException, IOException, SqlException, and so on. The

Exception class includes the essential functionality for identifying any type of error.

Exception Properties

Members Description

HelpLink A link to a help document, which can be a relative or fully qualified uniform resource locator (URL) or uniform resource name (URN). The .NET Framework doesn't use this property, but you can set it in your custom exceptions if you want to use it in your web page code

InnerException A nested exception. For example, a method might catch a simple file input/output (IO) error and create a higher-level "operation failed" error. The details about the original error could be retained in the InnerException property of the higher-level error.

Message A text description with a significant amount of information describing the problem.

Source The name of the application or object where the exception was raised.

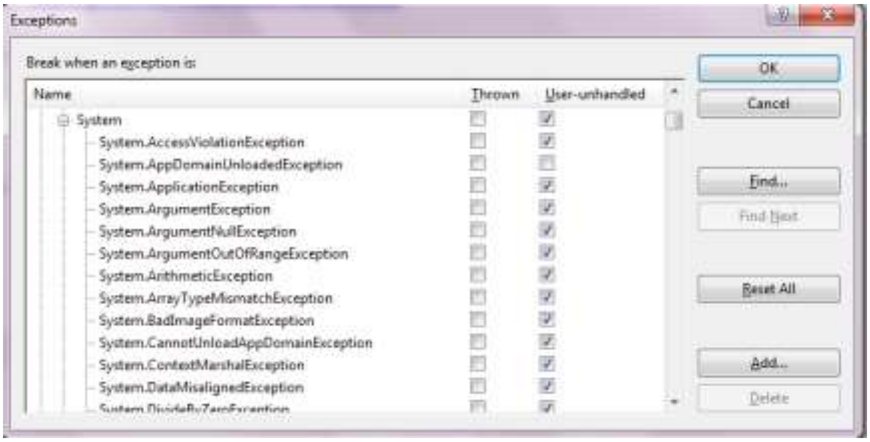
StackTrace A string that contains a list of all the current method calls on the stack, in the order of most recent to least recent. This is useful for determining where the problem occurred.

TargetSite A reflection object (an instance of the System.Reflection.Method Base class) that provides some information about the method where the error occurred. This information includes generic method details such as the method name and the data types for its parameter and return values. It doesn't contain any information about the actual parameter values that were used when the problem occurred.

GetBaseException () A method useful for nested exceptions that may have more than one layer. It retrieves the original (deepest nested) exception by moving to the base of the InnerException chain.

- When you catch an exception in an ASP.NET page, it won't be an instance of the generic System.Exception class. Instead, it will be an object that represents a specific type of error.
- This object will be based on one of the many classes that inherit from System.Exception.
- These include diverse classes such as DivideByZeroException, ArithmeticException, IOException, SecurityException, and many more. Some of these classes provide additional details about the error in additional properties.
- Visual Studio provides a useful tool to browse through the exceptions in the .NET class library.
- Click on Debug ➤ Exceptions from the menu.
- The Exceptions dialog box will appear. Expand the Common Language Runtime Exceptions group, which shows a hierarchical

tree of .NET exceptions arranged by namespace (see in following figure).



Break when an exception is:

Name	Thrown	User-unhandled
System	□	☑
System.AccessViolationException	□	☑
System.AppDomainUnloadedException	□	□
System.ApplicationException	□	☑
System.ArgumentException	□	☑
System.ArgumentNullException	□	☑
System.ArgumentOutOfRangeException	□	☑
System.ArithmeticException	□	☑
System.ArrayTypeMismatchException	□	☑
System.BadImageFormatException	□	☑
System.CannotUnloadAppDomainException	□	☑
System.ContextMarshalException	□	☑
System.DataMisalignedException	□	☑
System.DivideByZeroException	□	☑

Figure: Visual Studio's Exception Viewer

6.4.2 The Exception Chain

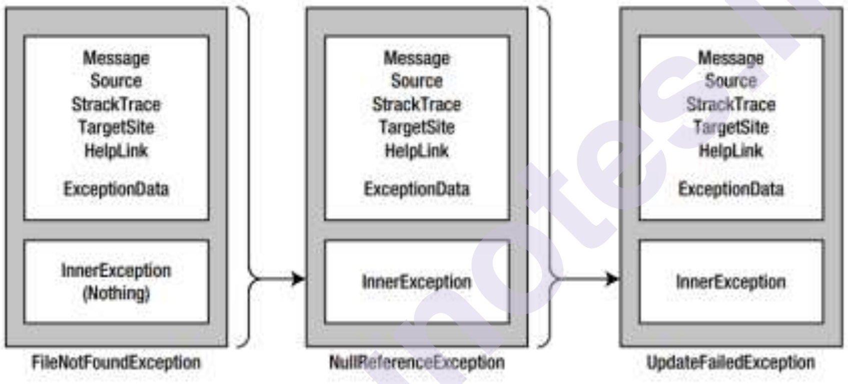


Figure: Exception chain

- In the above scenario, FileNotFoundException led to a NullReferenceException, which led to a custom Update Failed Exception.
- Using an exception-handling block, the application can catch the UpdateFailedException.
- It can then get more information about the source of the problem by following the InnerException property to the NullReferenceException, which in turn references the original FileNotFoundException.

6.5 HANDLING EXCEPTIONS

The first line of defense in an application is to check for potential error conditions before performing an operation. For example, a program can explicitly check whether the divisor is 0 before performing a calculation or whether a file exists before attempting to open it:

```
if (divisor != 0)
{
 // It's safe to divide some number by divisor.
}
 if (System.IO.File.Exists("myfile.txt"))
{
 // You can now open the myfile.txt file.
// However, you should still use exception handling because a variety of
// problems can intervene (insufficient rights, hardware failure, etc.).
}
```

Best way to handle the exceptions is by implementing try catch block which is ***structured exception handling.***

```
try
 {
 // Risky code goes here (opening a file, connecting to a database, and so on).
}
 catch
{
 // An error has been detected. You can deal with it here.
}
finally
 {
 // Time to clean up, regardless of whether or not there was an error.
}
```

- The try statement enables error handling. Any exceptions that occur in the following lines can be “caught” automatically.
- The code in the catch block will be executed when an error is detected. And either way, whether a bug occurs or not, the finally block of the code will be executed last.
- This allows you to perform some basic cleanup, such as closing a database connection. The finally code is important because it will execute even if an error has occurred that will prevent the program from continuing.
- In other words, if an unrecoverable exception halts your application, you’ll still have the chance to release resources.

A try statement looks like this:

```
try
{
... // exception may get thrown within execution of this block
}
catch (ExceptionA e)
{
... // handle exception of type ExceptionA
}
catch (ExceptionB e)
{
... // handle exception of type ExceptionB
}
finally
{
... // cleanup code
}
```

Example

```
FileStream s = null;
try
{
s = new FileStream(curName, FileMode.Open);
...
}
catch (FileNotFoundException e)
{
Console.WriteLine("file {0} not found", e.FileName);
}
catch (IOException)
{
Console.WriteLine("some IO exception occurred");
} catch
{
Console.WriteLine("some unknown error occurred");
}
finally
{
if (s != null) s.Close();
}
```

6.5.1 Catch Specific Exceptions

- Structured exception handling is particularly flexible because it allows you to catch specific types of exceptions.
- To do so, you add multiple catch statements, each one identifying the type of exception (and providing a new variable to catch it in), as follows:

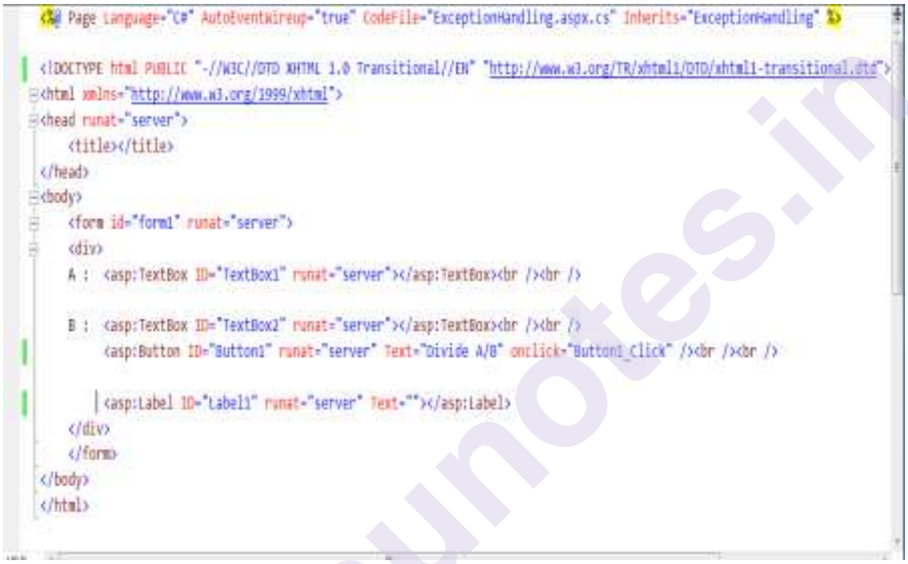
```
try
 {
 // Risky database code goes here.
}
catch (System.Data.SqlClient.SqlException err)
{
 // Catches common problems like connection errors.
 }
 catch (System.NullReferenceException err)
 {
 // Catches problems resulting from an uninitialized object.
}
catch (System.Exception err)
{
// Catches any other errors.
}
```

6.5.2 Using Nested Exception Handlers

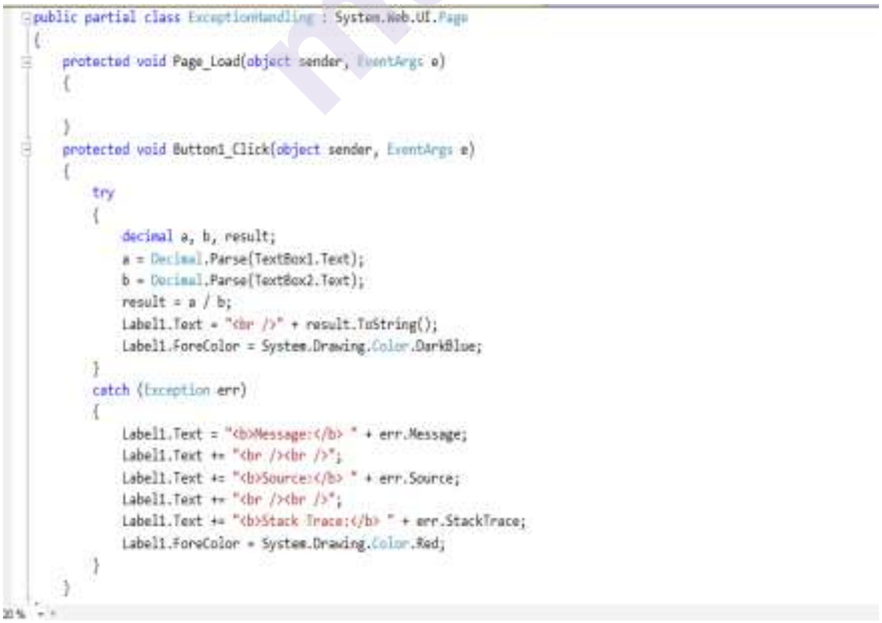
- When an exception is thrown, .NET tries to find a matching catch statement in the current method.
- If the code isn't in a local structured exception block or if none of the catch statements matches the exception, .NET will move up the call stack one level at a time, searching for active exception handlers.
- Consider the example shown here, where the Page.Load event handler calls a private DivideNumbers() method:

```
protected void Page_Load(object sender, EventArgs e)
{
 try
{
 DivideNumbers(5, 0);
}
 catch (DivideByZeroException err)
{
 // Report error here.
}
}
 private decimal DivideNumbers(decimal number, decimal divisor)
{
 return number/divisor;
}
```

6.5.3 Exception Handling in Action

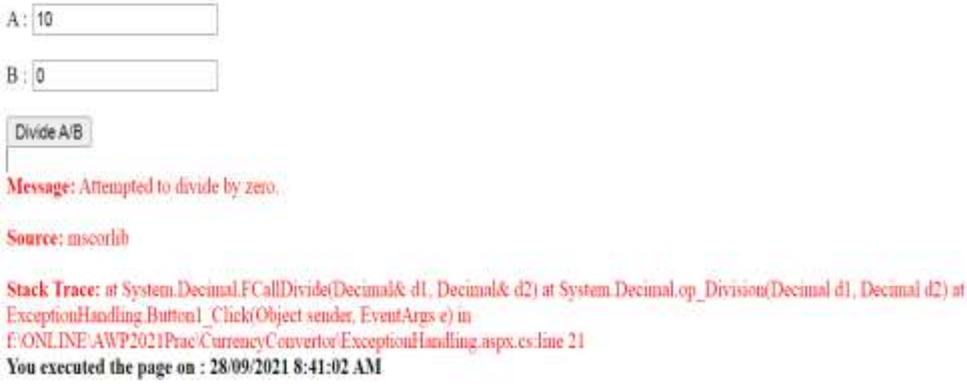


```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="ExceptionHandling.aspx.cs" Inherits="ExceptionHandling" %>

<!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>
    A :  <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox><br /><br />

    B :  <asp:TextBox ID="TextBox2" runat="server"></asp:TextBox><br /><br />
         <asp:Button ID="Button1" runat="server" Text="Divide A/B" onclick="Button1_Click" /><br /><br />

         <asp:Label ID="Label1" runat="server" Text=""></asp:Label>
    </div>
    </form>
</body>
</html>
```



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

    }
    protected void Button1_Click(object sender, EventArgs e)
    {
        try
        {
            decimal a, b, result;
            a = Decimal.Parse(TextBox1.Text);
            b = Decimal.Parse(TextBox2.Text);
            result = a / b;
            Label1.Text = "<br />" + result.ToString();
            Label1.ForeColor = System.Drawing.Color.DarkBlue;
        }
        catch (Exception err)
        {
            Label1.Text = "<b>Message:</b> " + err.Message;
            Label1.Text += "<br /><br />";
            Label1.Text += "<b>Source:</b> " + err.Source;
            Label1.Text += "<br /><br />";
            Label1.Text += "<b>Stack Trace:</b> " + err.StackTrace;
            Label1.ForeColor = System.Drawing.Color.Red;
        }
    }
```

Output



A : 10

B : 0

Message: Attempted to divide by zero.

Source: mscorlib

Stack Trace: at System.Decimal.FCallDivide(Decimal& d1, Decimal& d2) at System.Decimal.op_Division(Decimal d1, Decimal d2) at ExceptionHandling.Button1_Click(Object sender, EventArgs e) in f:\ONLINE\AWP2021Prac\CurrencyConvertor\ExceptionHandling.aspx.cs:line 21
You executed the page on : 28/09/2021 8:41:02 AM

6.5.4 Mastering Exceptions

Remember these points when working with structured exception handling:

- **Break down your code into multiple try/catch blocks:** If you put all your code into one exception handler, you'll have trouble determining where the problem occurred. You have no way to "resume" the code in a try block.

  The rule of thumb is to use one exception handler for one related task (such as opening a file and retrieving information).

- **Report all errors:** During debugging, portions of your application's error-handling code may mask easily correctable mistakes in your application. To prevent this from happening, make sure you report all errors, and consider leaving out some error handling logic in early builds.

- **Don't use exception handlers for every statement:** Simple code statements (assigning a constant value to a variable, interacting with a control, and so on) may cause errors during development testing but will not cause any future problems once perfected. Error handling should be used when you're accessing an outside resource or dealing with supplied data that you have no control over.

6.6 THROWING YOUR OWN EXCEPTIONS

- You can also define your own exception objects to represent custom error conditions.

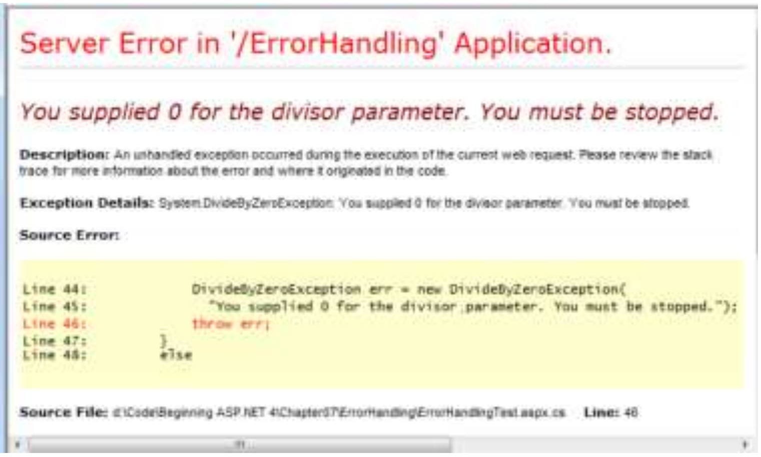
- All you need to do is create an instance of the appropriate exception class and then use the throw statement.

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

{

 try { DivideNumbers(5, 0);

}
```

```
 catch (DivideByZeroException err)
 {
 // Report error here.
}
}
private decimal DivideNumbers(decimal number, decimal divisor)
{
 if (divisor == 0)
 {
 DivideByZeroException err = new DivideByZeroException();
 throw err;
 }
 else
 {
 return number/divisor;
}
}
```

Alternatively, you can create a .NET exception object and specify a custom error message by using a different **constructor**:

```
private decimal DivideNumbers(decimal number, decimal divisor)
{
 if (divisor == 0)
 {
 DivideByZeroException err = new DivideByZeroException( "You supplied 0 for the divisor parameter. You must be stopped.");
throw err;
 }
 else
{
 return number/divisor;
}
 }
```



Server Error in '/ErrorHandling' Application.

You supplied 0 for the divisor parameter. You must be stopped.

Description: An unhandled exception occurred during the execution of the current web request. Please review the stack trace for more information about the error and where it originated in the code.

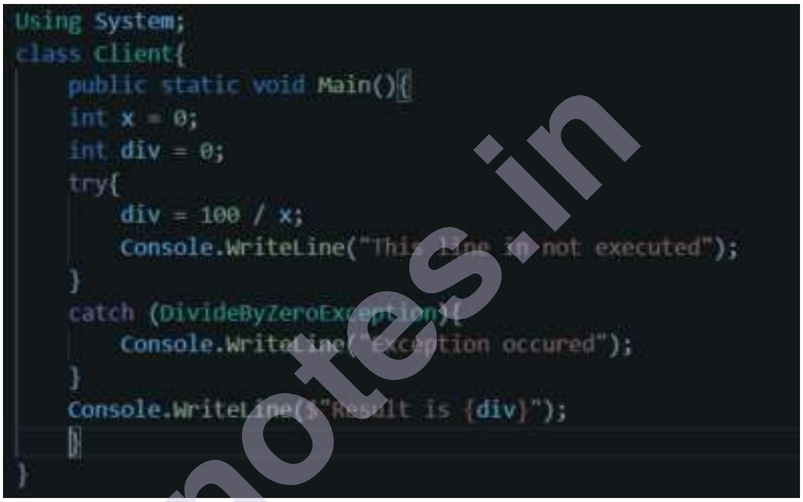
Exception Details: System.DivideByZeroException: You supplied 0 for the divisor parameter. You must be stopped.

Source Error:

```
Line 44:            DivideByZeroException err = new DivideByZeroException(
Line 45:              "You supplied 0 for the divisor parameter. You must be stopped.");
Line 46:            throw err;
Line 47:        }
Line 48:        else
```

Source File: d:\Code\Beginning ASP.NET 4\Chapter07\ErrorHandling\ErrorHandlingTest.aspx.cs   **Line:** 46

Example of Exception handling using Try Catch block



```
Using System;
class Client{
    public static void Main(){
    int x = 0;
    int div = 0;
    try{
        div = 100 / x;
        Console.WriteLine("This line in not executed");
    }
    catch (DivideByZeroException){
        Console.WriteLine("Exception occured");
    }
    Console.WriteLine($"Result is {div}");
    }
}
```

Output:

Exception occurred

Result is 0

6.7 CUSTOM EXCEPTION / USER-DEFINED EXCEPTION

- We have seen built-in exception classes however; we often like to raise an exception when the business rule of our application gets violated. So, for this we can create a custom exception class by deriving Exception or Application Exception class.
- C# allows us to create user-defined or custom exception. It is used to make the meaningful exception.
- Custom exception classes should always inherit from System. Application Exception, which itself derives from the base Exception class. This allows .NET to distinguish between two broad classes of exceptions—those you create and those that are native to the .NET Framework

```
public class CustomDivideByZeroException : ApplicationException
{
 // Add a variable to specify the "other" number.
// This might help diagnose the problem. public decimal DividingNumber;
}
```

You can throw this custom exception like this:

```
private decimal DivideNumbers(decimal number, decimal divisor)
{
 if (divisor == 0)
{
CustomDivideByZeroException err = new CustomDivideByZeroException ();
err.DividingNumber = number;
throw err;
}
 else
{
 return number/divisor;
}
}
```

To perfect the custom exception, you need to supply it with the three standard constructors.

This allows your exception class to be created in the standard ways that every exception supports:

- On its own, with no arguments
- With a custom message
- With a custom message and an exception object to use as the inner exception

```
public class CustomDivideByZeroException : ApplicationException
{
 // Add a variable to specify the "other" number.
private decimal dividingNumber;
public decimal DividingNumber
{
 get
{
 return dividingNumber;
}
 set
{
 dividingNumber = value;
}
}
public CustomDivideByZeroException() : base()
{}
public CustomDivideByZeroException(string message) : base(message)
{}
 public CustomDivideByZeroException(string message, Exception inner) : base(message, inner)
{}
}
```

One more Example of User-defined Exception

For example, create InvalidStudentNameException class in a school application, which does not allow any special character or numeric value in a name of any of the students.

```
class Student
{
public int StudentID
{ get; set; }
public string StudentName
{ get; set; }
}
class InvalidStudentNameException : Exception
{
public InvalidStudentNameException()
{}
public InvalidStudentNameException (string name) : base(String.Format("InvalidStudentName: {0}", name))
{ }
}
class Program
{
static void Main(string[] args)
{
Student newStudent = null;
try
{
newStudent = new Student();
newStudent.StudentName = "James007";
ValidateStudent(newStudent);
}
catch(InvalidStudentNameException ex)
{
Console.WriteLine(ex.Message );
}
Console.ReadKey();
}
private static void ValidateStudent(Student std)
{
Regex regex = new Regex("^[a-zA-Z]+$");
if (!regex.IsMatch(std.StudentName))
throw new InvalidStudentNameException(std.StudentName);
}
}
```

6.8 PAGE TRACING

- Visual Studio's debugging tools and ASP.NET's detailed error pages are extremely helpful when you're testing an application. However, sometimes you need a way to identify problems after you've deployed an application, when you don't have Visual Studio to rely on.
- When you don't have Visual Studio to rely on. You could try to identify these errors by recording diagnostic information in an event log, but this assumes that someone will actually review the log regularly.
- Otherwise, your website users could see strange debugging messages when they least expect it.
- ASP. NET provides a feature called **tracing** that gives you a far more convenient and flexible way to report diagnostic information.

6.8.1 Enabling Tracing

To use tracing, you need to explicitly enable it. There are several ways to switch on tracing. One of the easiest ways is by adding an attribute to the Page directive in the .aspx file:

<%@ Page Trace=”true”   ….. %>

You can also enable tracing by using the built-in Trace object (which is an instance of the System.Web.TraceContext class).

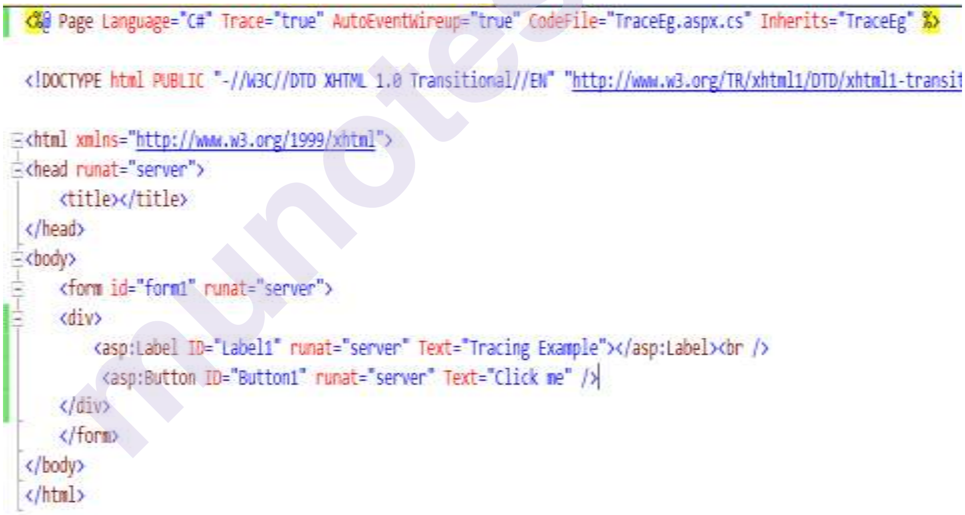
Here’s an example of how you might turn on tracing in the Page.Load event handler:

```
protected void Page_Load(Object sender, EventArgs e)
{
 Trace.IsEnabled = true;
}
```

This technique is useful because it allows you to enable or disable tracing for a page under specific circumstances that you test for in your code.

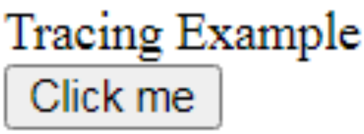
Tracing Information

- ASP.NET tracing automatically provides a lengthy set of standard, formatted information.
- This information allows you to monitor several important aspects of your application, such as the contents of the current session and the time taken to execute portions of code.



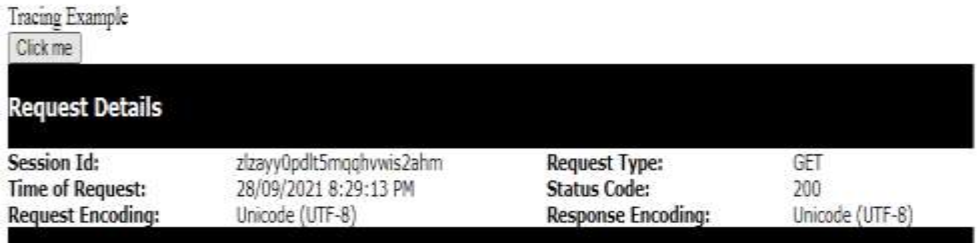
```
<%@ Page Language="C#" Trace="true" AutoEventWireup="true" CodeFile="TraceEg.aspx.cs" Inherits="TraceEg" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transit

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:Label ID="Label1" runat="server" Text="Tracing Example"></asp:Label><br />
        <asp:Button ID="Button1" runat="server" Text="Click me" />
    </div>
    </form>
</body>
</html>
```

Output



Tracing Example

- **Request Details**
- This section includes some basic information such as the current session ID, the time the web request was made, and the type of web request and encoding



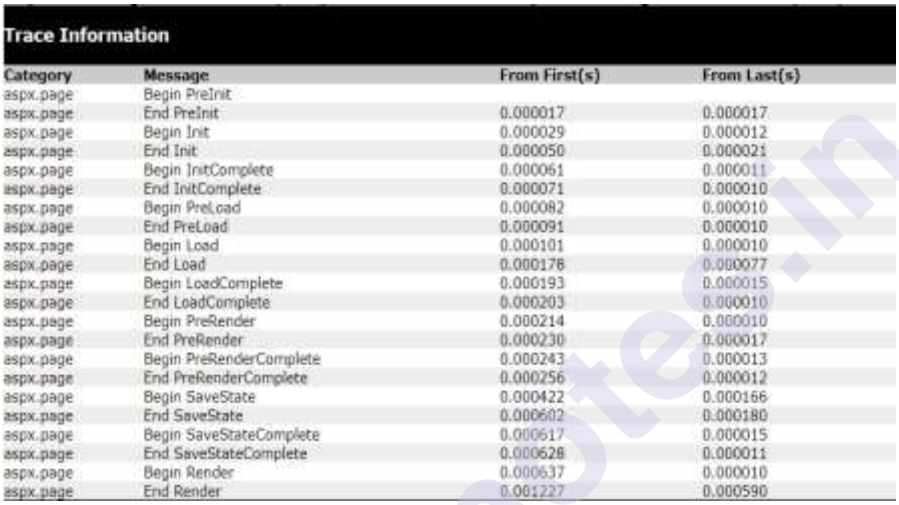
Tracing Example

Request Details

Session Id:	zlzayy0pdlt5mqqhvwis2ahm	Request Type:	GET
Time of Request:	28/09/2021 8:29:13 PM	Status Code:	200
Request Encoding:	Unicode (UTF-8)	Response Encoding:	Unicode (UTF-8)

- **Trace Information**

- Trace information shows the stages of processing that the page went through before being sent to the client

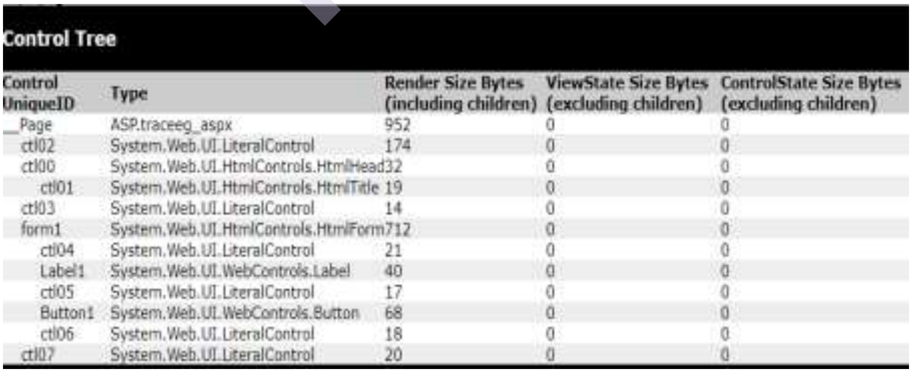


Trace Information

Category	Message	From First(s)	From Last(s)
aspx.page	Begin PreInit		
aspx.page	End PreInit	0.000017	0.000017
aspx.page	Begin Init	0.000029	0.000012
aspx.page	End Init	0.000050	0.000021
aspx.page	Begin InitComplete	0.000061	0.000011
aspx.page	End InitComplete	0.000071	0.000010
aspx.page	Begin PreLoad	0.000082	0.000010
aspx.page	End PreLoad	0.000091	0.000010
aspx.page	Begin Load	0.000101	0.000010
aspx.page	End Load	0.000178	0.000077
aspx.page	Begin LoadComplete	0.000193	0.000015
aspx.page	End LoadComplete	0.000203	0.000010
aspx.page	Begin PreRender	0.000214	0.000010
aspx.page	End PreRender	0.000230	0.000017
aspx.page	Begin PreRenderComplete	0.000243	0.000013
aspx.page	End PreRenderComplete	0.000256	0.000012
aspx.page	Begin SaveState	0.000422	0.000166
aspx.page	End SaveState	0.000602	0.000180
aspx.page	Begin SaveStateComplete	0.000617	0.000015
aspx.page	End SaveStateComplete	0.000628	0.000011
aspx.page	Begin Render	0.000637	0.000010
aspx.page	End Render	0.001227	0.000590

- **Control Tree**

- The control tree shows you all the controls on the page, indented to show their hierarchy (which controls are contained inside other controls).



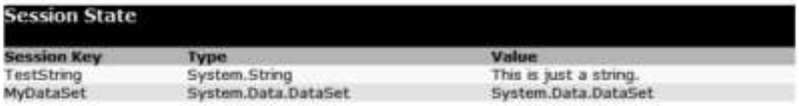
Control Tree

Control UniqueID	Type	Render Size Bytes (including children)	ViewState Size Bytes (excluding children)	ControlState Size Bytes (excluding children)
__Page	ASP.traceeg_aspx	952	0	0
ctl02	System.Web.UI.LiteralControl	174	0	0
ctl00	System.Web.UI.HtmlControls.HtmlHead	32	0	0
ctl01	System.Web.UI.HtmlControls.HtmlTitle	19	0	0
ctl03	System.Web.UI.LiteralControl	14	0	0
form1	System.Web.UI.HtmlControls.HtmlForm	712	0	0
ctl04	System.Web.UI.LiteralControl	21	0	0
Label1	System.Web.UI.WebControls.Label	40	0	0
ctl05	System.Web.UI.LiteralControl	17	0	0
Button1	System.Web.UI.WebControls.Button	68	0	0
ctl06	System.Web.UI.LiteralControl	18	0	0
ctl07	System.Web.UI.LiteralControl	20	0	0

- **Session State and Application State**

- These sections display every item that is in the current session or application state. Each item in the appropriate state collection is listed with its name, type, and value.

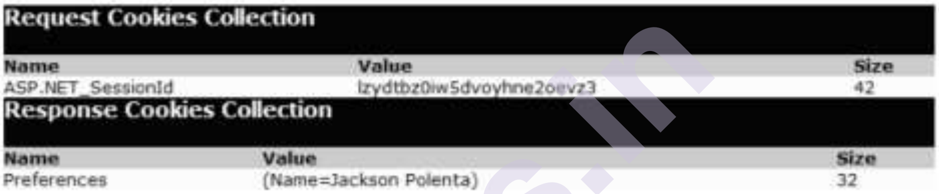
- If you’re storing simple pieces of string information, the value is straightforward—it’s the actual text in the string. If you’re storing an object, .NET calls the object’s ToString() method to get an appropriate string representation.



Session State

Session Key	Type	Value
TestString	System.String	This is just a string.
MyDataSet	System.Data.DataSet	System.Data.DataSet

- **Request Cookies and Response Cookies**
- These sections display the cookies that were sent by the web browser with the request for this page and display the cookies that were returned by the web server with the response.
- ASP.NET shows the content and the size of each cookie in bytes.



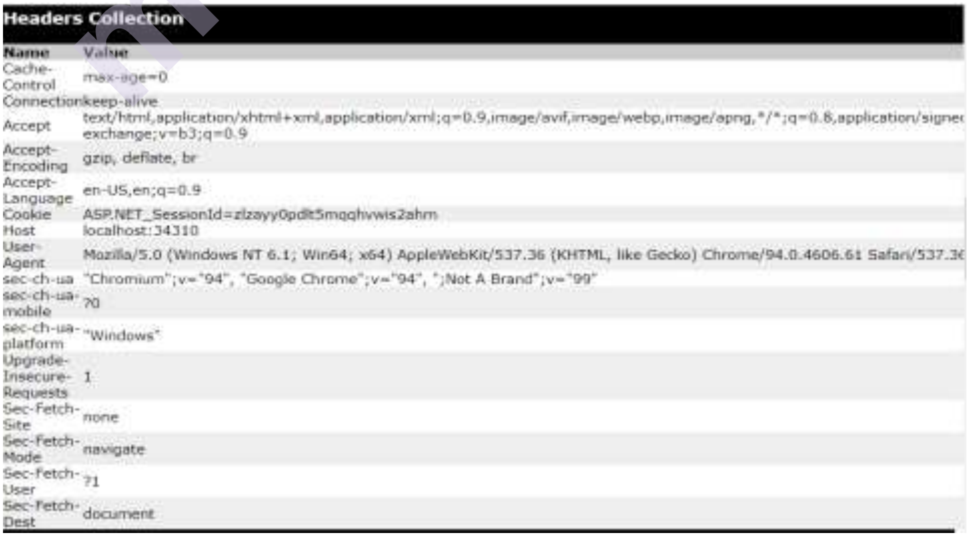
Request Cookies Collection

Name	Value	Size
ASP.NET_SessionId	lzydtbz0iw5dvoyhne2oevz3	42

Response Cookies Collection

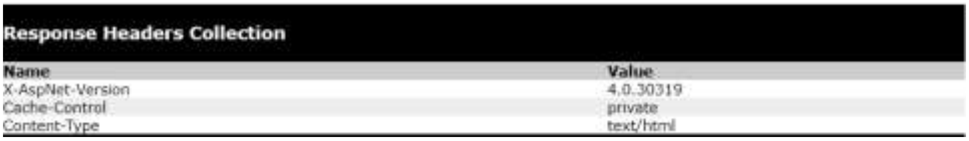
Name	Value	Size
Preferences	(Name=Jackson Polenta)	32

- **Headers Collection**
- This section lists all the HTTP headers. Technically, the headers are bits of information that are sent to the server as part of a request.
- They include information about the browser making the request, the types of content it supports, and the language it uses. In addition, the ResponseHeadersCollection lists the headers that are sent to the client as part of a response.



Headers Collection

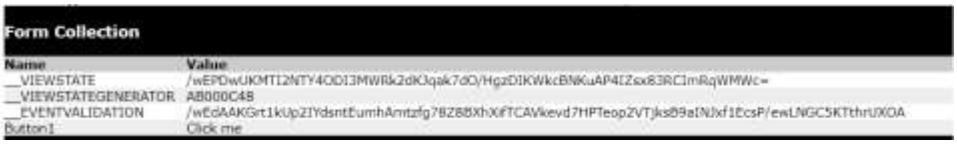
Name	Value
Cache-Control	max-age=0
Connection	keep-alive
Accept	text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Accept-Encoding	gzip, deflate, br
Accept-Language	en-US,en;q=0.9
Cookie	ASP.NET_SessionId=zlzayy0pdlt5mqqhvwis2ahm
Host	localhost:34310
User-Agent	Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/94.0.4606.61 Safari/537.36
sec-ch-ua	"Chromium";v="94", "Google Chrome";v="94", ";Not A Brand";v="99"
sec-ch-ua-mobile	?0
sec-ch-ua-platform	"Windows"
Upgrade-Insecure-Requests	1
Sec-Fetch-Site	none
Sec-Fetch-Mode	navigate
Sec-Fetch-User	?1
Sec-Fetch-Dest	document



Response Headers Collection

Name	Value
X-AspNet-Version	4.0.30319
Cache-Control	private
Content-Type	text/html

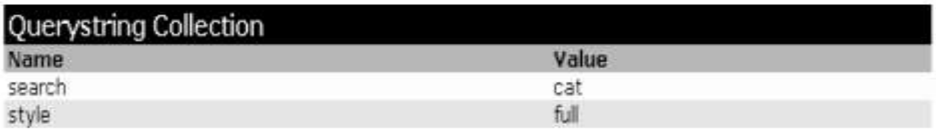
- **Form Collection**



Form Collection	
Name	Value
__VIEWSTATE	/wEPDwUKMTI2NTY4ODI3MWRk2dKJqak7dO/HgzDIKWkcBNKuAP4IZsx83RCImRqWMWc=
__VIEWSTATEGENERATOR	AB000C48
__EVENTVALIDATION	/wEdAAKGrt1kUp2IYdsntEumhAmtzfg78Z88XhXifTCAVkevd7HPTeop2VTjks89aINJxf1EcsP/ewLNGC5KTthrUXOA
Button1	Click me

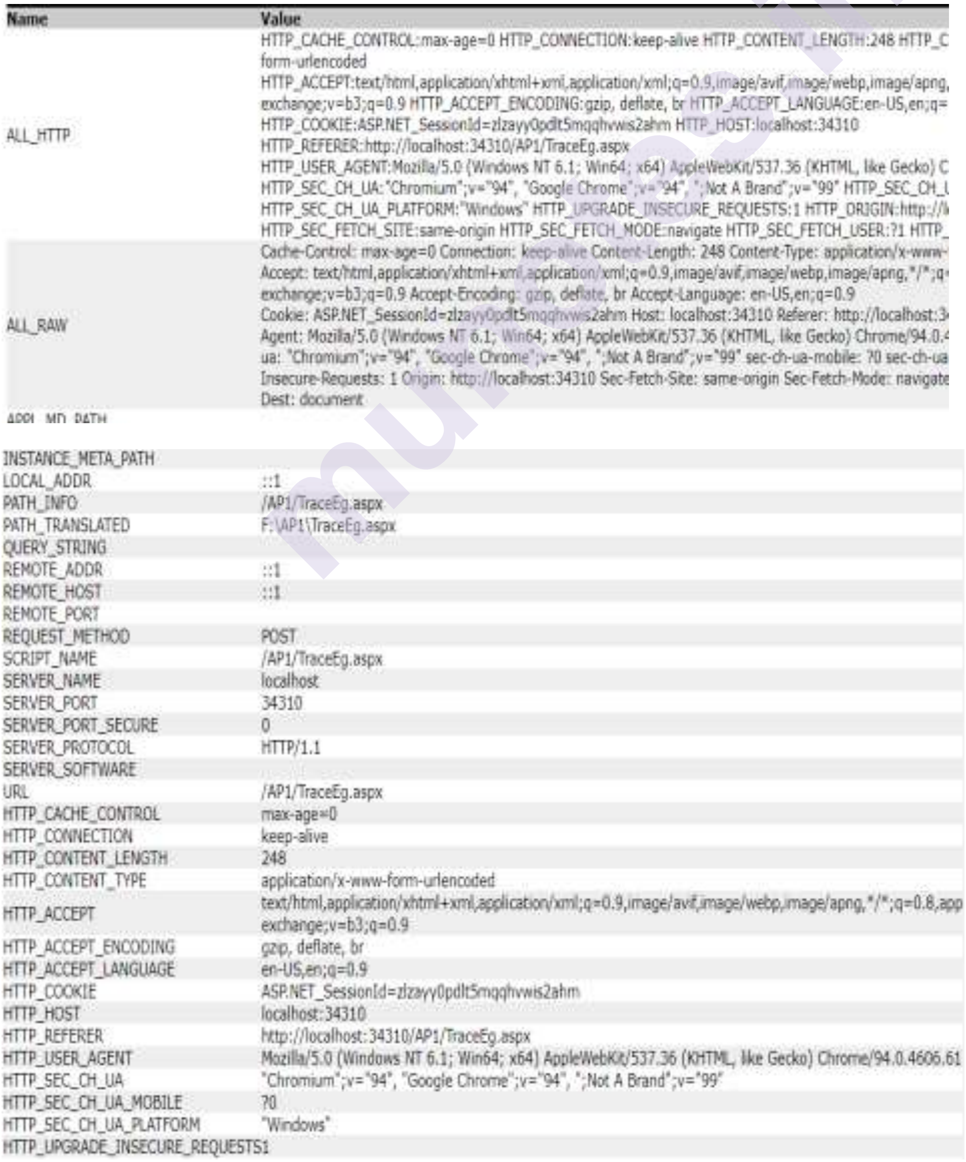
- **Query String Collection**

- This section lists the variables and values submitted in the query string. You can see this information directly in the web page URL in the address box in the browser.



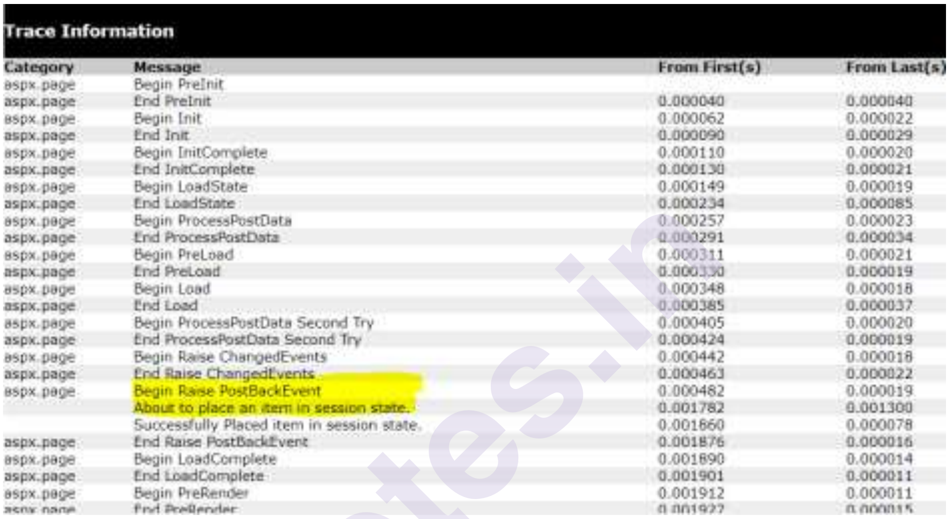
Querystring Collection	
Name	Value
search	cat
style	full

Server Variable



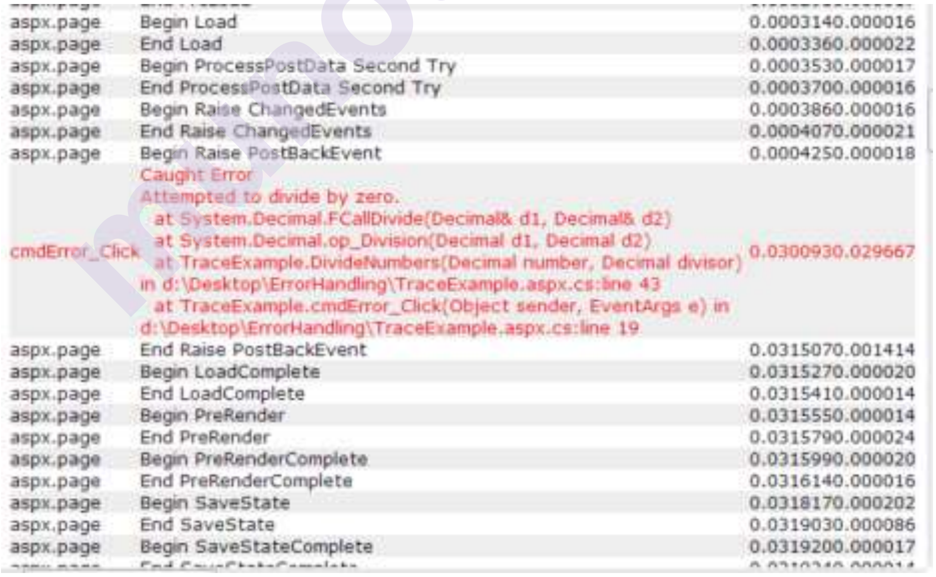
Name	Value
ALL_HTTP	HTTP_CACHE_CONTROL:max-age=0 HTTP_CONNECTION:keep-alive HTTP_CONTENT_LENGTH:248 HTTP_C form-urlencoded HTTP_ACCEPT:text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng, exchange;v=b3;q=0.9 HTTP_ACCEPT_ENCODING:gzip, deflate, br HTTP_ACCEPT_LANGUAGE:en-US,en;q= HTTP_COOKIE:ASP.NET_SessionId=zlzayy0pdlt5mqqhvwis2ahm HTTP_HOST:localhost:34310 HTTP_REFERER:http://localhost:34310/AP1/TraceEg.aspx HTTP_USER_AGENT:Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) C HTTP_SEC_CH_UA:"Chromium";v="94", "Google Chrome";v="94", ";Not A Brand";v="99" HTTP_SEC_CH_U HTTP_SEC_CH_UA_PLATFORM:"Windows" HTTP_UPGRADE_INSECURE_REQUESTS:1 HTTP_ORIGIN:http://l HTTP_SEC_FETCH_SITE:same-origin HTTP_SEC_FETCH_MODE:navigate HTTP_SEC_FETCH_USER:?1 HTTP_
ALL_RAW	Cache-Control: max-age=0 Connection: keep-alive Content-Length: 248 Content-Type: application/x-www- Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q exchange;v=b3;q=0.9 Accept-Encoding: gzip, deflate, br Accept-Language: en-US,en;q=0.9 Cookie: ASP.NET_SessionId=zlzayy0pdlt5mqqhvwis2ahm Host: localhost:34310 Referer: http://localhost:3 Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/94.0.4 ua: "Chromium";v="94", "Google Chrome";v="94", ";Not A Brand";v="99" sec-ch-ua-mobile: ?0 sec-ch-ua Insecure-Requests: 1 Origin: http://localhost:34310 Sec-Fetch-Site: same-origin Sec-Fetch-Mode: navigate Dest: document
APPL_MD_PATH	
INSTANCE_META_PATH	
LOCAL_ADDR	::1
PATH_INFO	/AP1/TraceEg.aspx
PATH_TRANSLATED	F:\AP1\TraceEg.aspx
QUERY_STRING	
REMOTE_ADDR	::1
REMOTE_HOST	::1
REMOTE_PORT	
REQUEST_METHOD	POST
SCRIPT_NAME	/AP1/TraceEg.aspx
SERVER_NAME	localhost
SERVER_PORT	34310
SERVER_PORT_SECURE	0
SERVER_PROTOCOL	HTTP/1.1
SERVER_SOFTWARE	
URL	/AP1/TraceEg.aspx
HTTP_CACHE_CONTROL	max-age=0
HTTP_CONNECTION	keep-alive
HTTP_CONTENT_LENGTH	248
HTTP_CONTENT_TYPE	application/x-www-form-urlencoded
HTTP_ACCEPT	text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,app exchange;v=b3;q=0.9
HTTP_ACCEPT_ENCODING	gzip, deflate, br
HTTP_ACCEPT_LANGUAGE	en-US,en;q=0.9
HTTP_COOKIE	ASP.NET_SessionId=zlzayy0pdlt5mqqhvwis2ahm
HTTP_HOST	localhost:34310
HTTP_REFERER	http://localhost:34310/AP1/TraceEg.aspx
HTTP_USER_AGENT	Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/94.0.4606.61
HTTP_SEC_CH_UA	"Chromium";v="94", "Google Chrome";v="94", ";Not A Brand";v="99"
HTTP_SEC_CH_UA_MOBILE	?0
HTTP_SEC_CH_UA_PLATFORM	"Windows"
HTTP_UPGRADE_INSECURE_REQUESTS1	

- **Writing Trace Information**
- In addition to the standard trace information, you can generate your own tracing messages.
- To write a custom trace message, you use the Write() method or the Warn() method of the built-in Trace object. These methods are equivalent.
- The only difference is that Warn() displays the message in red lettering, which makes it easier to distinguish from other messages in the list.



Trace Information

Category	Message	From First(s)	From Last(s)
aspx.page	Begin PreInit		
aspx.page	End PreInit	0.000040	0.000040
aspx.page	Begin Init	0.000062	0.000022
aspx.page	End Init	0.000090	0.000029
aspx.page	Begin InitComplete	0.000110	0.000020
aspx.page	End InitComplete	0.000130	0.000021
aspx.page	Begin LoadState	0.000149	0.000019
aspx.page	End LoadState	0.000234	0.000085
aspx.page	Begin ProcessPostData	0.000257	0.000023
aspx.page	End ProcessPostData	0.000291	0.000034
aspx.page	Begin PreLoad	0.000311	0.000021
aspx.page	End PreLoad	0.000330	0.000019
aspx.page	Begin Load	0.000348	0.000018
aspx.page	End Load	0.000385	0.000037
aspx.page	Begin ProcessPostData Second Try	0.000405	0.000020
aspx.page	End ProcessPostData Second Try	0.000424	0.000019
aspx.page	Begin Raise ChangedEvents	0.000442	0.000018
aspx.page	End Raise ChangedEvents	0.000463	0.000022
aspx.page	Begin Raise PostBackEvent	0.000482	0.000019
	About to place an item in session state.	0.001782	0.001300
	Successfully Placed item in session state.	0.001860	0.000078
aspx.page	End Raise PostBackEvent	0.001876	0.000016
aspx.page	Begin LoadComplete	0.001890	0.000014
aspx.page	End LoadComplete	0.001901	0.000011
aspx.page	Begin PreRender	0.001912	0.000011
aspx.page	End PreRender	0.001927	0.000015

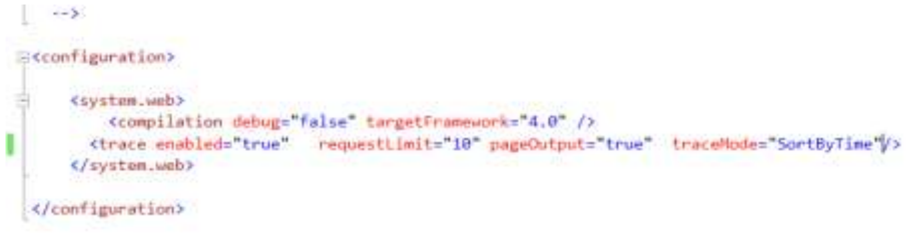


aspx.page	Begin Load	0.0003140.000016
aspx.page	End Load	0.0003360.000022
aspx.page	Begin ProcessPostData Second Try	0.0003530.000017
aspx.page	End ProcessPostData Second Try	0.0003700.000016
aspx.page	Begin Raise ChangedEvents	0.0003860.000016
aspx.page	End Raise ChangedEvents	0.0004070.000021
aspx.page	Begin Raise PostBackEvent	0.0004250.000018
cmdError_Click	Caught Error Attempted to divide by zero. at System.Decimal.FCallDivide(Decimal& d1, Decimal& d2) at System.Decimal.op_Division(Decimal d1, Decimal d2) at TraceExample.DivideNumbers(Decimal number, Decimal divisor) in d:\Desktop\ErrorHandling\TraceExample.aspx.cs:line 43 at TraceExample.cmdError_Click(Object sender, EventArgs e) in d:\Desktop\ErrorHandling\TraceExample.aspx.cs:line 19	0.0300930.029667
aspx.page	End Raise PostBackEvent	0.0315070.001414
aspx.page	Begin LoadComplete	0.0315270.000020
aspx.page	End LoadComplete	0.0315410.000014
aspx.page	Begin PreRender	0.0315550.000014
aspx.page	End PreRender	0.0315790.000024
aspx.page	Begin PreRenderComplete	0.0315990.000020
aspx.page	End PreRenderComplete	0.0316140.000016
aspx.page	Begin SaveState	0.0318170.000202
aspx.page	End SaveState	0.0319030.000086
aspx.page	Begin SaveStateComplete	0.0319200.000017

6.8.2 Application Level Tracing

- Application-level tracing allows you to enable tracing for an entire application.
- However, the tracing information won’t be displayed in the page. Instead, it will be collected and stored in memory for a short amount of time.

- You can review the recently traced information by requesting a special URL.
- To enable application-level tracing, you need to modify settings in the web.config file.



```
  -->

<configuration>

    <system.web>
       <compilation debug="false" targetFramework="4.0" />
     <trace enabled="true"   requestLimit="10" pageOutput="true"  traceMode="SortByTime"/>
    </system.web>

</configuration>
```

Attribute	Values	Description
enabled	true, false	Turns application-level tracing on or off.
requestLimit	Any integer (for example, 10)	Stores tracing information for a maximum number of HTTP requests.
pageOutput	true, false	Determines whether tracing information will be the displayed on the page.
traceMode	SortByTime, SortByCategory.	Determines the sort order of trace messages The default is SortByTime
localOnly	true, false	Determines whether tracing information will be shown only to local clients or to remote clients also.
mostRecent	true, false	Keeps only the most recent trace messages if true. When the request Limit maximum is reached the information for the oldest request is abandoned every time a new request is received.

6.9 REFERENCES

- **Beginning ASP.NET 4.5 in C# by Mathew MacDonald** – Apress Publication (2012)
- **Murach‟s ASP.NET 4.6 Web Programming in C#2015** by Anne Bohem and Joel Murach – Murach Publication (2016)
- ASP.NET 4.0 Programming by J. Kanjilal - Tata MsGrawhill Publication (2011)

6.10 UNIT END QUESTIONS

Answer the following questions

1) Explain different types of exceptions in .Net.
2) Write a shot note on Exception class and explain the properties of Exception class?
3) What is Exception chaining? Explain with examples.
4) Write a program to implement exception handling.
5) Explain DivideByZero Exception with an example.
6) Explain with examples try, catch and finally blocks in exception.
7) Explain Nested Exceptions with examples.
8) Write in detail about custom exceptions and explain with an example.
9) What is tracing? Explain page tracing in detail with examples

Explain in detail Application level tracing. Explain its properties.

❄❄❄❄❄❄❄

7

STATE MANAGEMENT

Unit Structure

7.0 OBJECTIVE

After going through this unit, you will be able to,

- Create a Web Application with Session Tracing.
- Create the Application using different types Session Management Methods.
- Know about Cookies and Session State Management.
- Create Web Application with View State Management.

7.1 UNDERSTANDING THE PROBLEM OF STATE

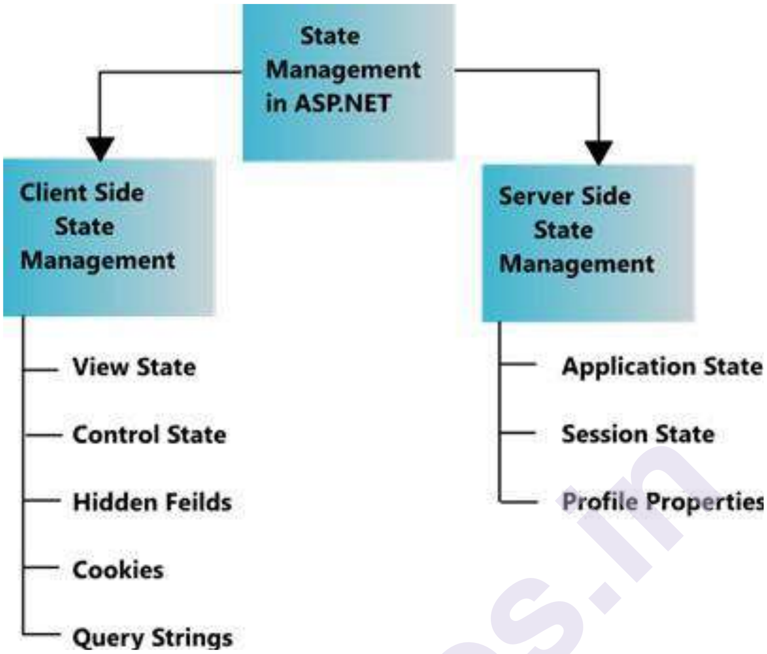
- A web application is stateless. That means that a new instance of a page is created every time when we make a request to the server to get the page and after the round trip our page has been lost immediately.

- It only happens because of one server; all the controls of the Web Page are created and after the round trip the server destroys all the instances. So, to retain the values of the controls we use state management techniques.

- In this method we will trace the user activity when user visits multiple pages.

7.2 USING VIEW STATE

The State Management techniques are classified into two categories



View State

View State is the method to preserve the Value of the Page and Controls between round trips. It is a Page-Level State Management technique. View State is turned on by default and normally serializes the data in every control on the page regardless of whether it is actually used during a post-back.

Features of View State

These are the main features of view state:

- Retains the value of the Control after post-back without using a session.
- Stores the value of Pages and Control Properties defined in the page.
- Creates a custom View State Provider that lets you store View State Information in a SQL Server Database or in another data store.

Advantages of View State

- Easy to Implement.
- No server resources are required: The View State is contained in a structure within the page load.
- Enhanced security features: It can be encoded and compressed or Unicode implementation.

Disadvantages of View State

Security Risk: The Information of View State can be seen in the page output source directly. We can manually encrypt and decrypt the contents of a Hidden Field, but it requires extra coding. If security is a concern, then consider using a Server-Based State Mechanism so that no sensitive information is sent to the client.

Performance: Performance is not good if we use a large amount of data because View State is stored in the page itself and storing a large value can cause the page to be slow.

Device limitation: Mobile Devices might not have the memory capacity to store a large amount of View State data.

It can store values for the same page only.

Example:

If we want to add one variable in View State,

ViewState["Var"]=Count;

For Retrieving information from View State

string Test=ViewState["TestVal"];

7.3 TRANSFERRING INFORMATION BETWEEN PAGES

- These are the Response objects of Asp.Net which are used to redirect page from one page to another page and true and false are the optional parameters of the Redirect methods which decides whether the current page response terminate or not.

Response.Redirect method takes the following parameter

1. **Url (string)**
2. **EndResponse (Boolean)**

In the above image you have seen that Redirect method takes the two parameters

- **Url**: This is a string parameter in which url or page name is given that helps to navigate from one page to another page.

Syntax

1. Response.Redirect("Default.aspx");

- **EndResponse**: EndResponse is the optional parameter of the Redirect method having true and false values, when you false value then it does not terminate the execution of the current page, the default is true.

Syntax

1. Response.Redirect("Default.aspx", **false**);

2. Response.Redirect("Default.aspx", **true**);

Difference between Response.Redirect() and Server.Transfer()
Both Response.Redirect and Server.Transfer methods are used to transfer a user from one web page to another web page. Both methods are used for the same purpose but still there are some differences as follows.

The Response.Redirect method redirects a request to a new URL and specifies the new URL while the Server.Transfer method for the current request, terminates execution of the current page and starts execution of a new page using the specified URL path of the page.

Both Response.Redirect and Server.Transfer has the same syntax like:

1. Response.Redirect("UserDetail.aspx");
2. Server.Transfer("UserDetail.aspx");

7.4 USING COOKIES

1. A cookie is a small piece of information stored on the client machine.
2. This file is located on client machines "C:\Document and Settings\Currently_Login user\Cookie" path.
3. It is used to store user preference information like Username, Password, City and Phone No etc. on client machines.
4. We need to import namespace called Systen.Web.HttpCookie before we use cookie

Types of Cookies:

Persist Cookie - A cookie has not had expired time which is called as Persist Cookie

Non-Persist Cookie - A cookie has expired time which is called as Non-Persist Cookie

Creation of cookies:

It's really easy to create a cookie in the Asp.Net with help of Response object or HttpCookie

Example 1:

```
HttpCookie userInfo = new HttpCookie("userInfo");
userInfo["UserName"] = "abc";
userInfo["UserColor"] = "Black";
userInfo.Expires.Add(new TimeSpan(0, 1, 0));
Response.Cookies.Add(userInfo);
```

Example 2:

```
Response.Cookies["userName"].Value = "abc";
Response.Cookies["userColor"].Value = "Black";
```

Retrieve from cookie

Its easy way to retrieve cookie value form cookies by help of Request object.

Example 1:

```
string User_Name = string.Empty;
string User_Color = string.Empty;
User_Name = Request.Cookies["userName"].Value;
User_Color = Request.Cookies["userColor"].Value;
```

Example 2:

```
string User_name = string.Empty;
string User_color = string.Empty;
HttpCookie reqCookies = Request.Cookies["userInfo"];
if (reqCookies != null)
{
    User_name = reqCookies["UserName"].ToString();
    User_color = reqCookies["UserColor"].ToString();
}
```

1. When we make request from client to web server, the web server process the request and give the lot of information with big pockets which will have Header information, Metadata, cookies etc., Then repose object can do all the things with browser.

Cookie's common property:

Domain: This is used to associate cookies to domain.

Secure: We can enable secure cookie to set true (HTTPs).

Value: We can manipulate individual cookie.

Values: We can manipulate cookies with key/value pair.

Expires: Which is used to set expire date for the cookies.

Advantages of Cookie:

- Its clear text so user can able to read it.
- We can store user preference information on the client machine.
- Its easy way to maintain.
- Fast accessing.

Disadvantages of Cookie

- If user clears cookie information we can't get it back.
- No security.
- Each request will have cookie information with page.

7.5 MANAGING SESSION STATE

- ASP.NET maintains a unique id which is called as "session id" for each session. This id is generated using a custom algorithm and it is unique always.
- Session id will be sent to the client as a cookie and the browser resends this upon each request.
- ASP.NET uses this session id to identify the session object.

string sessionId = Session.SessionID

Session State

- Session state is user and browser specific.
- Session state can be stored in memory on the server as well as client's cookies.
- If client has disabled cookies in his browser then session state will be stored in URL.
- Session state has scope to the current browser only. If we change the browser session id is changed.
- The following code shows storing a string value in session.

Session["name"] = "Value";

Values stored in sessions can be removed by several methods.

Session.Abandon() : Cancels the session and fires end event. This is used when you are done with the session.

Session.Clear()/Session.RemoveAll() : Clears all contents of the session.

This will not end the session
Session.Remove(string) : Removes the session name supplied.

7.6 CONFIGURING SESSION STATE

- ASP.NET session state supports several different storage options for session data. Each option is identified by a value in the SessionStateMode enumeration.
- The following list describes the available session state modes:
- **InProc mode**, which stores session state in memory on the Web server. This is the default.
- **StateServer mode**, which stores session state in a separate process called the ASP.NET state service. This ensures that session state is preserved if the Web application is restarted and also makes session state available to multiple Web servers in a Web farm.

  - **SQLServer mode** stores session state in a SQL Server database. This ensures that session state is preserved if the Web application is restarted and also makes session state available to multiple Web servers in a Web farm.

- **Custom mode**, which enables you to specify a custom storage provider.
- **Off mode**, which disables session state.
- You can specify which mode you want ASP.NET session state to use by assigning a SessionStateMode enumeration values to the mode attribute of the sessionState element in your application's Web.config file. Modes other than InProc and Off require additional parameters, such as connection-string values. You can view the currently selected session state by accessing the value of the HttpSessionState.Mode property.

7.7 USING APPLICATION STATE

- If the information that we want to be accessed or stored globally throughout the application, even if multiple users access the site or application at the same time, then we can use an Application Object for such purposes.
- It stores information as a Dictionary Collection in key - value pairs. This value is accessible across the pages of the application / website.

There are 3 events of the Application which are as follows

Application_Start

Application_Error

Application_End

Example - Just for an example, I am setting the Page title in the Application Start event of the **Global.asax file.**

Code for setting value to the Application Object - "PageTitle" is the Key and "Welcome to State Management Application" is the value.

```
<%@ Application Language="C#" %>
<script runat="server">
   void Application_Start(object sender, EventArgs e)
   {
      // Code that runs on application startup
   }
   void Application_End(object sender, EventArgs e)
   {
      //  Code that runs on application shutdown
   }
   void Application_Error(object sender, EventArgs e)
   {
        // Code that runs when an unhandled error occurs
    }
    void Session_Start(object sender, EventArgs e)
    {
        // Code that runs when a new session is started
    }
    void Session_End(object sender, EventArgs e)
    {
        // Code that runs when a session ends.
        // Note: The Session_End event is raised only when the sessionstate
mode is set to InProc in the Web.config file. If session mode is set to
StateServer or SQLServer, the event is not raised.
    }
</script>
```

7.8 USING QUERYSTRING

- Query String is the most simple and efficient way of maintaining information across requests.
- The information we want to maintain will be sent along with the URL. A typical URL with a query string looks like **www.somewebsite.com/search.aspx?query=foo**
- The URL part which comes after the ? Symbol is called a QueryString.
- QueryString has two parts, a key and a value. In the above example, **query** is the key and **foo** is its value.
- We can send multiple values through querystring, separated by the & symbol. The following code shows sending multiple values to the foo.aspx page.

Response.Redirect("foo.aspx?id=1&name=foo");

The following code shows reading the QueryString values in foo.aspx

String id = Request.QueryString["id"];

String name = Request.QueryString["name"];

- The **HtmlEncode()** method is particularly useful if you're retrieving values from a database and you aren't sure if the text is valid HTML.
- We can use the **HtmlDecode()** method to revert the text to its normal form if we need to perform additional operations or comparisons with it in your code.
- The **UrlEncode()** method changes text into a form that can be used in a URL, escaping spaces and other special characters. This step is usually performed with information we want to add to the query string.

Label1.Text = Server.HtmlEncode("To bold text use the <b> tag.");

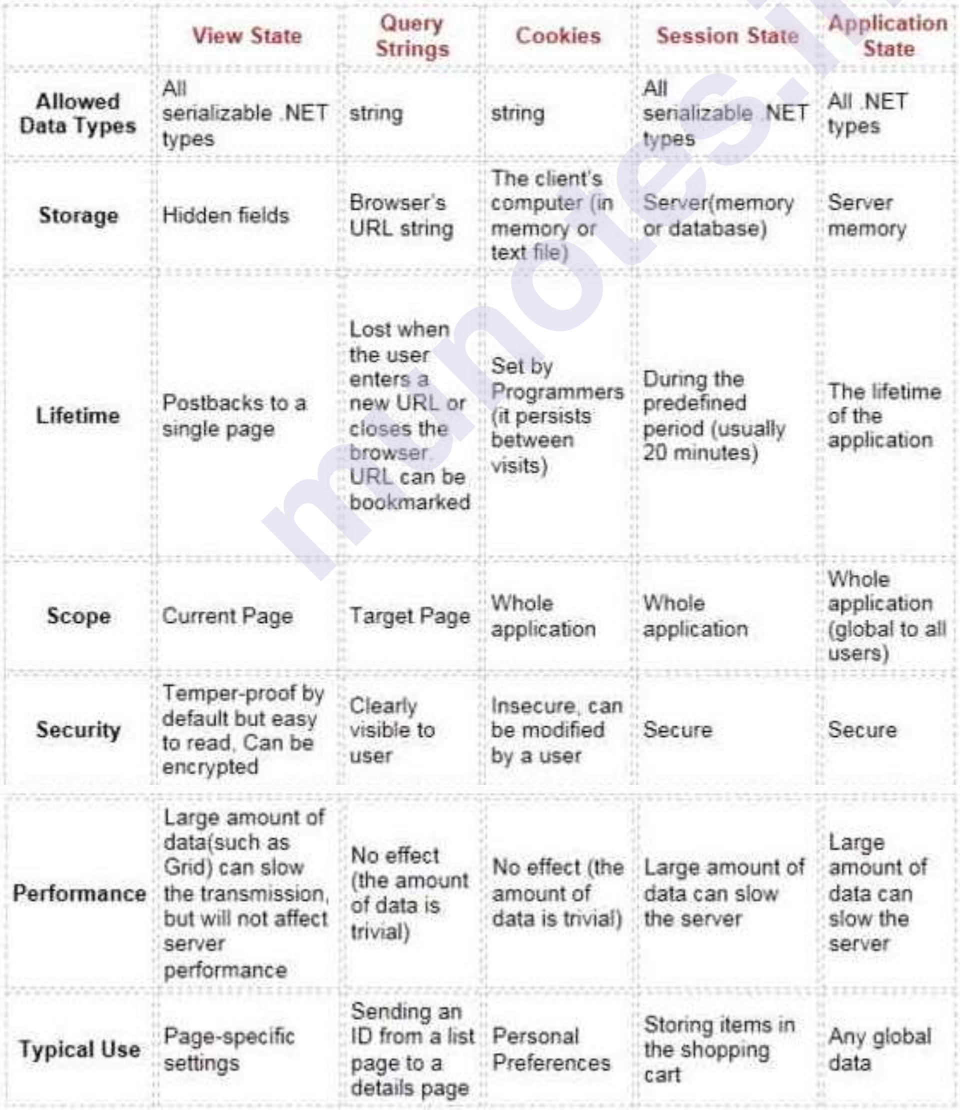
Advantages

- Query string is lightweight and will not consume any server resources.
- It is very easy to use, and it is the most efficient state management technique.

Disadvantages

- We can pass information only as a string.
- URL length has limitations. So, we can't send much information through URL.
- Information passed is clearly visible to everyone and can be easily altered.

7.9 COMPARING STATE MANAGEMENT OPTIONS



	View State	Query Strings	Cookies	Session State	Application State
Allowed Data Types	All serializable .NET types	string	string	All serializable .NET types	All .NET types
Storage	Hidden fields	Browser's URL string	The client's computer (in memory or text file)	Server(memory or database)	Server memory
Lifetime	Postbacks to a single page	Lost when the user enters a new URL or closes the browser. URL can be bookmarked	Set by Programmers (it persists between visits)	During the predefined period (usually 20 minutes)	The lifetime of the application
Scope	Current Page	Target Page	Whole application	Whole application	Whole application (global to all users)
Security	Temper-proof by default but easy to read, Can be encrypted	Clearly visible to user	Insecure, can be modified by a user	Secure	Secure
Performance	Large amount of data(such as Grid) can slow the transmission, but will not affect server performance	No effect (the amount of data is trivial)	No effect (the amount of data is trivial)	Large amount of data can slow the server	Large amount of data can slow the server
Typical Use	Page-specific settings	Sending an ID from a list page to a details page	Personal Preferences	Storing items in the shopping cart	Any global data

7.10 REFERENCE

- **Beginning ASP.NET 4.5 in C# by Mathew MacDonald** – Apress Publication (2012)
- **Murach‟s ASP.NET 4.6 Web Programming in C#2015** by Anne Bohem and Joel Murach – Murach Publication (2016)
- ASP.NET 4.0 Programming by J. Kanjilal - Tata MsGrawhill Publication (2011)

7.11 UNIT END QUESTION

Answer the following questions

1. How to manage states in ASP .Net?
2. What are the different types of State Management techniques?
3. Explain Client Side Management technique with its types.
4. Explain Viewstate with an example.
5. Explain Query string with an example.
6. Explain Cross page posting with an example.
7. Explain cookies with an example.
8. Explain session state with an example.
9. Explain application state with an example.

❄❄❄❄❄❄❄

8

STYLES, THEMES AND MASTER PAGES

Unit Structure

8.0 OBJECTIVE

After going through this unit, you will be able to,

- Create a Web Application with better design.
- Create the Application using different types CSS.
- Know about Master Pages and Themes.
- Create Web Application with Master Pages and Styles.

8.1 STYLES

A style sheet is a file or form that is used in word processing and desktop publishing to define the layout style of a document.

A style sheet contains **the specifications of a document's layout**, such as the page size, margins, fonts and font sizes

8.1.1 What is CSS?

CSS stands for Cascading Style Sheets.

CSS saves a lot of work. It can control the layout of multiple web pages all at once.

Cascading Style Sheets (CSS) is used to format the layout of a webpage.

With CSS, you can control the color, font, size of text, the spacing between elements, how elements are positioned and laid out, what background images or background colors are to be used, different displays for different devices and screen sizes, and much more!

8.1.2 Types of CSS

There are three types of CSS as follows:

- **External CSS**
- **Internal CSS or Embedded CSS**
- **Inline CSS**

External Style Sheet

The first way to add CSS style sheets to your web pages is through the **<link>** element that points to an external CSS file.

For example the following <link> shows what options you have when embedding a style sheet in your page:

```
<link href=”StyleSheet.css” rel=”Stylesheet” type=”text/css”
media=”screen” />
```

The href property points to a file within our site when we create links between two pages. The rel and type attributes tell the browser that the linked file is in fact a cascading style sheet. The media attribute enables us to target different devices, including the screen, printer, and handheld devices. The default for the media attribute is screen.

Embedded style sheet

The second way to include style sheets is using embedded <style> elements. The <style> element should be placed at the top of your ASPX or HTML page, between the <head> tags.

For example, to change the appearance of an <h1> element in the current page alone, we can add the following code to the <head> of our page:

```
<head runat=”server”>
<style type=”text/css”>
h1
{
color: Blue;
}
</style>
</head>
```

Inline style sheet

The third way to apply CSS to your HTML elements is to use inline styles. Because the style attribute is already applied to a specific HTML element, we don’t need a selector and we can write the declaration in the attribute directly:

```
<span style=”color: White; background-color: Black ;”>
This is white text on a black background.
</span>
```

8.1.3 Types of CSS Selectors

1) Universal Selector

The Universal selector, indicated by an asterisk (*), applies to all elements in your page. The Universal selector can be used to set global settings like a font family. The following rule set changes the font for all elements in our page to Arial:

```
* {
  font-family: Arial;
  }
```

2) Type Selector

The Type selector enables us to point to an HTML element of a specific type. With a Type selector all HTML elements of that type will be styled accordingly.

```
h1
{
color: Green;
}
```

This Type selector now applies to all <h1> elements in your code and gives them a green color. Type Selectors are not case sensitive, so you can use both h1 and H1 to refer to the same heading.

3) ID Selector

The ID selector is always prefixed by a hash symbol (#) and enables us to refer to a single element in the page. Within an HTML or ASPX page, we can give an element a unique ID using the id attribute. With the ID selector, we can change the behavior for that single element, for example:

```
#IntroText
{
font-style: italic;
}
```

Because we can reuse this ID across multiple pages in our site (it only must be unique within a single page), you can use this rule to quickly change the appearance of an element that you use once per page, but more than once in our site, for example with the following HTML code:

```
<p id=”IntroText”>I am italic because I have the right ID. </p>
```

4) Class Selector

The Class selector enables us to style multiple HTML elements through the class attribute. This is handy when we want to give the same type of formatting to several unrelated HTML elements. The following rule changes the text to red and bold for all HTML elements that have their class attributes set to highlight:

```
.Highlight
{
font-weight: bold; color: Red;
}
```

The following code snippet uses the Highlight class to make the contents of a <span> element and a link (<a>) appear with a bold typeface:

This is normal text but <span class=”Highlight”>this is Red and Bold.</span>

This is also normal text but <a href=”CssDemo.aspx” class=”Highlight”>this link is Red and Bold as well</a>

The link tag

The HTML <link> tag is used for defining a link to an external document. It is placed in the <head> section of the document.

The <link> tag defines a link between a document and an external resource.

The <link> tag is used to link to external style sheets.

Syntax:

```
<head><link rel="stylesheet" type="text/css" href="theme.css"></head>
```

Example:

```
<html>
<head>
<title>HTML link Tag</title>
<link rel="stylesheet" type="text/css" href="default.css" />
</head>
<body>
<div>
<p>Welcome to our website. We provide tutorials on various subjects.</p>
</div>
</body>
</html>
```

Where,

rel-can be used to specify the relationship of the target of the link to the current page.

type-This attribute Provides information about the content type of the destination resource, telling whether it's an HTML document, a JPG image, an Excel document, etc.

href(uri)-The "href" attribute specifies the destination resource, which the element is linking to. It may specify a resource in the same website or in an external one.

8.2 THEMES

A theme decides the look and feel of the website. It is a collection of files that define the looks of a page. It can include skin files, CSS files & images.

We define themes in a special **App_Themes** folder. Inside this folder is one or more subfolders named Theme1, Theme2 etc. that define the actual themes. The theme property is applied late in the page's life cycle, effectively overriding any customization we may have for individual controls on our page.

There are 3 different options to apply themes to our website:

1. Setting the theme at the page level: The Theme attribute is added to the page directive of the page.

   ```
   <%@PageLanguage="C#"AutoEventWireup="true"CodeFile="Default.aspx.cs "Inherits="Default" Theme="Theme1"%>
   ```

2. Setting the theme at the site level: to set the theme for the entire website we can set the theme in the web.config of the website. Open the web.config file and locate the <pages> element and add the theme attribute to it:

   ```
   <pagestheme="Theme1">
   ....
   ....
   </pages>
   ```

3. Setting the theme programmatically at runtime: here the theme is set at runtime through coding. It should be applied earlier in the page's life cycle ie. Page_PreInit event should be handled for setting the theme. The better option is to apply this to the Base page class of the site as every page in the site inherits from this class.

   ```
   Page.Theme = Theme1;
   ```

Uses of Themes

1. Since themes can contain CSS files, images and skins, you can change colors, fonts, positioning and images simply by applying the desired themes.

2. We can have as many themes as we want, and we can switch between them by setting a single attribute in the web.config file or an individual aspx page. Also, we can switch between themes programmatically.
3. Setting the themes programmatically, we are offering our users a quick and easy way to change the page to their likings.
4. Themes allow us to improve the usability of our site by giving users with vision problems the option to select a high contrast theme with a large font size.

1. **"Start" - "All Programs" - "Microsoft Visual Studio 2010"**
2. **"File" - "New Website" - "C# - Empty website" (to avoid adding a master page)**
3. **Provide the web site a name, such as Using Skins or whatever you wish and specify the location**
4. **Then right-click on the solution in the Solution Explorer then select "Add New Item" - "Default.aspx page" .**

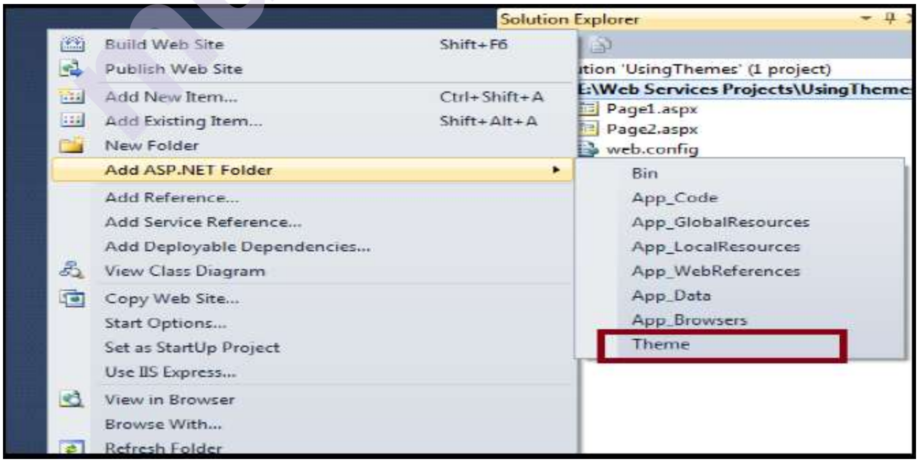
Skin file

ASP.Net skins can only be used to apply the styles to the ASP.Net controls. So in this article let us see the procedure for using ASP.Net Skins.

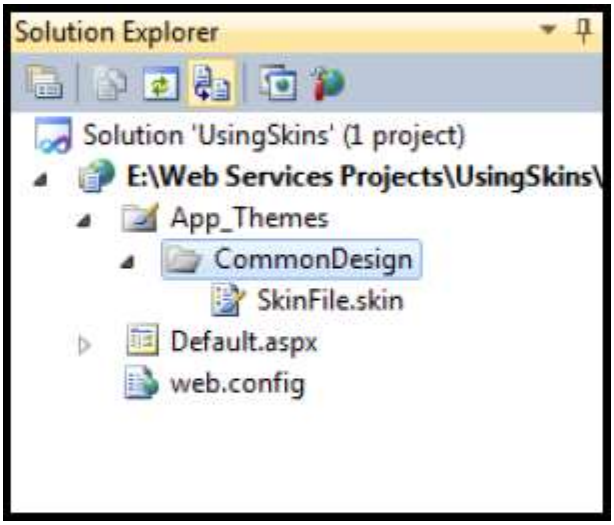
First create the web application as in the following:

Add an ASP.Net Themes Folder

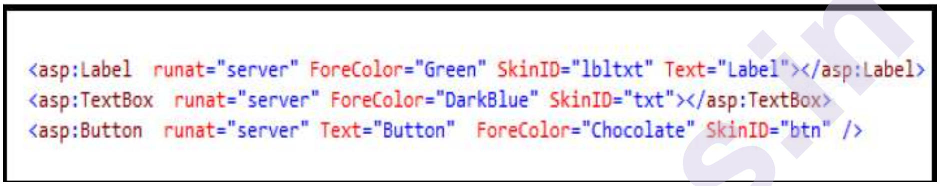
To use the themes in the web site, we need to add an ASP.Net Themes folder by right-clicking on Solution Explorer as in the following:



After adding the theme folder, add the SkinFile.skin file by right-clicking on the ASP.Net theme folder. The Solution Explorer will then look as follows:



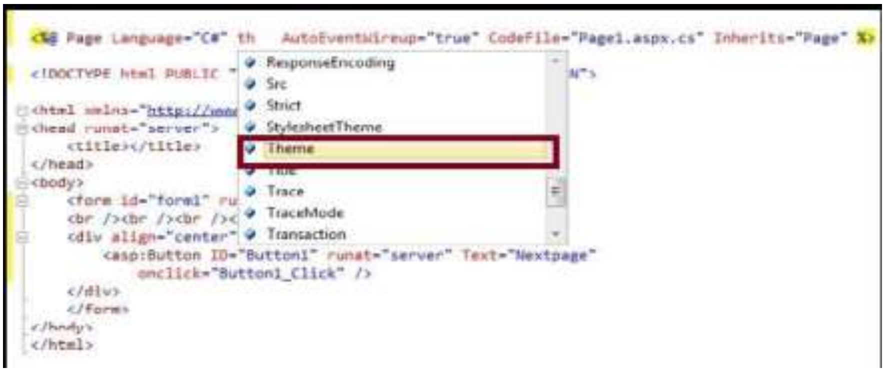
Now add the ASP.Net controls inside the SkinFile. Skin and assign the Style to the controls using their properties as in the following:



```
<asp:Label  runat="server" ForeColor="Green" SkinID="lbltxt" Text="Label"></asp:Label>
<asp:TextBox  runat="server" ForeColor="DarkBlue" SkinID="txt"></asp:TextBox>
<asp:Button  runat="server" Text="Button"  ForeColor="Chocolate" SkinID="btn" />
```

1. A control Id cannot be assigned to ASP.Net controls inside the SkinFile.skin.
2. SkinId must be assigned to the ASP.Net controls inside the SkinFile.skin.
3. The SkinId should be uniquely defined because duplicate SkinId's per control type are not allowed in the same theme.
4. Only one default control skin per control type is allowed in the same theme.

To use existing ASP.Net Skins in an ASP.Net page we need to assign the existing theme at page level as in the following.

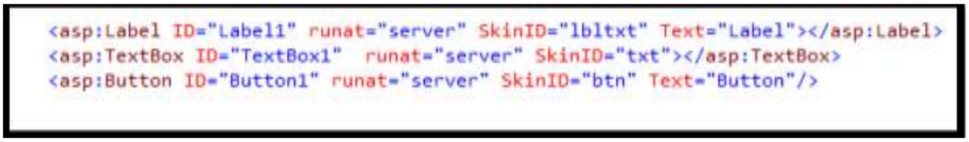


```
<%@ Page Language="C#" th   AutoEventWireup="true" CodeFile="Page1.aspx.cs" Inherits="Page" %>

<!DOCTYPE html PUBLIC "                                        N">

<html xmlns="http://www
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" ru
    <br /><br /><br /><
    <div align="center"
        <asp:Button ID="Button1" runat="server" Text="Nextpage"
            onclick="Button1_Click" />
    </div>
    </form>
</body>
</html>
```

In the preceding source code, we are assigning the existing ASP.Net Skin File at page level, the existing ASP.Net Skin automatically appears in the box after using the Themes property in the page header.

Assigning the Skin to the ASP.Net Controls

To assign the skin to the ASP.Net controls, you need to assign it to the control's SkinId Property as in the following:

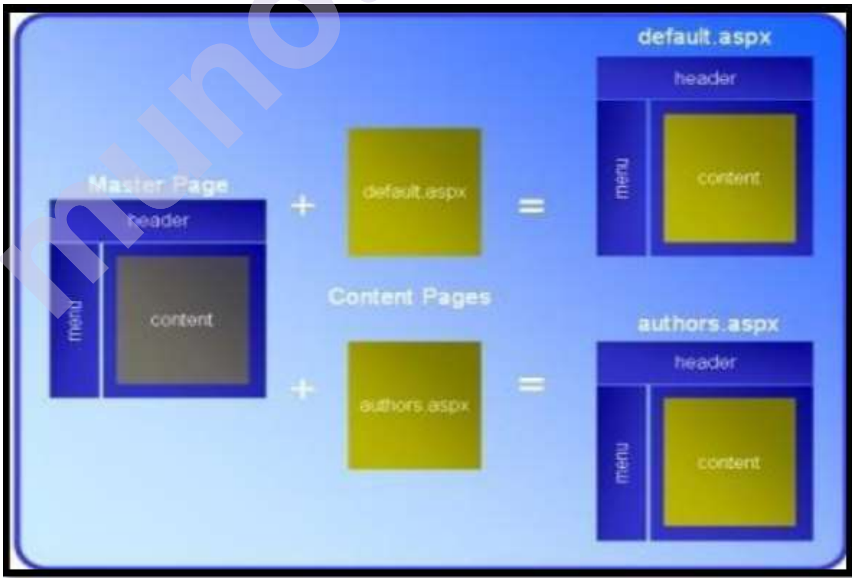


```
<asp:Label ID="Label1" runat="server" SkinID="lbltxt" Text="Label"></asp:Label>
<asp:TextBox ID="TextBox1"  runat="server" SkinID="txt"></asp:TextBox>
<asp:Button ID="Button1" runat="server" SkinID="btn" Text="Button"/>
```

8.3 MASTER PAGE

ASP.NET master pages allow us to create a consistent layout for the pages in our application.

- A single master page defines the look and feel and standard behavior that we want for all of the pages (or a group of pages) in our application.
- We can then create individual content pages that contain the content we want to display.
- When users request the content pages, they merge with the master page to produce output that combines the layout of the master page with the content from the content page.
- Master pages actually consist of two pieces, the master page itself and one or more content pages.



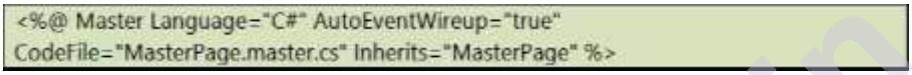
- A centralized way to change the above created set of controls which will effectively change all the web pages.
- To some extent, a master page looks like a normal ASPX page.
- It contains static HTML such as the <html>, <head>, and <body> elements, and it can also contain other HTML and ASP.NET server controls.

- Inside the master page, you set up the markup that you want to repeat on every page, like the general structure of the page and the menu.
- However, a master page is not a true ASPX page and cannot be requested in the browser directly it only serves as the template that real web pages called content pages
- One difference is that while web forms start with the Page directive, a master page starts with a Master directive that specifies the same information.

Use of Master Pages

The master pages can be used to accomplish the following:

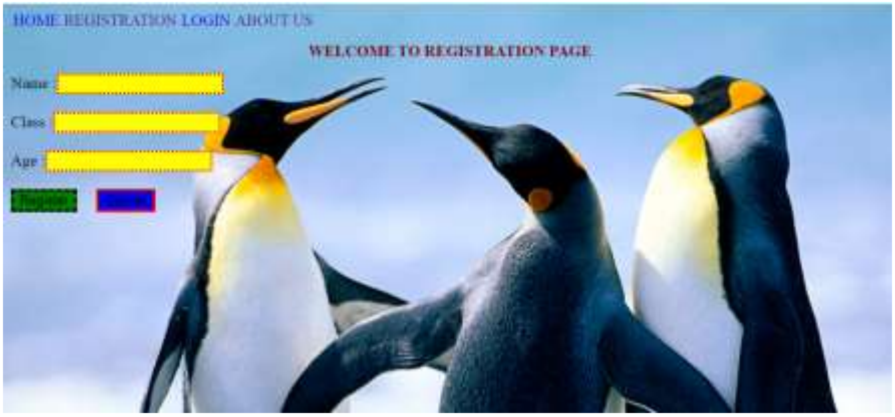
Creating a set of controls that are common across all the web pages and attaching them to all the web pages.



```
<%@ Master Language="C#" AutoEventWireup="true"
CodeFile="MasterPage.master.cs" Inherits="MasterPage" %>
```

Content page and Master page

- Master page provides a framework (common content as well as the layout) within which the content from other pages can be displayed.
- It provides elements such as headers, footers, style definitions, or navigation bars that are common to all pages in your web site.
- So the Content Pages need not have to duplicate code for shared elements within your Web site.
- It gives a consistent look and feel for all pages in your application.
- The master page layout consists of regions where the content from each content page should be displayed.
- These regions can be set using ContentPlaceHolder server controls.
- These are the regions where you are going to have dynamic content in your page
- A derived page also known as a content page is simply a collection of blocks the runtime will use to fill the regions in the master.
- To provide content for a ContentPlaceHolder, you use another specialized control, called Content.
- The ContentPlaceHolder control and the Content control have a one-to-one relationship.
- For each ContentPlaceHolder in the master page, the content page supplies a matching Content control
- ASP.NET links the Content control to the appropriate ContentPlaceHolder by matching the ID of the ContentPlaceHolder with the Content ContentPlaceHolderID property of the corresponding Content control.


HOME REGISTRATION LOGIN ABOUT US
WELCOME TO REGISTRATION PAGE
Name
Class
Age

8.4 REFERENCE

- **Beginning ASP.NET 4.5 in C# by Mathew MacDonald** – Apress Publication (2012)
- **Murach‟s ASP.NET 4.6 Web Programming in C#2015** by Anne Bohem and Joel Murach – Murach Publication (2016)
- ASP.NET 4.0 Programming by J. Kanjilal - Tata MsGrawhill Publication (2011)

8.5 UNIT END QUESTIONS

1. What is CSS? Explain with its advantages and disadvantages.
2. What are different types of CSS?
3. Explain different CSS selectors.
4. Explain Theme and Global Theme.
5. How to create and add skin files in a web form?
1. Give the uses of master page. How master pages work?

❄❄❄❄❄❄❄

9

ADO.NET FUNDAMENTALS

Unit Structure

9.1 OBJECTIVES:

This chapter would make you understand the following concepts:

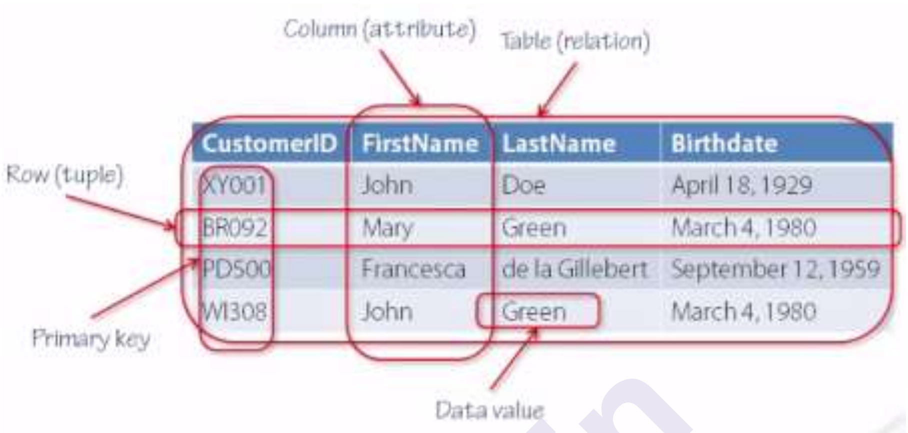
- Concept of database and its configuration.
- Basic of SQL
- Data Provider model
- Direct and indirect way of accessing the data

9.2 UNDERSTANDING THE DATABASE:

- A database is a collection of information which is structured for storing, managing and retrieving information.
- In relational database approach, data is stored in different tables instead of placing all data in one large table.
- A table is a collection of related data entities and it consists of columns and rows.
- It allows defining relationships between different tables.

- The relationship allows merging of data from several tables for querying and reporting.
- This is accomplished by the use of keys which are used to uniquely identify specific records in a table.

9.2.1 Relational table design:



CustomerID	FirstName	LastName	Birthdate
XY001	John	Doe	April 18, 1929
BR092	Mary	Green	March 4, 1980
PD500	Francesca	de la Gillebert	September 12, 1959
WI308	John	Green	March 4, 1980

- Primary key: Each table must have a column which uniquely identifies every record of the table which is called as Primary Key. For example, CustomerID.
- Foreign key: The column in a table which is used to reference a Primary key in another table is called as Foreign key.

9.2.2 Types of Relationships among tables:

<u>One to One :</u>

- Each record in Table A relates to one, and only one, record in Table B, and each record in Table B relates to one, and only one, record in Table A.
- Look at the following example of tables from a company's Employees database:

PERSONAL						
Employee ID	**First Name**	**Last Name**	**Address**	**City**	**State**	**Zip**
EN1-10	Carol	Schaaf	2306 Palisade Ave.	Union City	NJ	07087
EN1-12	Gayle	Murray	1855 Broadway	New York	NY	12390
EN1-15	Steve	Baranco	742 Forrest St.	Kearny	NJ	07032
EN1-16	Kristine	Racich	416 Bloomfield St.	Hoboken	NJ	07030
EN1-19	Barbara	Zumbo	24 Central Ave.	Ritchfield Park	NJ	07660
EN1-20	Daniel	Gordon	2 Angelique St.	Weehawken	NJ	07087

PAYROLL	
EmployeeID	**PayRate**
EN1-10	$25.00
EN1-12	$27.50
EN1-15	$20.00
EN1-16	$19.00
EN1-19	$22.75
EN1-20	$23.00

From above, each record in the Personal table is about one employee. That record relates to one, and only one, record in the Payroll table.

One to Many :

- It means a record in Table A can relate to zero, one, or many records in Table B. Many records in Table B can relate to one record in Table A.
- Look at the following tables about a company's Customers and Orders.

CUSTOMERS					
Customer ID	**Customer Name**	**Address**	**City**	**State**	**Zip**
20151	Engel's Books	19 International Dr	Ryebrook	NY	10273-9764
20493	Jamison Books	396 Apache Ave	Fountain Valley	CA	92708-4982
20512	Gardening Galore	79 Gessner Pk	Houston	TX	77024-6261
20688	Books Abound	51 Ulster St	Denver	CO	80237-3386

ORDERS				
Order Num	**Customer ID**	**Order Date**	**Ship Date**	**Shipper**
76654	20151	2/1/00	2/6/00	USPS
74432	20151	6/30/99	7/2/99	Federal Express
75987	20151	11/10/99	11/12/99	UPS
62922	20493	9/5/99	9/6/99	UPS
65745	20493	10/1/99	10/3/99	USPS

From above, the Customers table holds a unique record for each customer. Each customer can (and, we hope, does) place many orders. Many records in the Orders table can relate to only one record in the Customers table.

Many to many :

- It means a record in Table A can relate many records in Table B and records in Table B can relate to many record in Table A.
- Look at the following tables about Employees and Projects.

EMPLOYEES			
EmployeeID	**Last Name**	**First Name**	**ProjectNum**
EN1-26	O'Brien	Sean	30-452-T3
EN1-26	O'Brien	Sean	30-457-T3
EN1-26	O'Brien	Sean	31-124-T3
EN1-33	Guya	Amy	30-452-T3
EN1-33	Guya	Amy	30-482-TC
EN1-33	Guya	Amy	31-124-T3

PROJECTS		
ProjectNum	**ProjectTitle**	**EmployeeID**
30-452-T3	Woodworking Around The House	EN1-26
30-452-T3	Woodworking Around The House	EN1-33
30-452-T3	Woodworking Around The House	EN1-35
30-457-T3	Basic Home Electronics	EN1-26
30-482-TC	The Complete American Auto Repair Guide	EN1-33
31-124-T3	The Sport Of Hang Gliding	EN1-26

Above, tables with a many-to-many relationship.

9.3 INTRODUCTION TO ADO.NET:

- ADO (ActiveX Data Object) is the database access technologies including components for retrieving data, storing data in memory and binding data to controls.
- It is an Object Oriented set of libraries that allows us to interact with data source.
- The data source can be a database, text file, Excel spreadsheet or an XML file.
- We can use the ADO.NET libraries with several types of database systems like Microsoft SQL Server, Microsoft Access, Oracle etc.

9.3.1 ADO.NET Architecture:

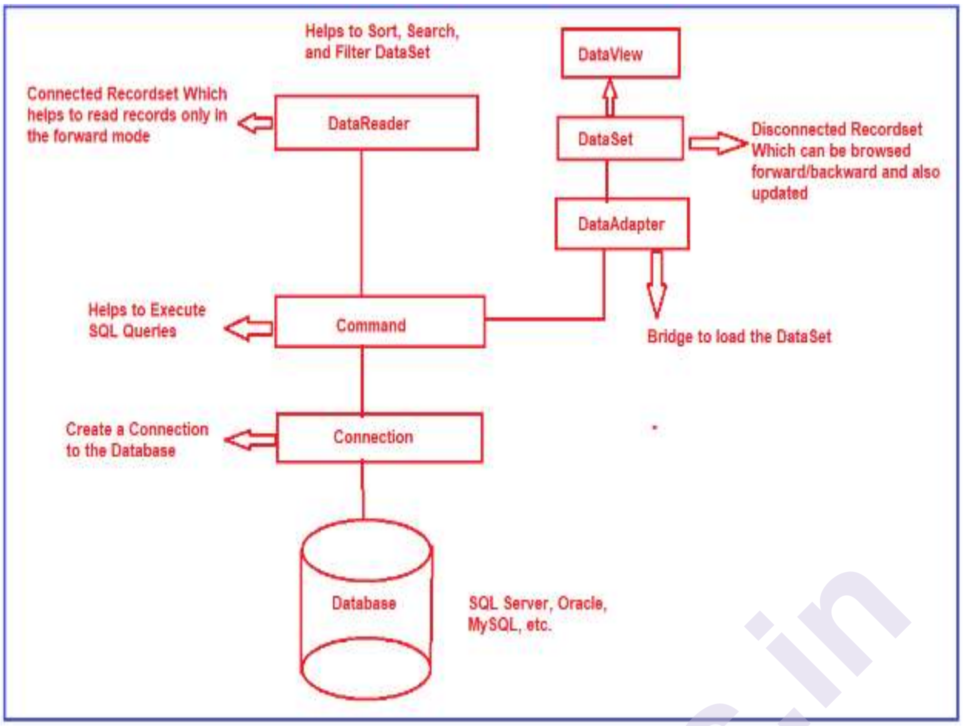

Helps to Sort, Search, and Filter DataSet
DataView
Connected Recordset Which helps to read records only in the forward mode
DataReader
DataSet
Disconnected Recordset Which can be browsed forward/backward and also updated
DataAdapter
Helps to Execute SQL Queries
Command
Bridge to load the DataSet
Create a Connection to the Database
Connection
Database
SQL Server, Oracle, MySQL, etc.

Fig. ADO.NET Architecture

Let us understand each of the components in detail.

Connection:

The first important component is the connection object. The connection object is required to connect with your backend database which can be SQL Server, Oracle, MySQL, etc. To create a connection object, you need at least two things. The first one is where is your database located i.e. the Machine name or IP Address or someplace where your database is located. And the second thing is the security credentials i.e. whether it is a windows authentication or user name and password-based authentication. So, the first is to create the connection object and the connection is required to connect to the backend data source.

Command:

The second important component is the command object. When we talk about databases like SQL Server, Oracle, MySQL, then understand SQL. The command object is the component where you go and write your SQL queries. Later you take the command object and execute it over the connection. Then you can fetch data or send data to the database using the command object and SQL queries.

Data Reader:

Data Reader is a read-only connected record set that helps us to read the records only in the forward mode. Here, you need to understand three things i.e. read-only, connected, and forward mode.

DataSet:

It is a disconnected recordset that can be browsed in both i.e. forward and backward. It is also possible to update via dataset. DataSet gets filled by somebody called DataAdapter.

DataAdapter:

The DataAdapter acts as a bridge between the command object and the dataset. What the DataAdapter does, it takes the data from the command object and fills the data set.

DataView:

A DataView enables you to create different views of the data stored in a DataTable, a capability that is often used in data-binding applications. Using a DataView, you can expose the data in a table with different sort orders, and you can filter the data by row state or based on a filter expression.

9.4 UNDERSTANDING DATA PROVIDER MODEL:

- ADO.NET provides common way to interact with data sources, but comes in different sets of libraries for each we can talk to a data source.
- These libraries are called Data Providers and usually named for the protocol or data source type they allow us to interact with.
- Following table shows some well known data providers, the API prefix they use.

Provider Name	API Prefix	Data Source Description
ODBC Data Provider	Odbc	Data Sources with an ODBC interface. Normally older data bases.
OleDb Data Provider	OleDb	Data Sources that expose an OleDb interface i.e. Access or Excel.
Oracle Data Provider	Oracle	For Oracle Databases.
SQL Data Provider	Sql	For interacting with Microsoft SQL Server

9.5 UNDERSTANDING SQL BASICS:

We are going to study the connection of ASP.NET website to SQL Server in this unit.

a. Create Command:

CREATE DATABASE database-name;

The CREATE DATABASE command is used is to create a new SQL database.

The following SQL creates a database called "testDB":

```
CREATE DATABASE testDB;
```

CREATE TABLE

The CREATE TABLE command creates a new table in the database.

The following SQL creates a table called "Persons" that contains five columns: PersonID, LastName, FirstName, Address, and City:

```
CREATE TABLE Persons (
PersonID int,
LastName varchar(255),
FirstName varchar(255),
Address varchar(255),
City varchar(255)
);
```

CREATE VIEW

The CREATE VIEW command creates a view. A view is a virtual table based on the result set of an SQL statement.

The following SQL creates a view that selects all customers from Brazil:

```
CREATE VIEW [Brazil Customers] AS
SELECT CustomerName, ContactName
FROM Customers
WHERE Country = "Brazil";
```

b. Inserting the data:

INSERT INTO

The INSERT INTO command is used to insert new rows in a table.

The following SQL inserts a new record in the "Customers" table:

```
INSERT INTO Customers (CustomerName, ContactName, Address, City,
PostalCode, Country) VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen
21', 'Stavanger', '4006', 'Norway');
```

c. Selecting the data:

The SELECT command is used to select data from a database. The data returned is stored in a result table, called the result set.

The following SQL statement selects the "CustomerName" and "City" columns from the "Customers" table:

```
SELECT CustomerName, City FROM Customers;
```

The following SQL statement selects all the columns from the Customers" table:

```
SELECT * FROM Customers;
```

d. Updating the data:

UPDATE

The UPDATE command is used to update existing rows in a table.

The following SQL statement updates the first customer (CustomerID = 1) with a new contact person *and* a new city.

```
UPDATE Customers
SET ContactName = 'Alfred Schmidt', City= 'Frankfurt'
WHERE CustomerID = 1;
```

e. Deleting the data:

DELETE

The DELETE command is used to delete existing records in a table.

The following SQL statement deletes the customer "Alfreds Futterkiste" from the "Customers" table:

```
DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste';
```

f. Altering the table:

ALTER TABLE

The ALTER TABLE command adds, deletes, or modifies columns in a table.

The ALTER TABLE command also adds and deletes various constraints in a table.

The following SQL adds an "Email" column to the "Customers" table:

```
ALTER TABLE Customers
ADD Email varchar(255);
```

The following SQL deletes the "Email" column from the "Customers" table:

```
ALTER TABLE Customers
DROP COLUMN Email;
```

g. Drop:

The DROP COLUMN command is used to delete a column in an existing table.

The following SQL deletes the "ContactName" column from the "Customers" table:

```
ALTER TABLE Customers
DROP COLUMN ContactName;
```

The DROP TABLE command deletes a table in the database.

The following SQL deletes the table "Shippers":

```
DROP TABLE Shippers;
```

The DROP DATABASE command is used to delete an existing SQL database.

The following SQL drops a database named "testDB":

```
DROP DATABASE testDB;
```

9.5.1 SQL Server Data Types :

String Data Types

Data type	Description
char(n)	Fixed width character string
varchar(n)	Variable width character string
varchar(max)	Variable width character string
text	Variable width character string
nchar	Fixed width Unicode string
nvarchar	Variable width Unicode string
nvarchar(max)	Variable width Unicode string
ntext	Variable width Unicode string
binary(n)	Fixed width binary string
varbinary	Variable width binary string
varbinary(max)	Variable width binary string
image	Variable width binary string

Numeric Data Types

Data type	Description
bit	Integer that can be 0, 1, or NULL
tinyint	Allows whole numbers from 0 to 255
smallint	Allows whole numbers between -32,768 and 32,767
int	Allows whole numbers between -2,147,483,648 and 2,147,483,647
bigint	Allows whole numbers between -9,223,372,036,854,775,808 and 9,223,372,036,854,775,807
decimal(p,s)	Fixed precision and scale numbers. Allows numbers from -10^38 +1 to 10^38 –1. The p parameter indicates the maximum total number of digits that can be stored (both to the left and to the right of the decimal point). p must be a value from 1 to 38. Default is 18. The s parameter indicates the maximum number of digits stored to the right of the decimal point. s must be a value from 0 to p. Default value is 0
numeric(p,s)	Fixed precision and scale numbers. Allows numbers from -10^38 +1 to 10^38 –1. The p parameter indicates the maximum total number of digits that can be stored (both to the left and to the right of the decimal point). p must be a value from 1 to 38. Default is 18. The s parameter indicates the maximum number of digits stored to the right of the decimal point. s must be a value from 0 to p. Default value is 0
smallmoney	Monetary data from -214,748.3648 to 214,748.3647
money	Monetary data from -922,337,203,685,477.5808 to 922,337,203,685,477.5807
float(n)	Floating precision number data from -1.79E + 308 to 1.79E + 308. The n parameter indicates whether the field should hold 4 or 8 bytes. float(24) holds a 4-byte field and float(53) holds an 8-byte field. Default value of n is 53.
real	Floating precision number data from -3.40E + 38 to 3.40E + 38

Date and Time Data Types

Data type	Description
datetime	From January 1, 1753 to December 31, 9999 with an accuracy of 3.33 milliseconds
datetime2	From January 1, 0001 to December 31, 9999 with an accuracy of 100 nanoseconds
smalldatetime	From January 1, 1900 to June 6, 2079 with an accuracy of 1 minute
date	Store a date only. From January 1, 0001 to December 31, 9999
time	Store a time only to an accuracy of 100 nanoseconds
datetimeoffset	The same as datetime2 with the addition of a time zone offset
timestamp	Stores a unique number that gets updated every time a row gets created or modified. The timestamp value is based upon an internal clock and does not correspond to real time. Each table may have only one timestamp variable

9.6 CONFIGURING DATABASE

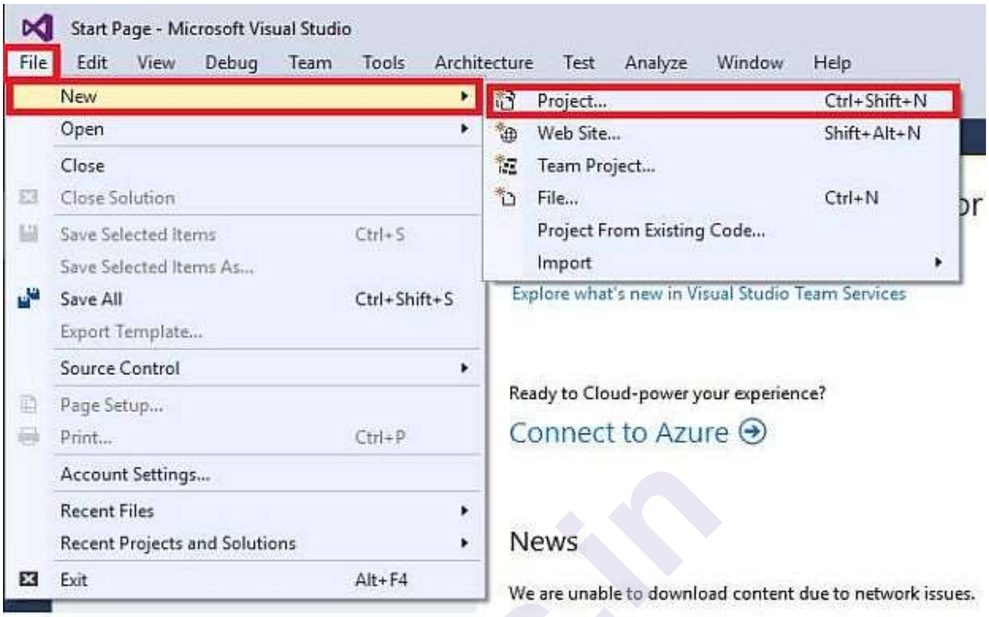
Requirements

- Visual Studio 2015 Update 3
- ASP.NET 4.5.2
- SQL Server

If you want to connect to the SQL database into ASP.NET, using C#, it should follow the steps given below.

Step 1

Now, Open Visual Studio 2015 Update 3, go to the File >> New >> Project or use the shortcut key "Ctrl+Shift +N".

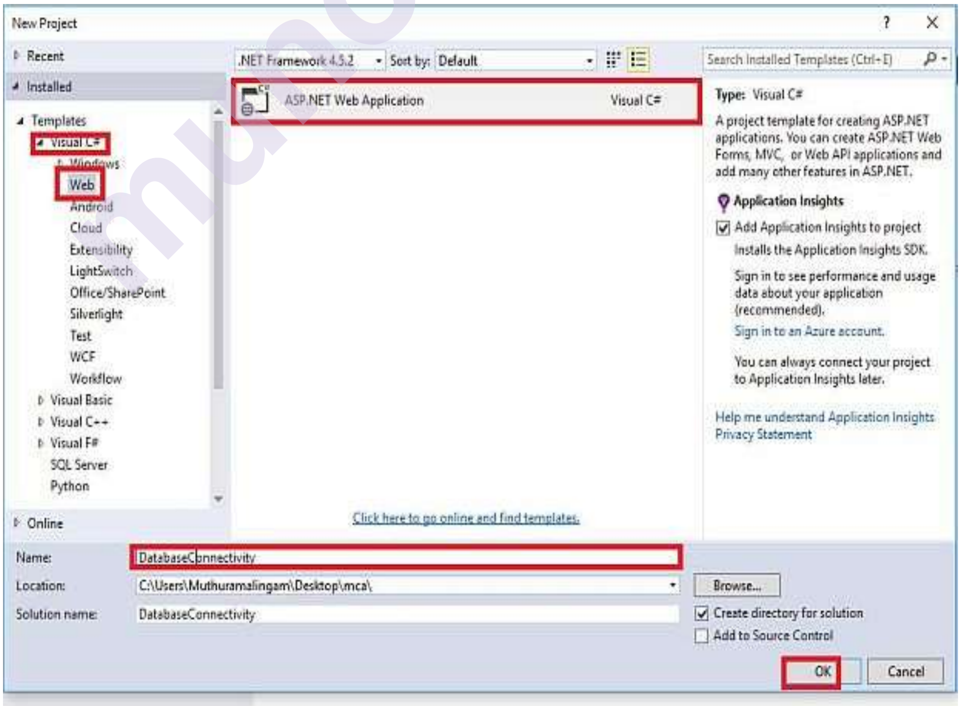


Ready to Cloud-power your experience?

We are unable to download content due to network issues.

Step 2

Here, select Visual C# >> Web >> ASP.NET Web Application. Finally, click "OK" button.



Type: Visual C#

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

Application Insights

☑ Add Application Insights to project

Installs the Application Insights SDK.

Sign in to see performance and usage data about your application (recommended).

Sign in to an Azure account.

You can always connect your project to Application Insights later.

Help me understand Application Insights

Privacy Statement

Name: DatabaseConnectivity

Location: C:\Users\Muthuramalingam\Desktop\mca\

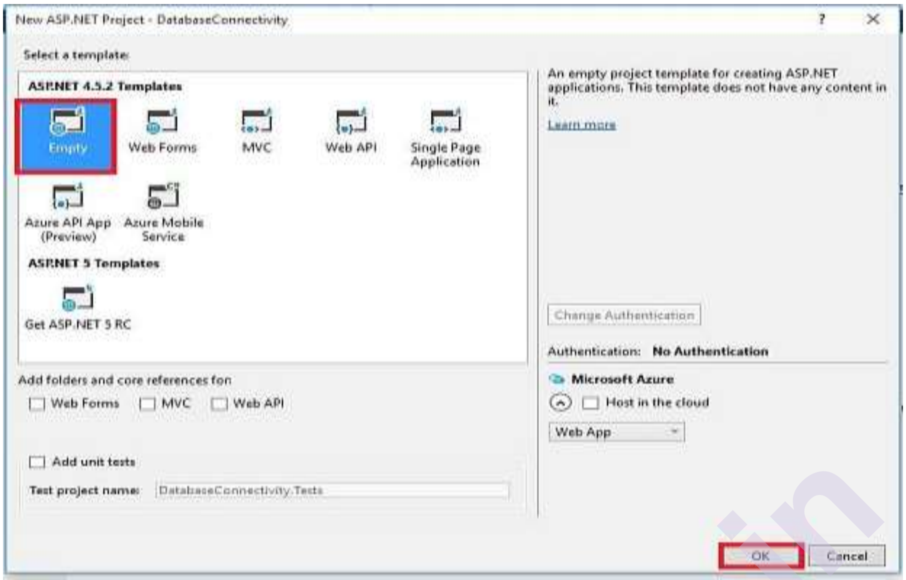
Solution name: DatabaseConnectivity

☑ Create directory for solution

□ Add to Source Control

Step 3

Here, you can select the template for your ASP.NET Application. We are choosing "Empty" here. Now, click OK button.



Select a template:

ASP.NET 4.5.2 Templates

ASP.NET 5 Templates

An empty project template for creating ASP.NET applications. This template does not have any content in it.

Authentication: **No Authentication**

Microsoft Azure

□ Host in the cloud

Web App

Add folders and core references for:

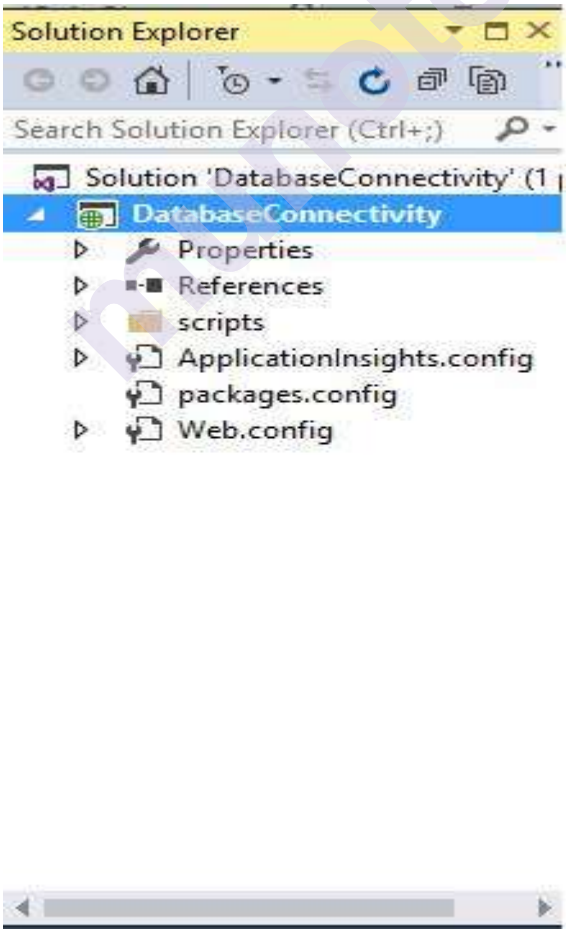
□ Web Forms □ MVC □ Web API

□ Add unit tests

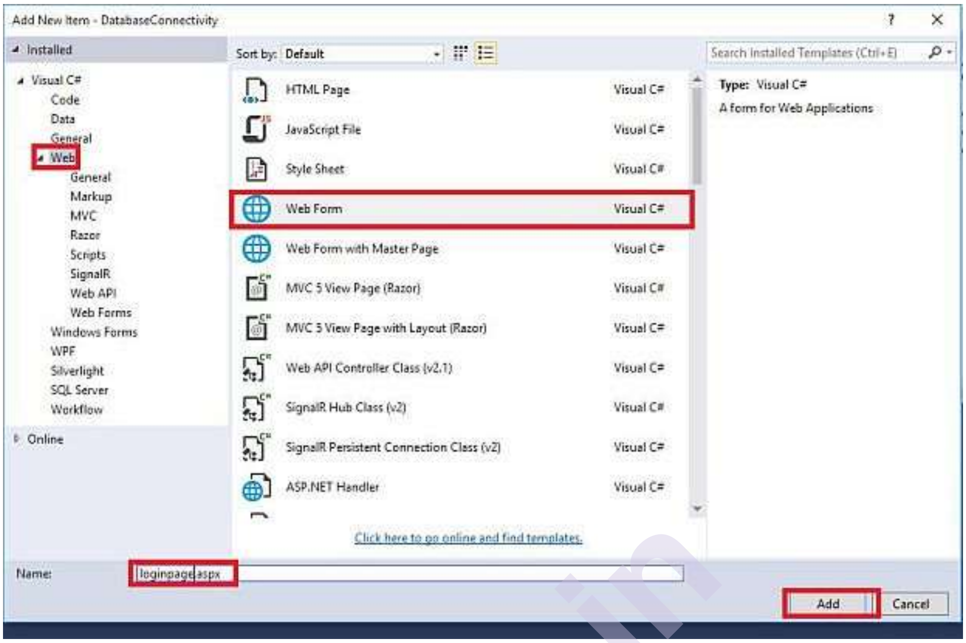
Test project name: DatabaseConnectivity.Tests

Step 4

Now, open the project and look for the Solution Explorer.

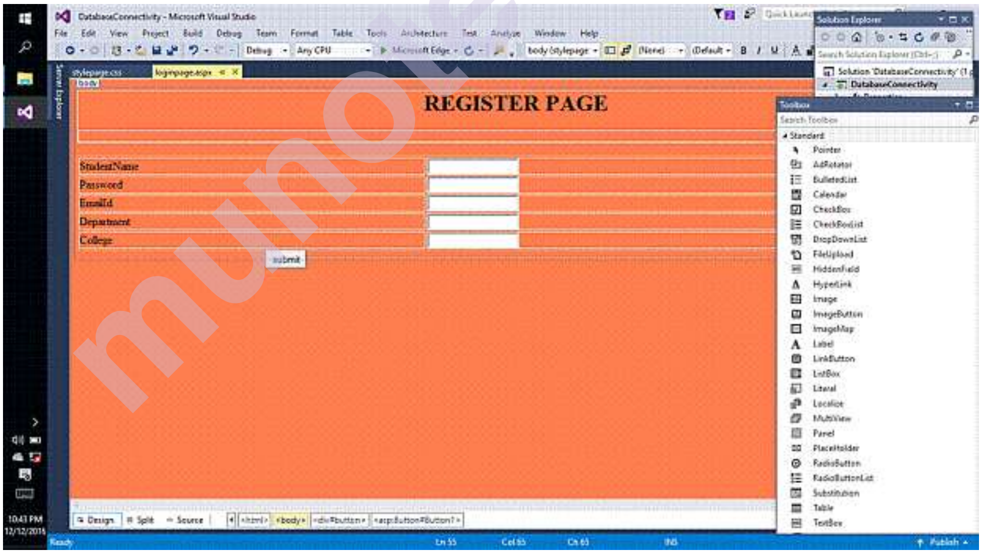

Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'DatabaseConnectivity' (1
DatabaseConnectivity
Properties
References
scripts
ApplicationInsights.config
packages.config
Web.config

Here, open the default .aspx. If you want a Webform, you can add the page (Web Form). Add+New item (Ctrl+Shift+A).


Add New Item - DatabaseConnectivity
Installed
Visual C#
Code
Data
General
Web
General
Markup
MVC
Razor
Scripts
SignalR
Web API
Web Forms
Windows Forms
WPF
Silverlight
SQL Server
Workflow
Online
Sort by: Default
Search Installed Templates (Ctrl+E)
HTML Page Visual C#
JavaScript File Visual C#
Style Sheet Visual C#
Web Form Visual C#
Web Form with Master Page Visual C#
MVC 5 View Page (Razor) Visual C#
MVC 5 View Page with Layout (Razor) Visual C#
Web API Controller Class (v2.1) Visual C#
SignalR Hub Class (v2) Visual C#
SignalR Persistent Connection Class (v2) Visual C#
ASP.NET Handler Visual C#
Type: Visual C#
A form for Web Applications
Click here to go online and find templates.
Add
Cancel

Name: loginpage.aspx

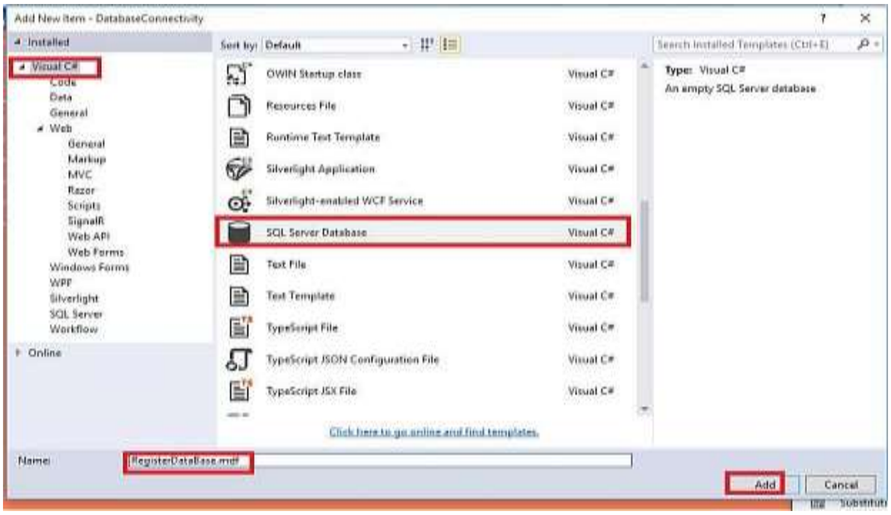
Now, we can create a login page, using ASP.NET code. You need to follow the drag and drop method. Here, we already created the login page.



Step 5

Now, open the project. If you want a SQL Server database, you can add the page (SQL Server database). Add+New item (Ctrl+Shift+A).

Here, you can select Visual C# and choose SQL Server database.
Afterwards, click "OK" button.

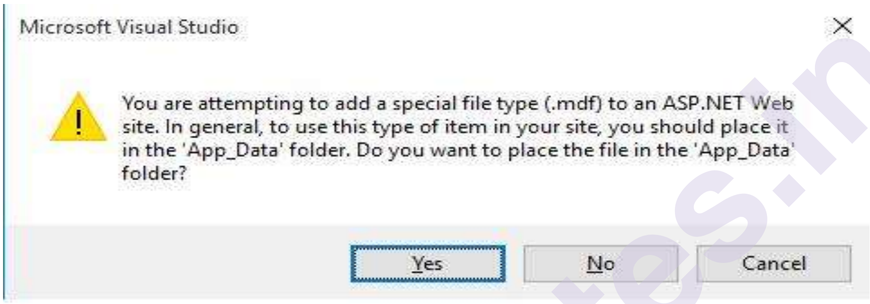

Add New Item - DatabaseConnectivity
Installed
Visual C#
Code
Data
General
Web
General
Markup
MVC
Razor
Scripts
SignalR
Web API
Web Forms
Windows Forms
WPF
Silverlight
SQL Server
Workflow
Online
Sort by: Default
OWIN Startup class Visual C#
Resources File Visual C#
Runtime Text Template Visual C#
Silverlight Application Visual C#
Silverlight-enabled WCF Service Visual C#
SQL Server Database Visual C#
Text File Visual C#
Text Template Visual C#
TypeScript File Visual C#
TypeScript JSON Configuration File Visual C#
TypeScript JSX File Visual C#
Click here to go online and find templates.
Search Installed Templates (Ctrl+E)
Add
Cancel

Type: Visual C#

An empty SQL Server database

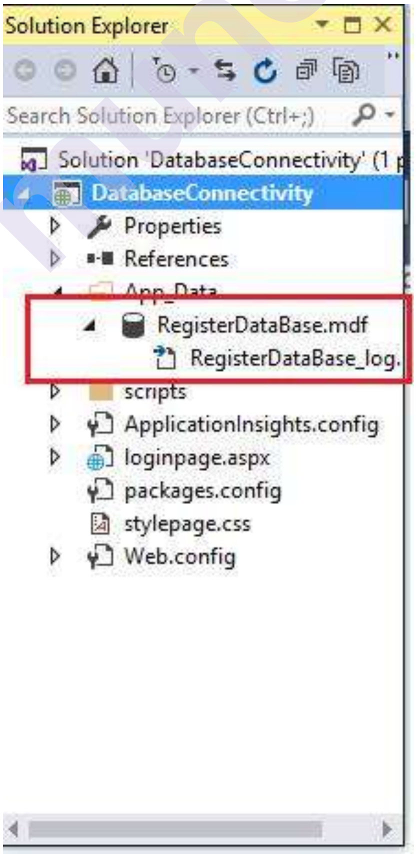
Name: RegisterDataBase.mdf

Here, open the new Window and click YES button.



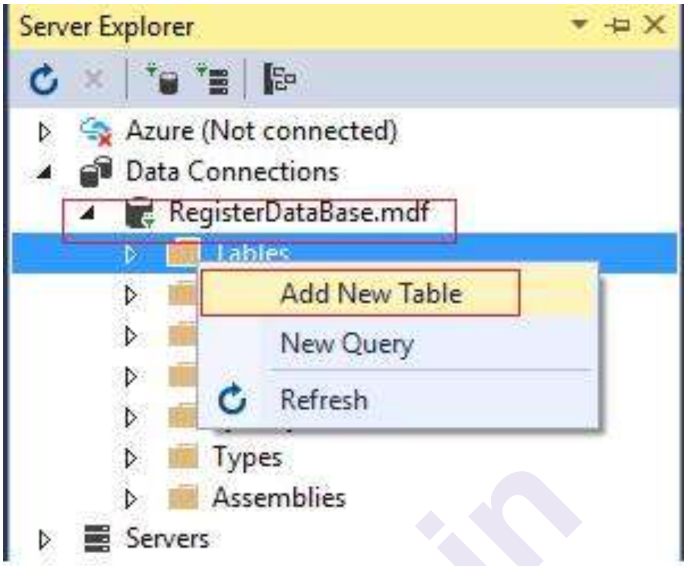
You are attempting to add a special file type (.mdf) to an ASP.NET Web site. In general, to use this type of item in your site, you should place it in the 'App_Data' folder. Do you want to place the file in the 'App_Data' folder?

Now, add this to the database in our project.

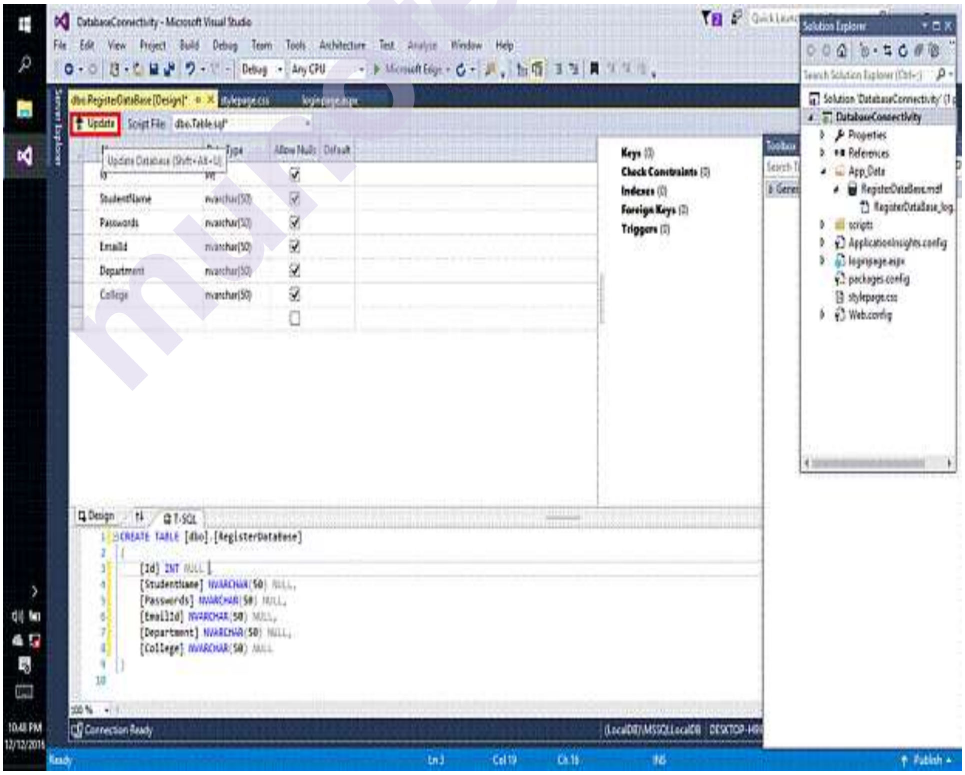


Step 6

Now, we can go to the Server Explorer and add your database. You can click the Tables and afterwards, click Add New Table.



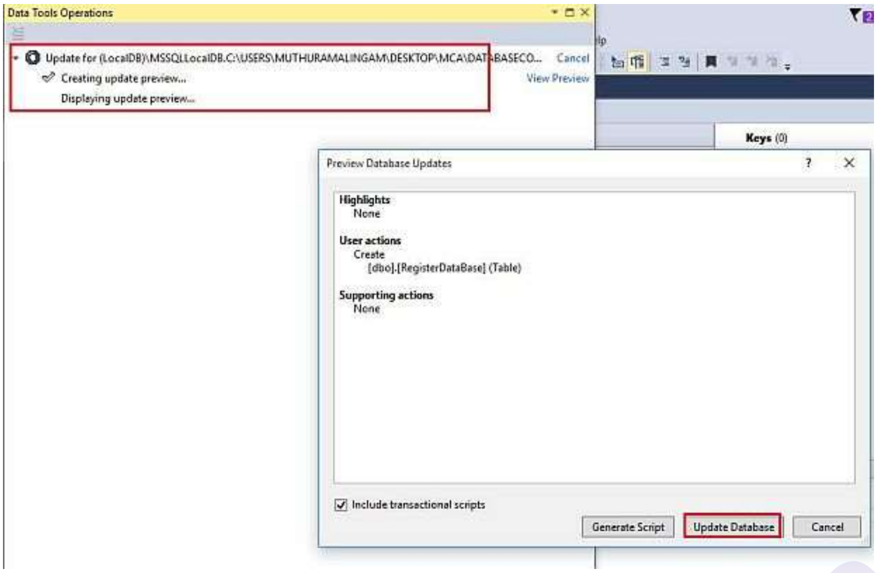
Now, open the new table and you can fill the data, which is like (studentname, password) and afterwards, you click the Update.



Name	Data Type	Allow Nulls	Default
Id	int	☑	
StudentName	nvarchar(50)	☑	
Passwords	nvarchar(50)	☑	
EmailId	nvarchar(50)	☑	
Department	nvarchar(50)	☑	
College	nvarchar(50)	☑	
		□	

```sql
CREATE TABLE [dbo].[RegisterDataBase]
(
    [Id] INT NULL
    [StudentName] NVARCHAR(50) NULL,
    [Passwords] NVARCHAR(50) NULL,
    [EmailId] NVARCHAR(50) NULL,
    [Department] NVARCHAR(50) NULL,
    [College] NVARCHAR(50) NULL
)
```

Here, click database in update and subsequently click update the database.



Update for (LocalDB)\MSSQLLocalDB.C:\USERS\MUTHURAMALINGAM\DESKTOP\MCA\DATABASECO...
Creating update preview...
Displaying update preview...

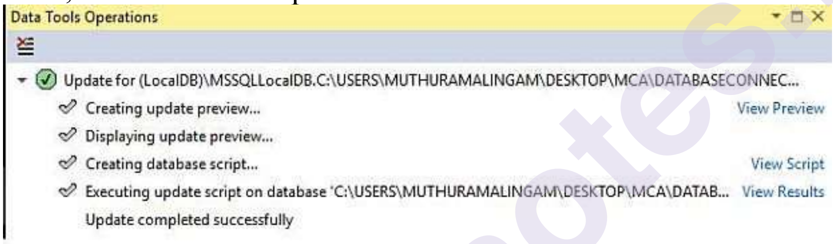
Highlights
None

User actions
Create
[dbo].[RegisterDataBase] (Table)

Supporting actions
None

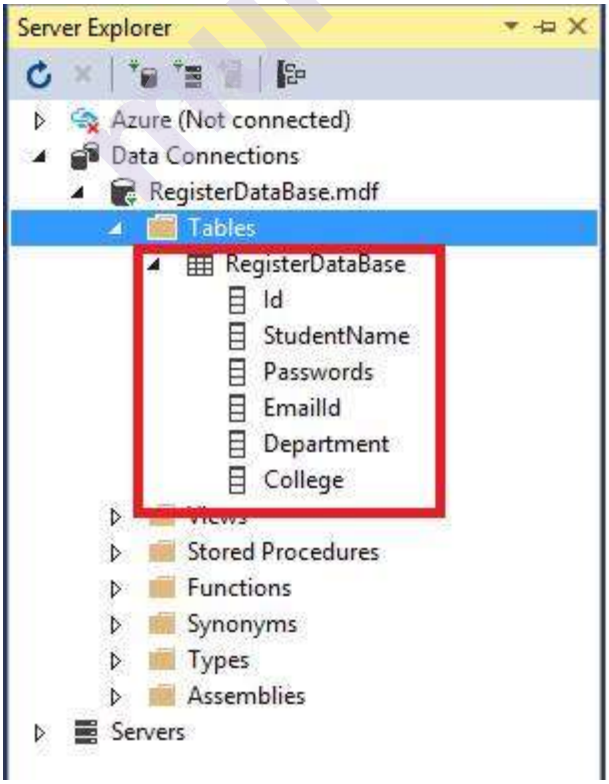
☑ Include transactional scripts

Here, the database is updated.



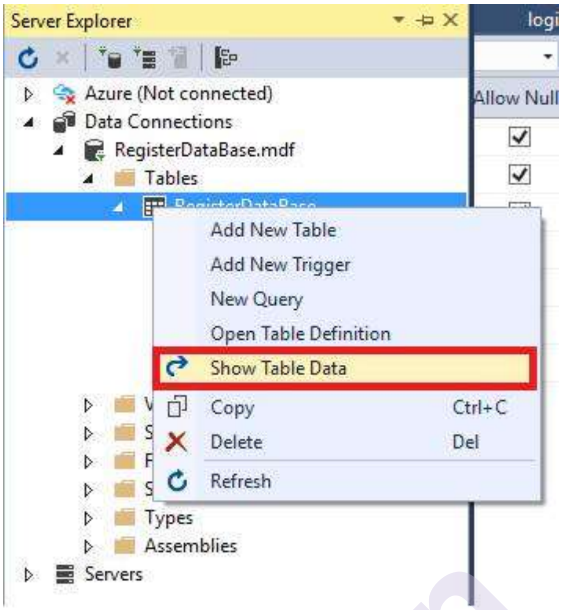
Update for (LocalDB)\MSSQLLocalDB.C:\USERS\MUTHURAMALINGAM\DESKTOP\MCA\DATABASECONNEC...
Creating update preview...
Displaying update preview...
Creating database script...
Executing update script on database 'C:\USERS\MUTHURAMALINGAM\DESKTOP\MCA\DATAB...
Update completed successfully

Here, the database and data are added.

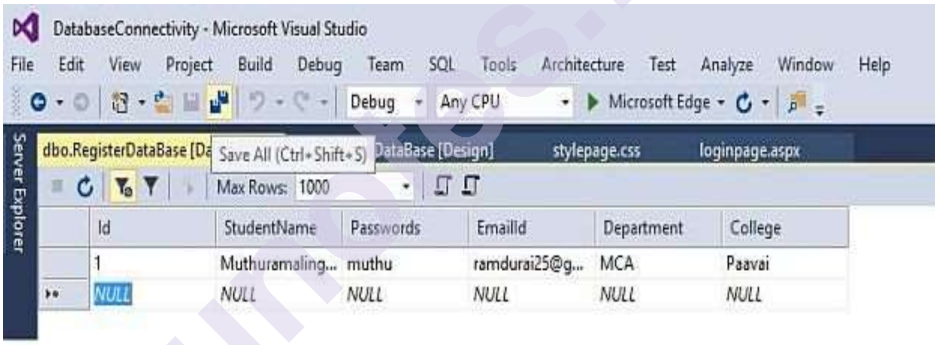


Azure (Not connected)
Data Connections
RegisterDataBase.mdf
Tables
RegisterDataBase
Id
StudentName
Passwords
EmailId
Department
College
Views
Stored Procedures
Functions
Synonyms
Types
Assemblies
Servers

Now, we can click on the right click and click Show Table Data.



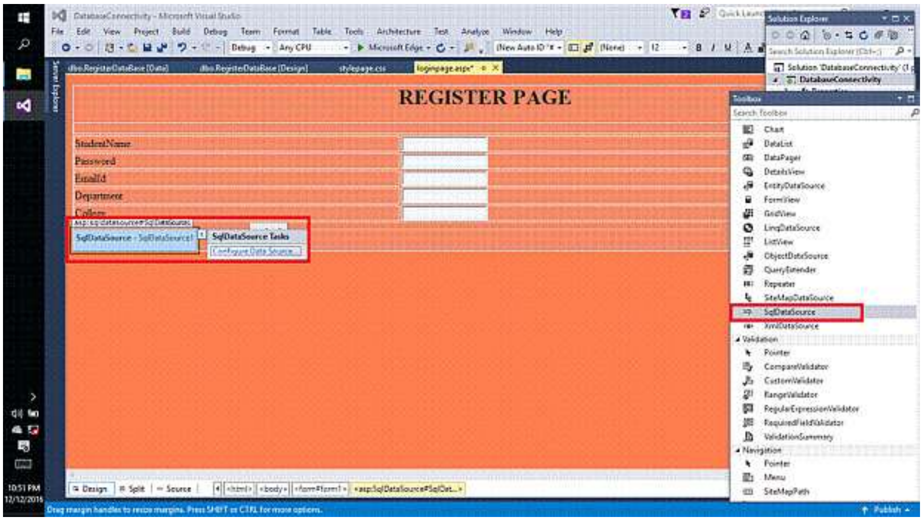
Now, the data is stored.


DatabaseConnectivity - Microsoft Visual Studio
File Edit View Project Build Debug Team SQL Tools Architecture Test Analyze Window Help
Debug Any CPU Microsoft Edge
dbo.RegisterDataBase [Da
Save All (Ctrl+Shift+S)
DataBase [Design]
stylepage.css
loginpage.aspx
Server Explorer
Max Rows: 1000

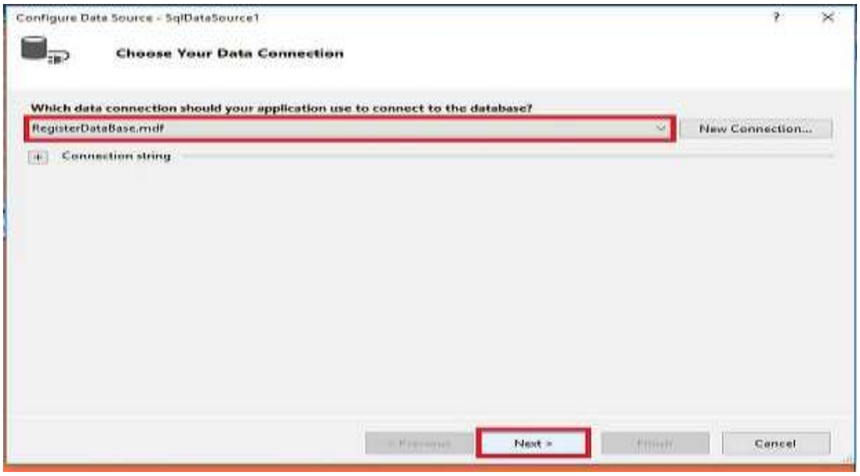
	Id	StudentName	Passwords	EmailId	Department	College
	1	Muthuramaling...	muthu	ramdurai25@g...	MCA	Paavai
▸*	NULL	NULL	NULL	NULL	NULL	NULL

Step 7

Now, you can add SQL Data Source. Drag and drop method. Here, click Configure Data Source



Now, we can choose your database and click NEXT button.



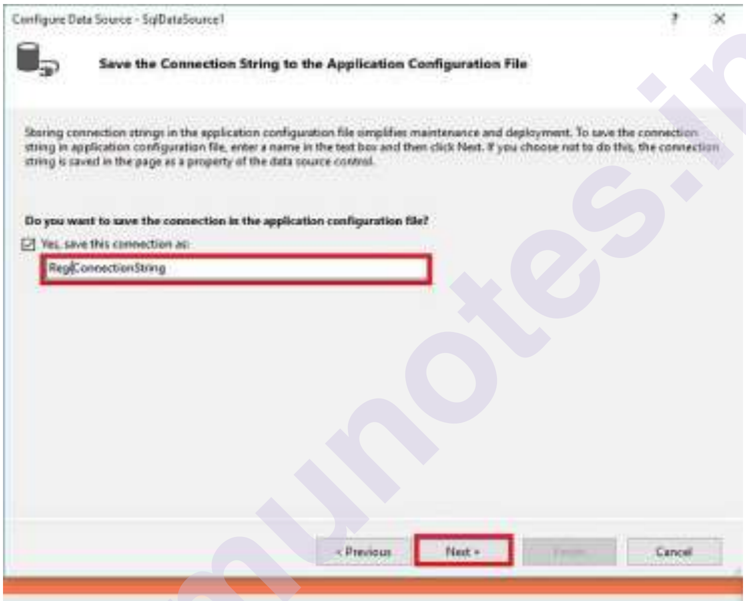
Configure Data Source - SqlDataSource1

Choose Your Data Connection

Which data connection should your application use to connect to the database?

RegisterDataBase.mdf

Now, you can select the ConnectionString and Click NEXT button



Configure Data Source - SqlDataSource1

Save the Connection String to the Application Configuration File

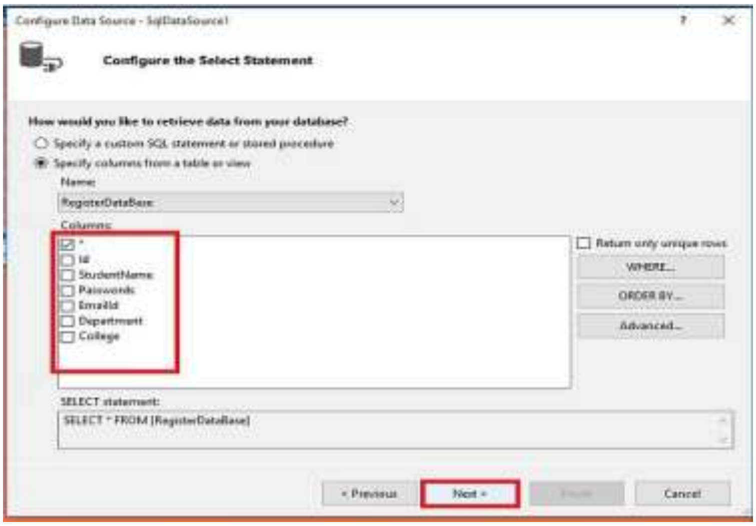
Storing connection strings in the application configuration file simplifies maintenance and deployment. To save the connection string in application configuration file, enter a name in the text box and then click Next. If you choose not to do this, the connection string is saved in the page as a property of the data source control.

Do you want to save the connection in the application configuration file?

☑ Yes, save this connection as:

RegConnectionString

Now, we can choose Specify columns from a table or view and afterwards, click Next button.



Configure Data Source - SqlDataSource1

Configure the Select Statement

How would you like to retrieve data from your database?

○ Specify a custom SQL statement or stored procedure

● Specify columns from a table or view

Name:

RegisterDataBase

Columns:

☑ *

□ Id

□ StudentName

□ Passwords

□ EmailId

□ Department

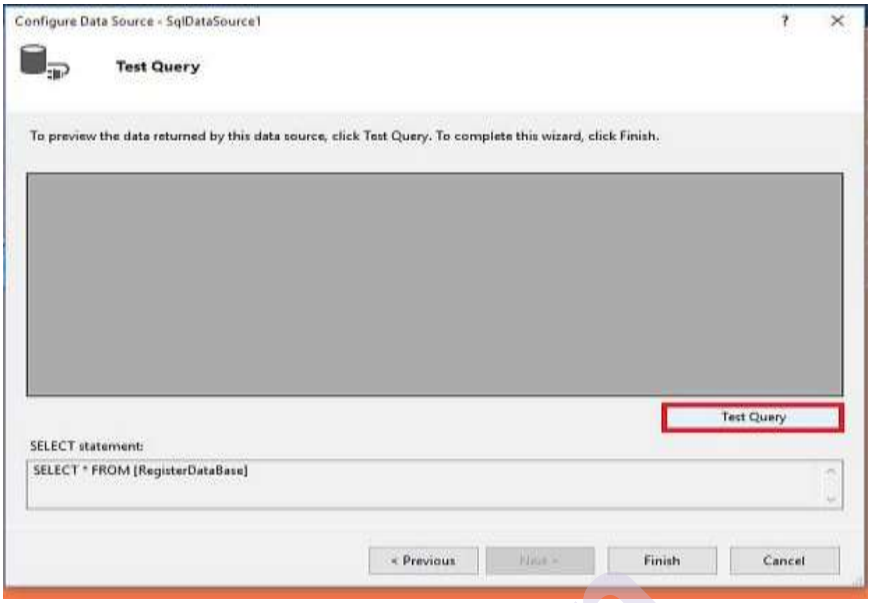
□ College

□ Return only unique rows

SELECT statement:

SELECT * FROM [RegisterDataBase]

Now, click Test Query.


Configure Data Source - SqlDataSource1

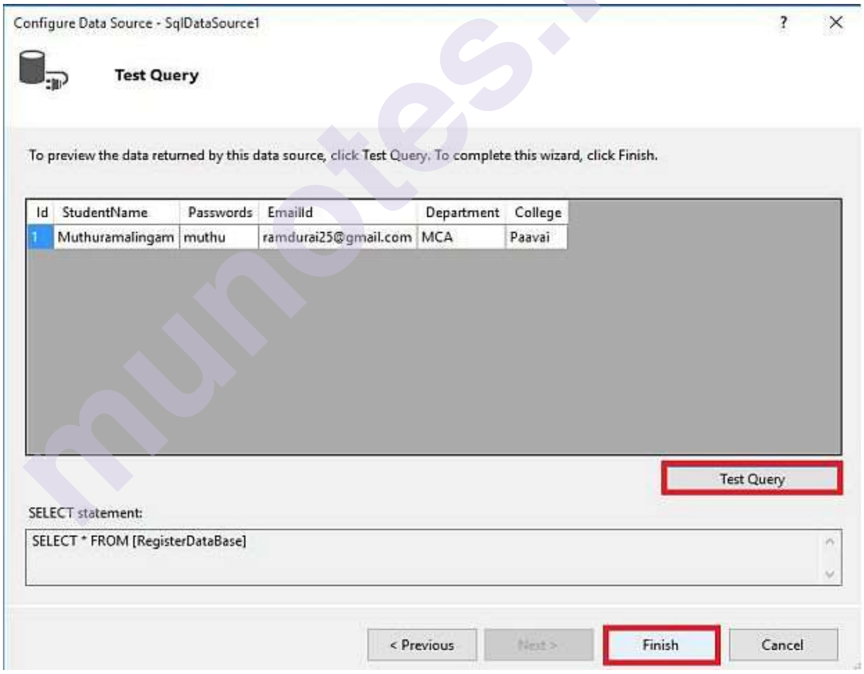
Test Query

To preview the data returned by this data source, click Test Query. To complete this wizard, click Finish.

SELECT statement:

```
SELECT * FROM [RegisterDataBase]
```

Here, add the data and click Finish button.


Configure Data Source - SqlDataSource1

Test Query

To preview the data returned by this data source, click Test Query. To complete this wizard, click Finish.

Id	StudentName	Passwords	EmailId	Department	College
1	Muthuramalingam	muthu	ramdurai25@gmail.com	MCA	Paavai

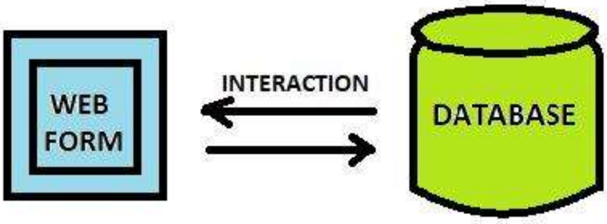
SELECT statement:

```
SELECT * FROM [RegisterDataBase]
```

9.7 USING DIRECT DATA ACCESS:

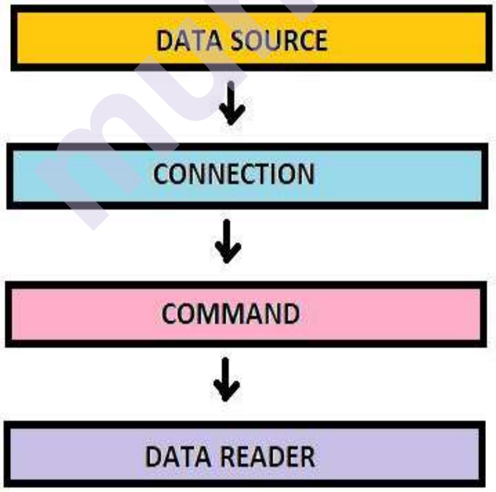
- It is also known as connected architecture refers to the fact that the connection is established for the full time between the database and application. For e.g. we make a program in C# that is connected with the database for the full time, so that will be connected architecture.
- Connected architecture 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.



DataReader in Connected architecture:

- DataReader class is used to read the data from the database. It works in forward only and reads the only mode and requires the connection for the complete time. That is why we use it in connected architecture.
- The forward only feature makes it an efficient way to read data. Thus we can say, DataReader is connection-oriented and requires an active connection while reading the data.



In order to make an object of the DataReader class, we never use the new keyword instead we call the ExecuteReader() of the command object. For e.g.

SqlCommand cmd= new SqlCommand("Select * from Table");

SqlDatareader rdr=cmd.ExecuteReader(cmd);

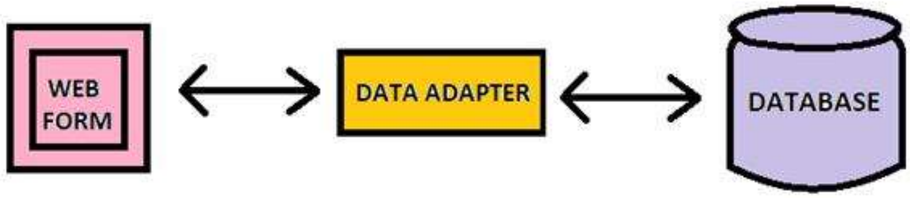
Here cmd.ExecuteReader() executes the command and creates the instance of DataReader class and loads the instance with data.

9.8 USING INDIRECT DATA ACCESS:

- 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 SqlDataAdapter 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. All the operations can be performed on the data using the Dataset and finally modified at the database.

DataAdapter in Disconnected architecture

- DataAdapter class acts as an interface between application and database. It provides the data to the Dataset which helps the user to perform the operations and finally the modifications are done in the Dataset which is passed to the DataAdapter which updates the database. DataAdapter takes the decision for the establishment and termination of the connection.



- DataAdapter is required for connectivity with the database. DataAdapter established a connection with the database and fetches the data from the database and fill it into the Dataset. And finally, when the task is completed it takes the data from the DataSet and updates it into the database by again establishing the connection.

- It can be said that DataAdapter acts as a mediator between the application and database which allows the interaction in disconnected architecture.

For example:-

```
public DataTable GetTable(string query)
        {
SqlDataAdapter adapter = new SqlDataAdapter(query, ConnectionString);
          DataTable Empl = new DataTable();
          adapter.Fill(Empl);
          return Empl;
        }
```

In the above lines of code, the object of the SqlDataAdapter is responsible for establishing the connection. It takes query and ConnectionString as a parameter. The query is issued on the database to fetch the data. ConnectionString allows connectivity with the database. The fill() of the SqlDataAdapter class adds the Table.

9.9 SUMMARY:

- A database is a collection of information which is structured for storing,managing and retrieving information.
- Primary key: Each table must have a column which uniquely identifies every record of the table which is called as Primary Key. For example, CustomerID.
- Foreign key: The column in a table which is used to reference a Primary key in another table is called as Foreign key.
- Types of Relationships among tables : One to One, One to Many, Many to Many
- ADO (ActiveX Data Object) is the database access technologies including components for retrieving data, storing data in memory and binding data to controls.
- ADO.NET provides common way to interact with data sources, but comes in different sets of libraries for each we can talk to a data source. These libraries are called Data Providers
- Direct data access or connected architecture refers to the fact that the connection is established for the full time between the database and application
- Indirect data access or 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.

9.10 REFERENCES :

Reference Books:

- Beginning ASP.NET 4.5 in C# by Apress.
- Murach's ASP.NET 4.6 Web Programming in C#2015 by SPD
- ASP.NET 4.0 programming by Tata McGrawHill.
- Programming ASP.NET by Microsoft Press

Web References:

- https://www.c-sharpcorner.com/
- https://www.aspsnippets.com/
- http://asp.net-informations.com/

9.11 UNIT END EXERCISES :

i. What do you mean by Database? How many types of relationships are available to have between tables?

ii. Explain the architecture of ADO DOT NET in detail.

iii. How to retrieve table data with SQL?

iv. How to manipulate table data with SQL?

v. What are the different data types in SQL server?

vi. How to create SQL server database and table?

vii. Explain different types of Data Providers.

viii. Write a note on Direct data access.

ix. Write a note on indirect data access.

❄❄❄❄❄❄❄

10

DATA BINDING

Unit Structure

10.1 OBJECTIVES:

This chapter would make you understand the following concepts:

- Concept of Data Binding.
- Single value and Repeated value data binding
- SqlDataSource Control

10.2 INTRODUCTION TO DATA BINDING:

- Data binding is the process of connecting the Application User Interface with the Data from database.
- Web Forms provides many controls that support data binding. You can connect these controls to ADO.NET components such as a DataView, DataSet, or DataViewManager at design-time as well as at run-time.
- The advantages of using data binding in .NET are as follows:-
    - Reduction in code size.
    - Better performance.
    - Rapid development of data driven application.
    - Fine control on data binding through events.

10.3 USING SINGLE-VALUE DATA BINDING:

- You use the single-item data-bound controls to display the value of a single item of a database table. These controls don't provide direct binding with the data source.
- Expression is enclosed in angle brackets and percent signs (<%....%>) and prefixed by the symbol (#).
- Use a single quotation mark (‘) to wrap the whole expression as shown below.
- Label1.Text = ‘<%# ”Hello,world” %>’;
- You use the Text, Caption, or Value property of these controls to show the data a field.
- Examples of single-item data-bound controls are textboxes, buttons, labels, images, and so on.
- Consider the following page, which defines a variable named LOC and uses it to point an image in the web directory.

Code :

```
public partial class SingleValueDB : System.Web.UI.Page
{
  protected string LOC;
  protected void Page_Load(Object sender,EventArgs e)
  {
      LOC = “Images/img01.jpg”;
      this.DataBind();
   }
}
```

- **With Label control:**

  `<asp:Label id=”lb1” runat=”server”><%#LOC%></asp:Label>`

- **With CheckBox control:**

  `<asp:CheckBox id=”chk1” “ text=”<%#LOC%>runat=”server” />`

10.4 USING REPEATED-VALUE DATA BINDING:

- You use the multi-item data bound controls to display the entire or a partial table. These controls provide direct binding to the data source.
- You use the DataSource property of these controls to bind a database table to these controls

- Some examples of multi-item data-bound controls are GridView, FormView, DetailsView, ListBox, DataList, DropDownList, and so on.
- Example : Data Binding with List Controls
- Create and fill data object. We can use any type of collection like ArrayList, Hashtable, Dictionary, Data Table, Data set object.
- Link the object to the respective control using Data Source property.
- Activate the binding using DataBind() method.

Let us take the following steps:

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

```
using System;
using System.Data;
using System.Configuration;
using System.Linq;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Xml.Linq;
namespace databinding
{
  public class booklist
  {
    protected String bookname;
    protected String authorname;
    public booklist(String bname, String aname)
    {
      this.bookname = bname;
      this.authorname = aname;
    }

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

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

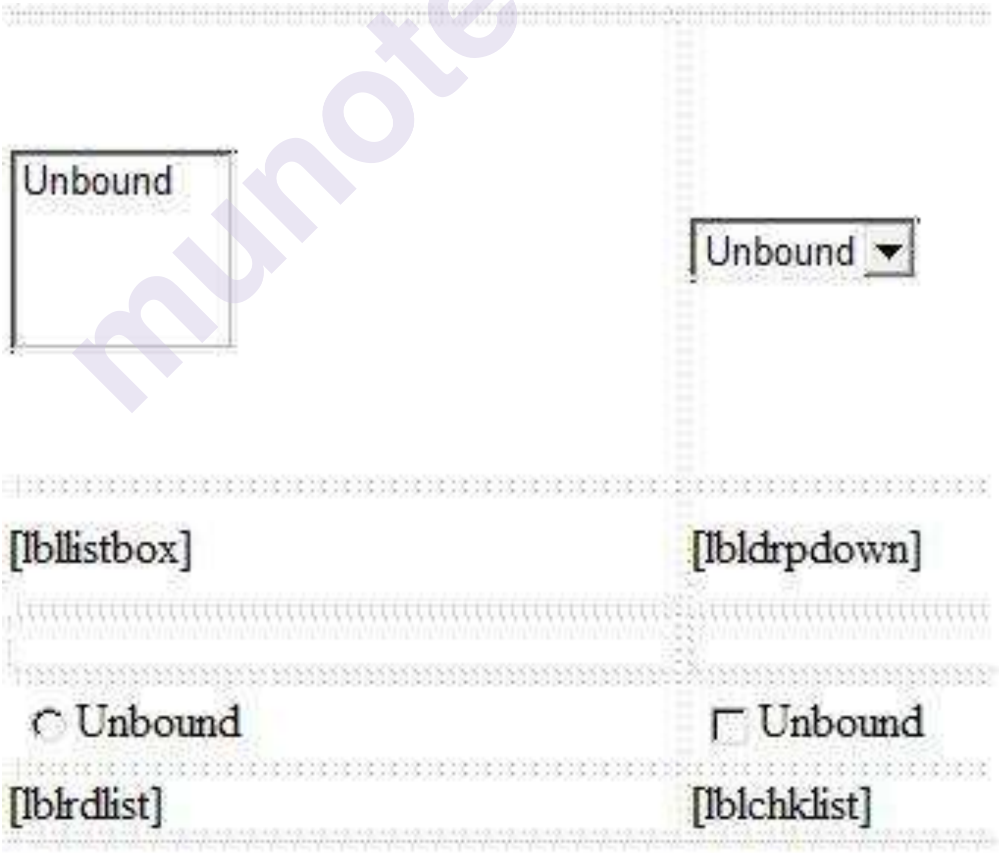


Fig. Form Design

The source file should look as the following:

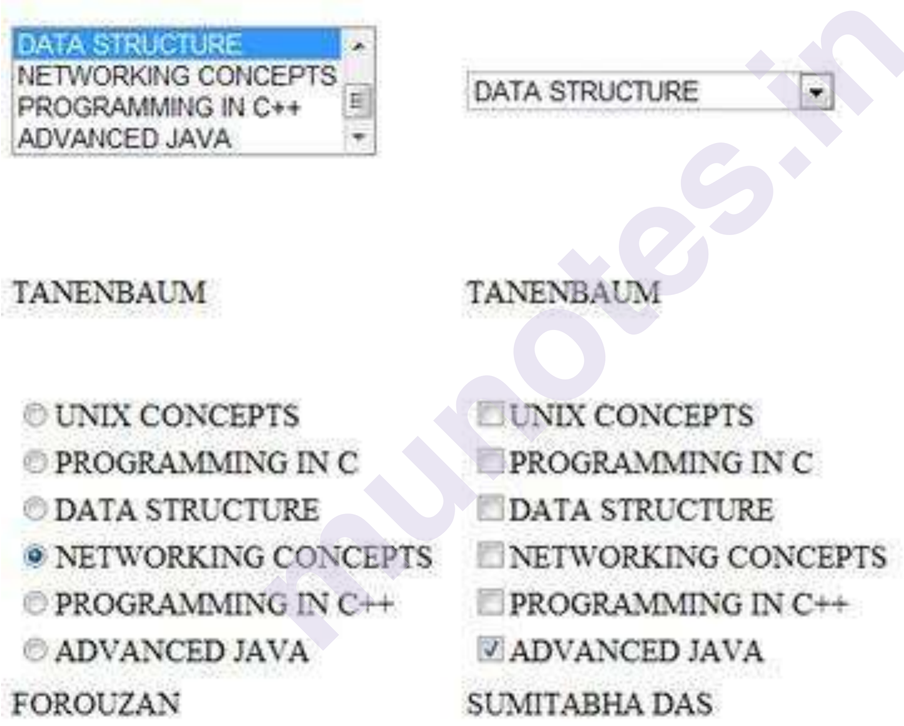
```
<form id="form1" runat="server">
  <div>

    <table style="width: 559px">
      <tr>
        <td style="width: 228px; height: 157px;">
          <asp:ListBox              ID="ListBox1"              runat="server"
AutoPostBack="True"
OnSelectedIndexChanged="ListBox1_SelectedIndexChanged">
          </asp:ListBox>
        </td>

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

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

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

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

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

      <tr>
        <td style="width: 228px; height: 21px">
          <asp:RadioButtonList ID="RadioButtonList1" runat="server"
            AutoPostBack="True"
OnSelectedIndexChanged="RadioButtonList1_SelectedIndexChanged">
          </asp:RadioButtonList>
        </td>

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

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

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

  </div>
</form>
```

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

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

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

      this.DropDownList1.DataSource = bklist;
      this.DropDownList1.DataTextField = "Book";
      this.DropDownList1.DataValueField = "Author";

      this.RadioButtonList1.DataSource = bklist;
      this.RadioButtonList1.DataTextField = "Book";
      this.RadioButtonList1.DataValueField = "Author";

      this.CheckBoxList1.DataSource = bklist;
      this.CheckBoxList1.DataTextField = "Book";
      this.CheckBoxList1.DataValueField = "Author";

      this.DataBind();
    }
  }

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

    bl = new booklist("UNIX CONCEPTS", "SUMITABHA DAS");
    allbooks.Add(bl);
    bl = new booklist("PROGRAMMING IN C", "RICHI
KERNIGHAN");
    allbooks.Add(bl);

    bl = new booklist("DATA STRUCTURE", "TANENBAUM");
    allbooks.Add(bl);
    bl = new booklist("NETWORKING CONCEPTS", "FOROUZAN");
    allbooks.Add(bl);

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

    bl = new booklist("ADVANCED JAVA", "SUMITABHA DAS");
    allbooks.Add(bl);

    return allbooks;
  }
  protected void ListBox1_SelectedIndexChanged(object sender,
EventArgs e)
  {
    this.lbllistbox.Text = this.ListBox1.SelectedValue;
  }
  protected void DropDownList1_SelectedIndexChanged(object sender,
EventArgs e)
  {
    this.lbldrpdown.Text = this.DropDownList1.SelectedValue;
  }
  protected void RadioButtonList1_SelectedIndexChanged(object sender,
EventArgs e)
  {
    this.lblrdlist.Text = this.RadioButtonList1.SelectedValue;
  }
protected void CheckBoxList1_SelectedIndexChanged(object sender,
EventArgs e)
  {
    this.lblchklist.Text = this.CheckBoxList1.SelectedValue;
  }
}
```

Observe the following:

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



Example : Data Binding with ADO.NET

- To use data binding with the data fetched from a database we need to first create data source which will be DataReader or DataSet object.
- DataReader offers the best performance, but it limits data binding to a single control because it is a forward only reader. After it is finished, it can’t go back to the beginning so it can’t be used in another data binding operation.
- For this reason, a DataSet is more common choice.
- To fill DataSet use following steps.

- Create the DataSet.
- Create new table and add it to the DataSet.Tables collection.
- Define the structure of the table by adding DataColumn objects to the DataTable.Column collection.
- Add the data using DataTable.NewRow() method.

Code :

```
<%@ Page Language="C#" %>
<%@Import namespace="System.Data.SqlClient" %>
<%@Import namespace="System.Data" %>
<html>
<body>
<head><TITLE>Bind Films data to a ListBox</TITLE>
<script  runat="Server">
void Page_Load(object sender, EventArgs e)
{      if(!this.IsPostBack)
       {
          getMovies();
       }
}
private void getMovies()
{
   string cnstr=GetConnString(); // Get a connection string
SqlConnection conn = new SqlConnection(cnstr);
IDbCommand cmd = new SqlCommand();
cmd.Connection= conn;
conn.Open();
cmd.Connection=conn;
string sql="SELECT movie_id, movie_title FROM movies";
SqlDataAdapter da = new SqlDataAdapter(sql,conn);
DataSet ds = new DataSet();
da.Fill(ds,"Movies");
dview = new DataView(ds.Tables["Movies"]);
dview.RowFilter = "movie_year < 1951";
// List box containing movies before 1951
ListBoxMovie.DataSource= dview;
ListBoxMovie.DataBind();
conn.Close();
}
</script>
</head>
<FORM NAME="FORM1" runat=server>
<asp:ListBox id=”ListBoxMovie” dataValueField = "movie_ID"
dataTextField = "movie_title" AppendDataBoundItems=true Rows="10"
BackColor=#efefe4 font-size=9pt runat="server" >
<asp:ListItem Value=-1 Text="Select Movie" />
</asp:ListBox>
</FORM> </body> </html>
```

10.5 WORKING WITH DATA SOURCE CONTROLS:

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

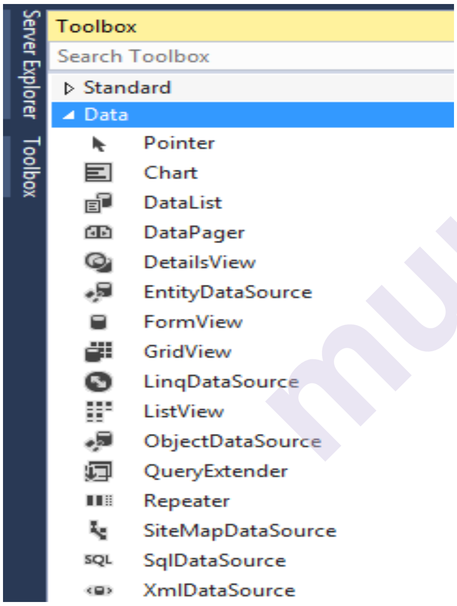


Fig Data Source Controls

- The data source controls used for **hierarchical** data are:
    - **XMLDataSource** - It allows binding to XML files and strings with or without schema information.
    - **SiteMapDataSource** - It allows binding to a provider that supplies site map information.
- The data source controls used for **tabular** data are:

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

The following table provides the methods of the DataSourceView class:

Methods	Description
CanExecute	Determines whether the specified command can be executed.
ExecuteCommand	Executes the specific command.
ExecuteDelete	Performs a delete operation on the list of data that the DataSourceView object represents.
ExecuteInsert	Performs an insert operation on the list of data that the DataSourceView object represents.
ExecuteSelect	Gets a list of data from the underlying data storage.
ExecuteUpdate	Performs an update operation on the list of data that the DataSourceView object represents.
Delete	Performs a delete operation on the data associated with the view.
Insert	Performs an insert operation on the data associated with the view.
Select	Returns the queried data.
Update	Performs an update operation on the data associated with the view.

10.5.1 The SqlDataSource Control

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

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

```
<asp:SqlDataSource runat="server" ID="MySqlSource"
  ProviderName='<%$ConnectionStrings:LocalNWind.ProviderName%>'
  ConnectionString='<%$ ConnectionStrings:LocalNWind %>'
  SelectionCommand= "SELECT * FROM EMPLOYEES" />
<asp:GridView ID="GridView1" runat="server"
DataSourceID="MySqlSource" />
```

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

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

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

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

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

10.6 SUMMARY

- Data binding is the process of connecting the Application User Interface with the Data from database.

- You use the single-item data-bound controls to display the value of a single item of a database table.

- You use the multi-item data bound controls to display the entire or a partial table.

- DataReader offers the best performance, but it limits data binding to a single control because it is a forward only reader. After it is finished, it can’t go back to the beginning so it can’t be used in another data binding operation.

- For this reason, a DataSet is more common choice.

- A data source control interacts with the data-bound controls and hides the complex data binding processes.

- These are the tools that provide data to the data bound controls and support execution of operations like insertions, deletions, sorting, and updates.

10.7 REFERENCES:

Reference Books:

- Beginning ASP.NET 4.5 in C# by Apress.
- Murach's ASP.NET 4.6 Web Programming in C#2015 by SPD
- ASP.NET 4.0 programming by Tata McGrawHill.
- Programming ASP.NET by Microsoft Press

Web References:

- https://www.c-sharpcorner.com/
- https://www.aspsnippets.com/
- http://asp.net-informations.com/

10.8 UNIT END EXERCISES :

1) What is Data Binding? Explain its types.
2) Explain Single Value Data binding.
3) Explain Repeated Value Data binding.
4) How to bind data with Simple List Control? Give Example.
5) Explain data binding with ADO.NET.
6) Write a note on Data Source Controls.
7) Explain Sql Data Source with properties.

❄❄❄❄❄❄❄

11

DATA CONTROLS

Unit Structure

11.1 OBJECTIVES:

This chapter would make you understand the following concepts:

- Introduction to GridView control and its different operations like select, edit etc.
- How to use DetailsView and FormView controls

11.2 INTRODUCTION OF GRID VIEW:

The GridView control displays the values of a data source in a table. Each column represents a field, while each row represents a record. The GridView control supports the following features:

- Binding to data source controls, such as SqlDataSource.
- Built-in sort capabilities.

- Built-in update and delete capabilities.
- Built-in paging capabilities.
- Built-in row selection capabilities.
- Programmatic access to the GridView object model to dynamically set properties, handle events, and so on.
- Multiple key fields.
- Multiple data fields for the hyperlink columns.
- Customizable appearance through themes and styles.

11.2.1 Creating a GridView:

```
<asp:GridView ID="gridService" runat="server">
</asp:GridView>
```

11.2.2 Properties of GridView :

ID	**The name assigned to the control.**
DataSourceID	**ID assigned to the Data Source.**
AllowSorting	**It is Boolean value. True value generates clickable column headers that sort the column's data when selected.**
AllowPaging	**It is Boolean value indicating whether control will provide paging with page size defined in PageSize property.**
PageSize	**No of records included on each page of data.**
BackColor	**Color name or hex value assigned to GridView background.**
BorderColor	**Color name or hex value assigned to GridView's border.**

11.3 FORMATTING THE GRIDVIEW:

- To format the grid view you have to ensure that dates, currency and other number values are in good format. Grid View has property "DataFormatString" to apply formatting.
- You can change colors, fonts, borders and alignment of grid. Each BoundField column provides a DataFormatString property that you can use to configure the numbers and dates using a format string.
- Format strings are generally made up of a placeholder and format indicator, which are wrapped inside curly brackets, like this: {0:C}
- Here 0 shows the value that will be formatted and the letter indicates a predetermined format style. In this case C means currency format which formats a number as a dollar.

```
<asp:BoundField DataField="Price" HeaderText="Price"
DataFormatString="{0:C}" HtmlEncode="false" />
```

Here we are going to discuss about few format strings.

11.3.1 Numeric Format Strings :

- Currency {0:C} - $1,234.50 Brackets indicate negative values($1,234.50). Currency sign is locale specific (?1,234.50).
- Scientific (Exponential) {0:E } - 1.234.50E+004
- Percentage {0:P} - 35.5%
- Fixed Decimal {0:F?} - Depends on the number of decimal places you set {0:F3} would be 123.400. {0:F0} would be 123.

11.3.2 Time and Date Format Strings:

<asp:BoundField DataField="DOB" HeaderText="DOB" DataFormatString="{0:MM/dd/yy}" HtmlEncode="false" />

- Short Date {0:d} - M/d/yyyy (11/21/2003)
- Long Date {0:D} - dddd, MMMM dd, yyyy (Saturday, March 21, 2001)
- Long Date and Short Time {0:f} - dddd, MMMM dd, yyyy HH:mm aa (Saturday, March 21, 2003 11:00 AM)
- Long Date and Long Time {0:F} - dddd, MMMM dd, yyyy HH:mm:ss aa (Saturday, March 21, 2003 11:00:20 AM)
- ISO Sortable Standard {0:s} - yyyy-MM-dd HH:mm:ss (2003-01-21 11:00:21)
- Month and Day {0:M} - MMMM dd (March 21)
- General {0:G} - M/d/yyyy HH:mm:ss aa (depends on local specific setting) (10/21/2003 11:00:21 AM)

11.3.3.Styles : you can set eight GridView styles.

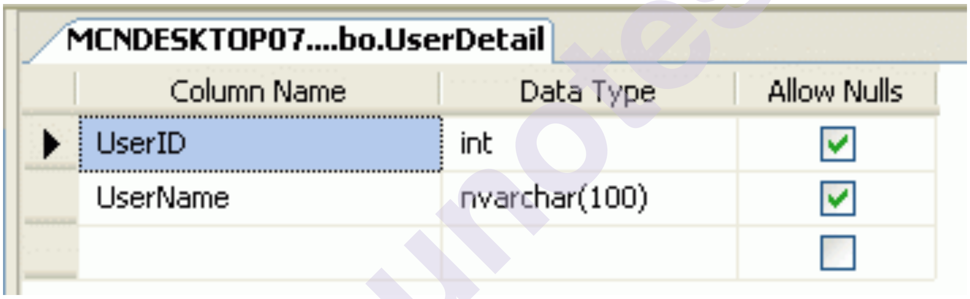
- Header Style: Set the header row style that contains column titles if you do ShowHeader property true.
- RowStyle: Set the style of every data row.
- AlternatingRowStyle: Set the style of every alternate row in gridview.
- SelectedRowStyle: Set the style of currently selected row.
- EditRowStyle: Set the style of row that is in edit mode. This formatting acts in addition to the RowStyle formatting.
- EmptyDataRowStyle: Set the style that is used for the single row in the special case where the bound data object contains no rows.
- FooterStyle: Set the style of the footer row at the bottom of the GridView, if you choose ShowFooter property true.
- PagerStyle: Set the style of the row with the page links if you enable AllowPaging property true.

Example:

```
<asp:GridView ID="GridView1" runat="server" AllowPaging="True" Sh
owFooter="True">
<RowStyle BackColor="Gray" Font-Italic="True" />
<EmptyDataRowStyle BackColor="Yellow" />
<PagerStyle BackColor="#FFC0C0" Font-Italic="True" Font-
Underline="True" />
<SelectedRowStyle BackColor="#00C000" Font-Bold="True" Font-
Italic="True" />
<EditRowStyle BackColor="#0000C0" />
<AlternatingRowStyle BackColor="Red" BorderColor="Green" BorderSt
yle="Dashed"   BorderWidth="1px" Font-Bold="True" />
<FooterStyle BackColor="#00C0C0" />
<HeaderStyle BackColor="#00C000" Font-Bold="True" Font-
Italic="True" Font-Underline="True" />
</asp:GridView>
```

11.4 CREATING TABLE IN SQL SERVER DATABASE:

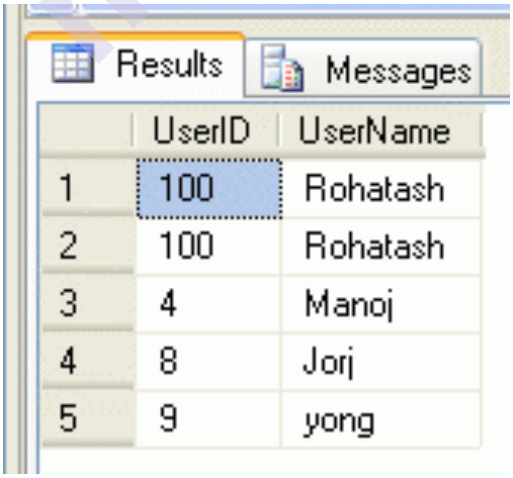
Step 1 : Create a table named UserDetail with the columns UserID and UserName. The table looks as below.



MCNDESKTOP07....bo.UserDetail

	Column Name	Data Type	Allow Nulls
▶	UserID	int	☑
	UserName	nvarchar(100)	☑
			☐

Step 2: Now insert data into the table.



	UserID	UserName
1	100	Rohatash
2	100	Rohatash
3	4	Manoj
4	8	Jorj
5	9	yong

Step 3 : Drag the GridView control from the Data controls menu.

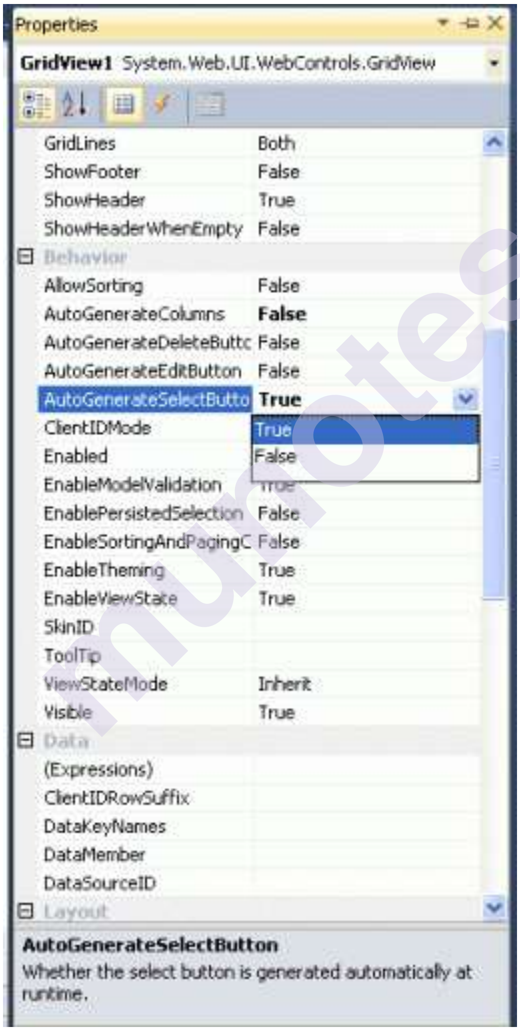
You will need to set the Bound column in order to see the text in the cells of the GridView control.

.aspx source code

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeBehind="WebForm1.aspx.cs"
Inherits="WebApplication120.WebForm1" %>

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

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

<head runat="server">

   <title></title>

</head>

<body>

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

   <div>

      UserID:<asp:TextBox ID="TextBoxUserID" runat="server">
</asp:TextBox>

      <br />

      UserName:

      <asp:TextBox ID="TextBoxUserName" runat="server">
</asp:TextBox>

      <br />

      <br />

      <asp:GridView ID="GridView1" runat="server"
AutoGenerateSelectButton="True"
OnSelectedIndexChanged="GridView1_SelectedIndexChanged"

         BackColor="#DEBA84" BorderColor="#DEBA84"
BorderStyle="None" BorderWidth="1px" CellPadding="3"
CellSpacing="2" AutoGenerateColumns="False">

         <FooterStyle BackColor="#F7DFB5" ForeColor="#8C4510" />

         <HeaderStyle BackColor="#A55129" Font-Bold="True"
ForeColor="White" />

         <PagerStyle ForeColor="#8C4510" HorizontalAlign="Center" />

         <RowStyle BackColor="#FFF7E7" ForeColor="#8C4510" />

         <SelectedRowStyle BackColor="#738A9C" Font-Bold="True"
ForeColor="White" />

         <SortedAscendingCellStyle BackColor="#FFF1D4" />

         <SortedAscendingHeaderStyle BackColor="#B95C30" />

         <SortedDescendingCellStyle BackColor="#F1E5CE" />

         <SortedDescendingHeaderStyle BackColor="#93451F" />

         <Columns>
```

```
            <asp:BoundField HeaderText="UserID" DataField="UserID" />
            <asp:BoundField                HeaderText="UserName"
DataField="UserName" />
          </Columns>
        </asp:GridView>
    </div>
    </form>
</body>
</html>
```

See the following image of a GridView after setting 'AutoGenerateSelectButton=True'.

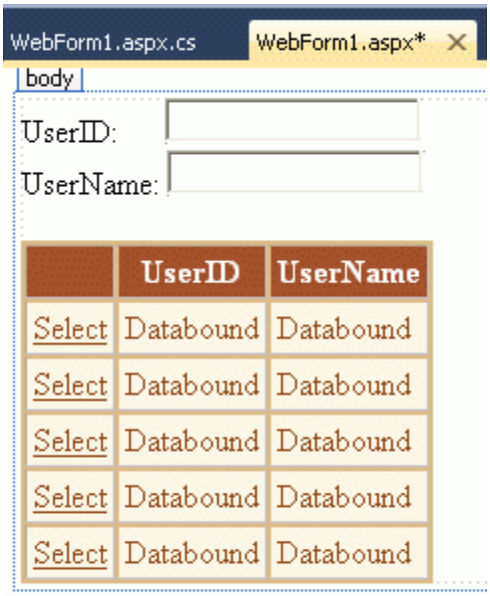


GridView1 System.Web.UI.WebControls.GridView

GridLines	Both
ShowFooter	False
ShowHeader	True
ShowHeaderWhenEmpty	False
⊟ Behavior	
AllowSorting	False
AutoGenerateColumns	**False**
AutoGenerateDeleteButtc	False
AutoGenerateEditButton	False
AutoGenerateSelectButto	**True**
ClientIDMode	
Enabled	
EnableModelValidation	True
EnablePersistedSelection	False
EnableSortingAndPagingC	False
EnableTheming	True
EnableViewState	True
SkinID	
ToolTip	
ViewStateMode	Inherit
Visible	True
⊟ Data	
(Expressions)	
ClientIDRowSuffix	
DataKeyNames	
DataMember	
DataSourceID	
⊟ Layout	

AutoGenerateSelectButton
Whether the select button is generated automatically at runtime.

Step 4 : See the Design view of your GridView. You will find a Hyperlink with a text as 'Select'.



UserID:

UserName:

	UserID	UserName
Select	Databound	Databound
Select	Databound	Databound
Select	Databound	Databound
Select	Databound	Databound
Select	Databound	Databound

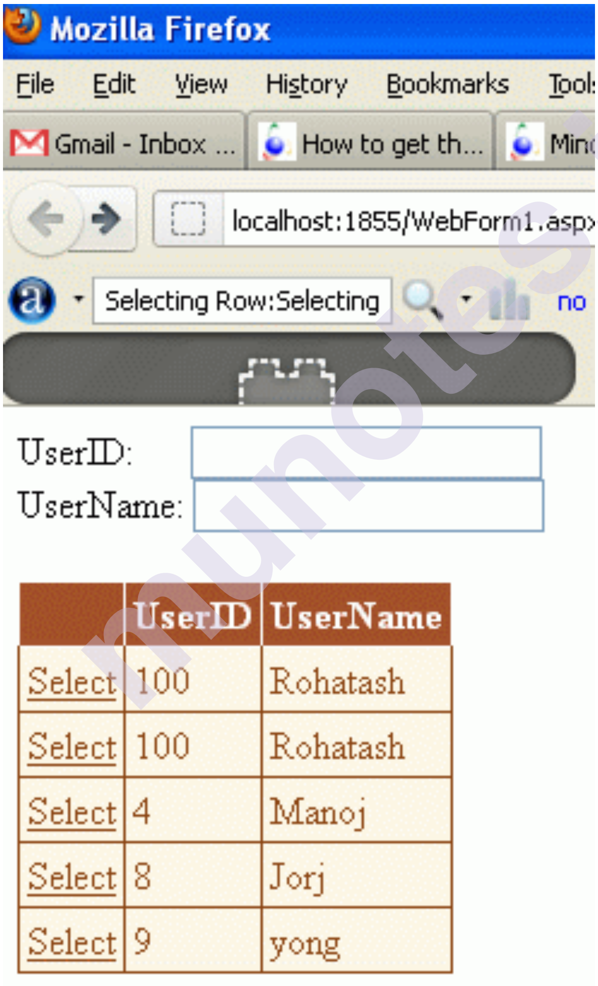
Step 5 : Now double-click on the page and write the following code for binding the data with the GridView.

Source code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Data.SqlClient;
using System.Data;
namespace WebApplication120
{
  public partial class WebForm1 : System.Web.UI.Page
  {
    protected void Page_Load(object sender, EventArgs e)
    {
      show();
    }
    private void show()
    {
      {
        SqlConnection con = new SqlConnection("Data Source=.;
uid=sa; pwd=wintellect; database=Rohatash;");
        string strSQL = "Select * from UserDetail";
```

```
                SqlDataAdapter dt = new SqlDataAdapter(strSQL, con);
                DataSet ds = new DataSet();
                dt.Fill(ds, "UserDetail");
                con.Close();
                GridView1.DataSource = ds;
                GridView1.DataBind();
            }
        }
    }
}
```

Step 6 :Now run the application.


Mozilla Firefox
File Edit View History Bookmarks Tool
Gmail - Inbox ... How to get th... Minc
localhost:1855/WebForm1.aspx
Selecting Row:Selecting
no

UserID:

UserName:

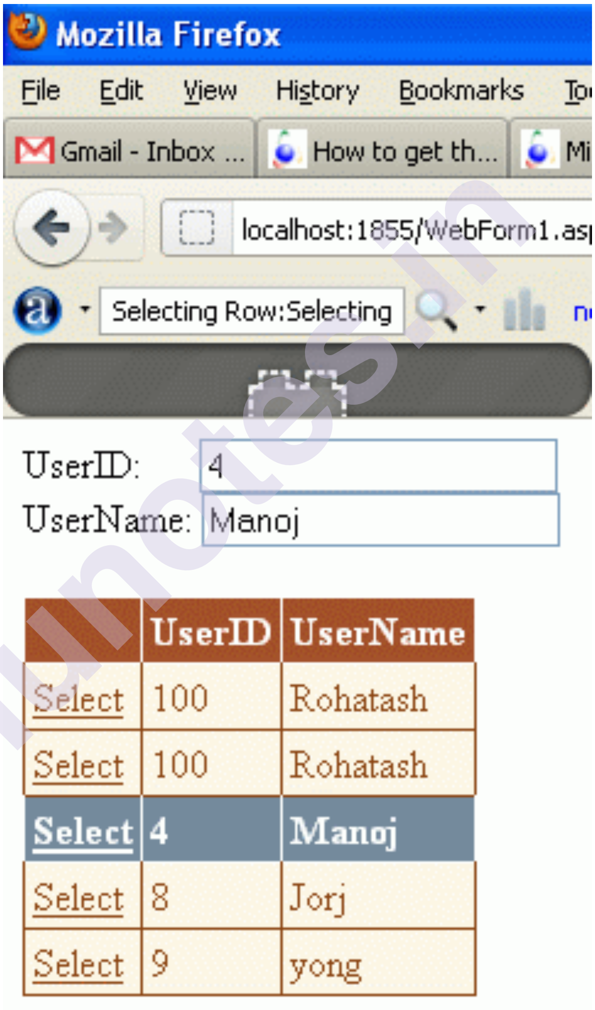
	UserID	UserName
Select	100	Rohatash
Select	100	Rohatash
Select	4	Manoj
Select	8	Jorj
Select	9	yong

11.5 SELECTING A GRID VIEW ROW:

Selecting the row event is fired when you make a click on the select link. If you need any particular item in that row you can easily select it using the cells property. In the Gridview, double-Click on the SelectedIndexChanged Event and write the following code:

```
protected void GridView1_SelectedIndexChanged(object sender, EventArgs e)

{

   TextBoxUserID.Text = GridView1.SelectedRow.Cells[1].Text;

   TextBoxUserName.Text = GridView1.SelectedRow.Cells[2].Text;

}
```

Now run the application and select a row; that will show the selected row data in the TextBoxes.



UserID: 4

UserName: Manoj

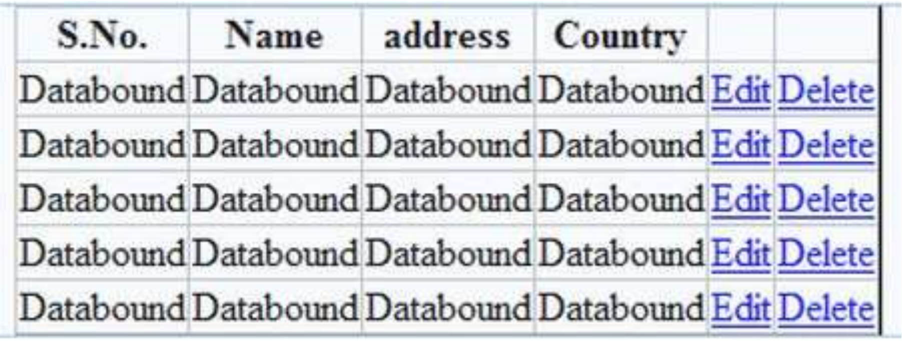
	UserID	UserName
Select	100	Rohatash
Select	100	Rohatash
Select	4	Manoj
Select	8	Jorj
Select	9	yong

11.6 EDITING WITH THE GRIDVIEW:

There is a method in GridView to edit, delete and update.

Design

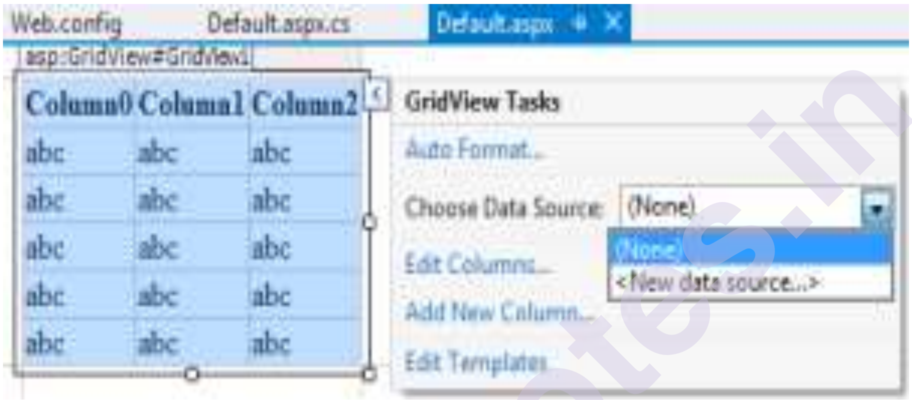
The design part will look as in the following image:



S.No.	Name	address	Country		
Databound	Databound	Databound	Databound	Edit	Delete
Databound	Databound	Databound	Databound	Edit	Delete
Databound	Databound	Databound	Databound	Edit	Delete
Databound	Databound	Databound	Databound	Edit	Delete
Databound	Databound	Databound	Databound	Edit	Delete

Using a CommandField , we can display one or more command buttons in a column of Grid control.

To add the Command field in a Grid view control, first select the control and click on add new column. It will display the following window.



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

From this, select command buttons “Edit/Update”

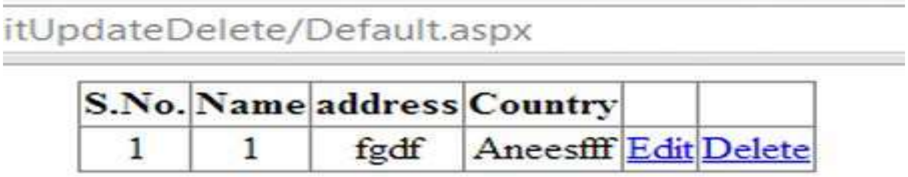
Code behind:

```
using System;
using System.Configuration;
using System.Data;
using System.Data.SqlClient;
using System.Drawing;
using System.Linq;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Xml.Linq;
public partial class _Default: System.Web.UI.Page {
  private SqlConnection conn = new SqlConnection("Data
Source=NEHASHAMA; Integrated Security=true; Initial Catalog=rp");
  protected void Page_Load(object sender, EventArgs e) {
    if (!IsPostBack) {
      gvbind();
    }
  }
  protected void gvbind() {
    conn.Open();
    SqlCommand cmd = new SqlCommand("Select * from detail",
conn);
    SqlDataAdapter da = new SqlDataAdapter(cmd);
    DataSet ds = new DataSet();
    da.Fill(ds);
    conn.Close();
    if (ds.Tables[0].Rows.Count > 0) {
      GridView1.DataSource = ds;
      GridView1.DataBind();
    } else {
      ds.Tables[0].Rows.Add(ds.Tables[0].NewRow());
      GridView1.DataSource = ds;
      GridView1.DataBind();
      int columncount = GridView1.Rows[0].Cells.Count;
      GridView1.Rows[0].Cells.Clear();
      GridView1.Rows[0].Cells.Add(new TableCell());
      GridView1.Rows[0].Cells[0].ColumnSpan = columncount;
      GridView1.Rows[0].Cells[0].Text = "No Records Found";
    }
  }
  protected void GridView1_RowDeleting(object sender,
GridViewDeleteEventArgs e) {
    GridViewRow row = (GridViewRow)
GridView1.Rows[e.RowIndex];
    Label lbldeleteid = (Label) row.FindControl("lblID");
    conn.Open();
    SqlCommand cmd = new SqlCommand("delete FROM detail where
id='" +
Convert.ToInt32(GridView1.DataKeys[e.RowIndex].Value.ToString()) +
"'", conn);
```

```
        cmd.ExecuteNonQuery();
        conn.Close();
        gvbind();
    }
    protected void GridView1_RowEditing(object sender,
GridViewEditEventArgs e) {
        GridView1.EditIndex = e.NewEditIndex;
        gvbind();
    }
    protected void GridView1_RowUpdating(object sender,
GridViewUpdateEventArgs e) {
        int userid =
Convert.ToInt32(GridView1.DataKeys[e.RowIndex].Value.ToString());
        GridViewRow row = (GridViewRow)
GridView1.Rows[e.RowIndex];
        Label lblID = (Label) row.FindControl("lblID");
        //TextBox txtname=(TextBox)gr.cell[].control[];
        TextBox textName = (TextBox) row.Cells[0].Controls[0];
        TextBox textadd = (TextBox) row.Cells[1].Controls[0];
        TextBox textc = (TextBox) row.Cells[2].Controls[0];
        //TextBox textadd = (TextBox)row.FindControl("txtadd");
        //TextBox textc = (TextBox)row.FindControl("txtc");
        GridView1.EditIndex = -1;
        conn.Open();
        //SqlCommand cmd = new SqlCommand("SELECT * FROM detail",
conn);
        SqlCommand cmd = new SqlCommand("update detail set name='" +
textName.Text + "',address='" + textadd.Text + "',country='" + textc.Text
+ "'where id='" + userid + "'", conn);
        cmd.ExecuteNonQuery();
        conn.Close();
        gvbind();
        //GridView1.DataBind();
    }
    protected void GridView1_PageIndexChanging(object sender,
GridViewPageEventArgs e) {
        GridView1.PageIndex = e.NewPageIndex;
        gvbind();
    }
```

```
 protected void GridView1_RowCancelingEdit(object sender, GridViewCancelEditEventArgs e) {
        GridView1.EditIndex = -1;
        gvbind();
    }
}
```

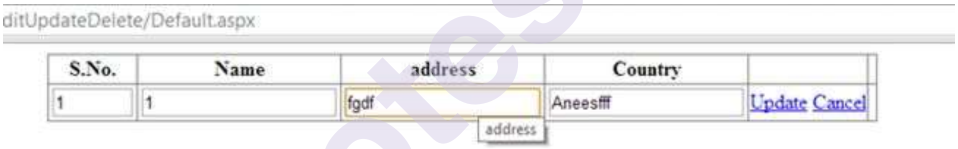
Save all or press "Ctrl+S" and hit "F5" to run the page, the page will look as in the following image:



itUpdateDelete/Default.aspx

S.No.	Name	address	Country		
1	1	fgdf	Aneesfff	Edit	Delete

Click on "Edit the GridView", it will display Textboxes in each cell as in the following image:



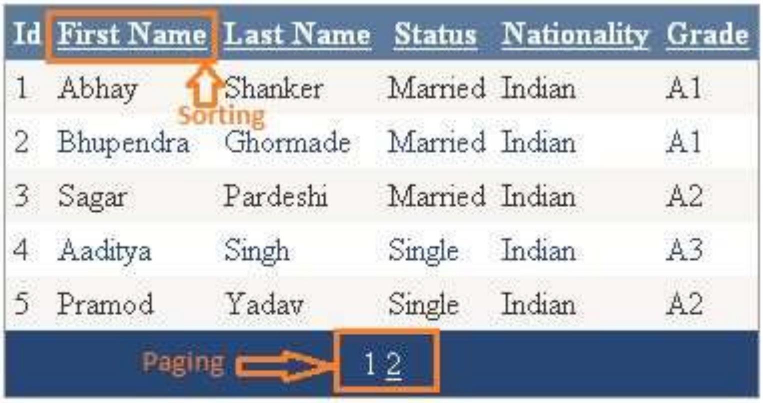
ditUpdateDelete/Default.aspx

S.No.	Name	address	Country	
1	1	fgdf	Aneesfff	Update Cancel

Edit the value(s) here and click on the Update link, it will update all the data or to remove it click on the "Delete" link above the image shown.

11.7 SORTING AND PAGING THE GRID VIEW:

The ASP.NET GridView control is used to display the values of a data source in a table. ASP.NET provides the sorting feature in a GridView Control. The records displayed in a GridView control can be sorted in ascending or descending order.



Id	First Name	Last Name	Status	Nationality	Grade
1	Abhay	Shanker	Married	Indian	A1
2	Bhupendra	Ghormade	Married	Indian	A1
3	Sagar	Pardeshi	Married	Indian	A2
4	Aaditya	Singh	Single	Indian	A3
5	Pramod	Yadav	Single	Indian	A2

In this article I will explain how to do paging and sorting in ASP.NET GridView.

The following is the step-by-step explanation.

Step1: Create a table in the database.

```
CREATE TABLE [dbo].[Teacher](
[TeacherId] [int] NULL,
[FirstName] [varchar](50) NULL,
[LastName] [varchar](50) NULL,
[Status] [varchar](50) NULL,
[Nationality] [varchar](50) NULL,
[Grade] [nchar](10) NULL
) ON [PRIMARY]
```

Step 2: Create a new ASP.NET web application and drag a GridView control in the Default.aspx design view. Set the property AllowSorting="true".

Step 3: Write the following in the page load event:

```
if (!Page.IsPostBack)
{
gvTeacher.DataSource = BindGridView();
gvTeacher.DataBind();
}
```

Step 4: The BindGridView() method populates the data in the GridView. Write the following method in the Default.aspx.cs file:

```
protected void gvTeacher_Sorting(object sender, GridViewSortEventArgs e)
{
string sortingDirection = string.Empty;
if (direction == SortDirection.Ascending)
{
direction = SortDirection.Descending;
sortingDirection = "Desc";
  }
else
{
direction = SortDirection.Ascending;
sortingDirection = "Asc";
}
DataView sortedView = new DataView(BindGridView());
sortedView.Sort = e.SortExpression + " " + sortingDirection;
Session["SortedView"] = sortedView;
gvTeacher.DataSource = sortedView;
gvTeacher.DataBind();
}
```

Note: GridViewSortEventArgs is used to perform GridView sorting. There are two properties.

SortDirection specifies the direction to sort the GridView column, either by ascending or descending order.

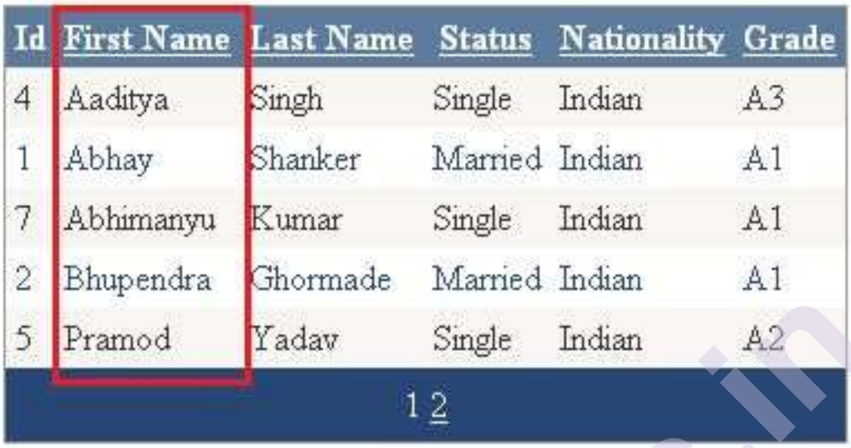
Step 5: Select the GridView and double-click on PageIndexChanging .Write the following code:

```
protected void gvTeacher_PageIndexChanging(object sender,
GridViewPageEventArgs e)
{
gvTeacher.PageIndex = e.NewPageIndex;
if (Session["SortedView"] != null)
{
gvTeacher.DataSource = Session["SortedView"];
gvTeacher.DataBind();
}
else
{
gvTeacher.DataSource = BindGridView();
gvTeacher.DataBind();
}
}
```

Step 6: Now maintain the SortDirection (Ascending or Descending) in ViewState by adding the following code.

```
public SortDirection direction
{
get
{
if (ViewState["directionState"] == null)
{
ViewState["directionState"] = SortDirection.Ascending;
}
return (SortDirection)ViewState["directionState"];
}
set
{
ViewState["directionState"] = value;
}
}
```

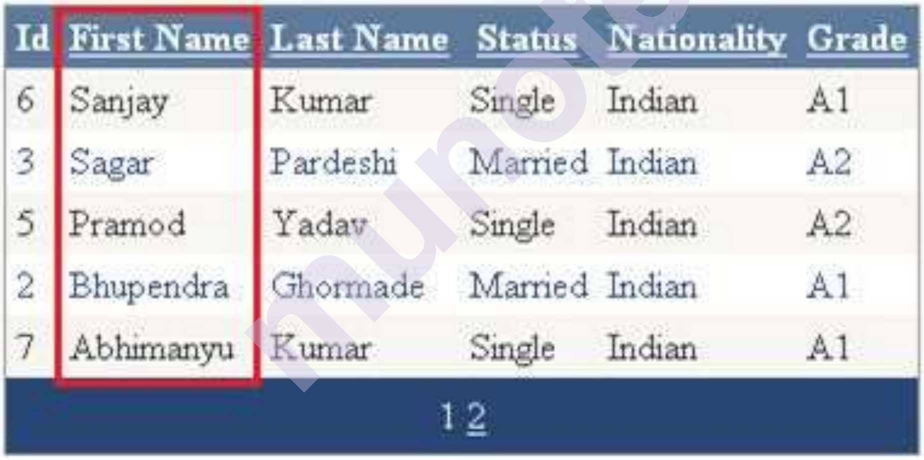
Step 7: The First Name is in ascending order as in the following:



Id	First Name	Last Name	Status	Nationality	Grade
4	Aaditya	Singh	Single	Indian	A3
1	Abhay	Shanker	Married	Indian	A1
7	Abhimanyu	Kumar	Single	Indian	A1
2	Bhupendra	Ghormade	Married	Indian	A1
5	Pramod	Yadav	Single	Indian	A2

1 2

The First Name in descending order as in the following:



Id	First Name	Last Name	Status	Nationality	Grade
6	Sanjay	Kumar	Single	Indian	A1
3	Sagar	Pardeshi	Married	Indian	A2
5	Pramod	Yadav	Single	Indian	A2
2	Bhupendra	Ghormade	Married	Indian	A1
7	Abhimanyu	Kumar	Single	Indian	A1

1 2

11.8 USING GRIDVIEW TEMPLATES:

i. We use TemplateFields when we wish to display ASP.Net controls in a GridView column.

ii. We display ASP.Net controls in a GridView column to provide additional functionality in the user interface.

iii. For example, by placing a DropDownList control in a GridView column, users will be able to select a list of options from within the Gridview control interface.

iv. A Template field supports many types of templates and a list of template types is given in the table below.

TemplateType	Description
AlternatingItemTemplate	The contents of this template are displayed for every other row rendered by the GridView
EditItemTemplate	The contents of this template are displayed when a row is selected for editing
FooterTemplate	The contents of this template are displayed in the column footer
HeaderTemplate	The contents of this template are displayed in the column header
InsertTemplate	The contents of this template are displayed when a new data item is inserted
ItemTemplate	The contents of this template are displayed for every row rendered by the GridView

We can create TemplateFields in the GridView control using <TemplateField> element.

Steps to create the <TemplateField> element in the GridView control

a. Declare the GridView and set the AutoGenerateColumns property to 'false'.

b. Create a Template column using <asp:TemplateField> tag within the <Columns> element.

Create within the <asp:TemplateField> element to display value of field as text.

Create <EditItemTemplate> to display TextBox control to modify value of field when editing the record.

```
<asp:GridView ID="GridView1" DataSourceId="MyDataSource"
DataKeyNames="Code" AutoGenerateColumns="false"
AutoGenerateEditButton="true" AutoGenerateDeleteButton="true"
runat="server">
<Columns>
<asp:TemplateField HeaderText="Name">
<ItemTemplate>
<%#Eval("Name")%>
</ItemTemplate>
<EditItemTemplate>
<asp:TextBox ID="txtName" Text='<%# Bind("Name")%>'
runat="server" />
</EditItemTemplate>
</asp:TemplateField>
<asp:TemplateField HeaderText="Description">
<ItemTemplate>
<%#Eval("Description")%>
</ItemTemplate>
<EditItemTemplate>
```

```
<asp:TextBox ID="txtDesctiption"Text='<%# Bind("Description")%>'
runat="server" />
</EditItemTemplate>
</asp:TemplateField>
</Columns>
</asp:GridView>

<asp:SqlDataSource ID="MyDataSource"
ConnectionString="<%$Connectionstrings:ERPConnectionString%>"
SelectCommand="SELECT * FROM Sample"
UpdateCommand="Update SAMPLE SET
Name=@Name,Description=@Description Where Code=@Code"
DeleteCommand="Delete SAMPLE Where Code=@Code"
runat="server"/>
```

- Configure the SqlDataSource control and set the SelectCommand and UpdateCommand properties to display and update records from the Sample Table.
- We have created two Template fields the GridView control.
- The first TemplateField is used to display and edit the value of the 'Name' field in the SAMPLE table.
- The second TemplateField is used to display and edit the value of the 'Description' field in the SAMPLE table.
- The contents of the ItemTemplate are displayed when a row is in normal mode and the contents of the EditItemTemplate are displayed when a row is in edit mode.

Extra Example:

Now let’s take an asp.net example to bind data in to GridView.

Here, we learn how to bind and display data in GridView control from sql server database in asp.net c#.

Step 1 – Open the Visual Studio –> Create a new empty Web application.

Step 2 – Create a New web page.

Step 3 – Drag and drop GridView Control on web page from toolbox.

Step 4 – Create New Database in SQL Server

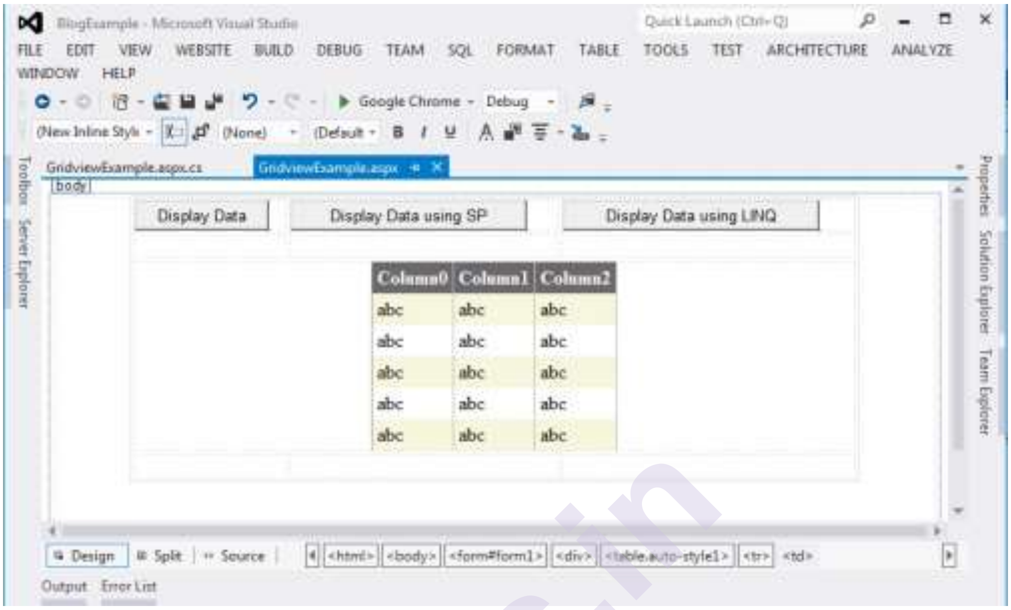
Step 5 – Make connection between Database and web application.

In this asp.net example we will bind data to GridView from SQL server database. So, we need to do connection between our web application and sql server database. There are several method for make connection, we will explain three different connection method in our example.

1. Bind data using SQL Connection and SQL DataAdapter
2. Bind Data using DataSet and Table Adapter
3. Bind Data using LINQ method.

Please check the following steps:

Step 1: Design asp.net web form with GridView control along with three



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

button as show s in below figure.

ASP.Net GridView Control.

Step 2: add namespace on code page.

using System.Data.SqlClient;

using System.Data;

Step 3: Bind data using SQL Connection and SQL DataAdapter

Now, write below code on 'Display Data' button. This connection method we use SQLConnection and SQLDataAdapter object and write sql query in code behind page. This sql connection method are basic connection method.

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

{

SqlConnection SQLConn = new SqlConnection("Data Source=.\\SQLEXPRESS;Initial Catalog='Blog';Integrated Security=True");

SqlDataAdapter SQLAdapter = new SqlDataAdapter("Select * from UserMst", SQLConn);

DataTable DT = new DataTable();

SQLAdapter.Fill(DT);
GridView1.DataSource = DT;

GridView1.DataBind();

}
```

Step 4: Bind Data using DataSet and Table Adapter

Write below code on ‘Display Data using SP’ Button.

In this method we use DataSet and SQL Stored Procedure for sql connection. Use Table Adapter instead of DataAdapter.

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

{

DS_USER.USERMST_SELECTDataTable UDT = new

DS_USER.USERMST_SELECTDataTable();

DS_USERTableAdapters.USERMST_SELECTTableAdapter UAdapter =

new DS_USERTableAdapters.USERMST_SELECTTableAdapter();

UDT = UAdapter.SelectData();

GridView1.DataSource = UDT;

GridView1.DataBind();

}
```

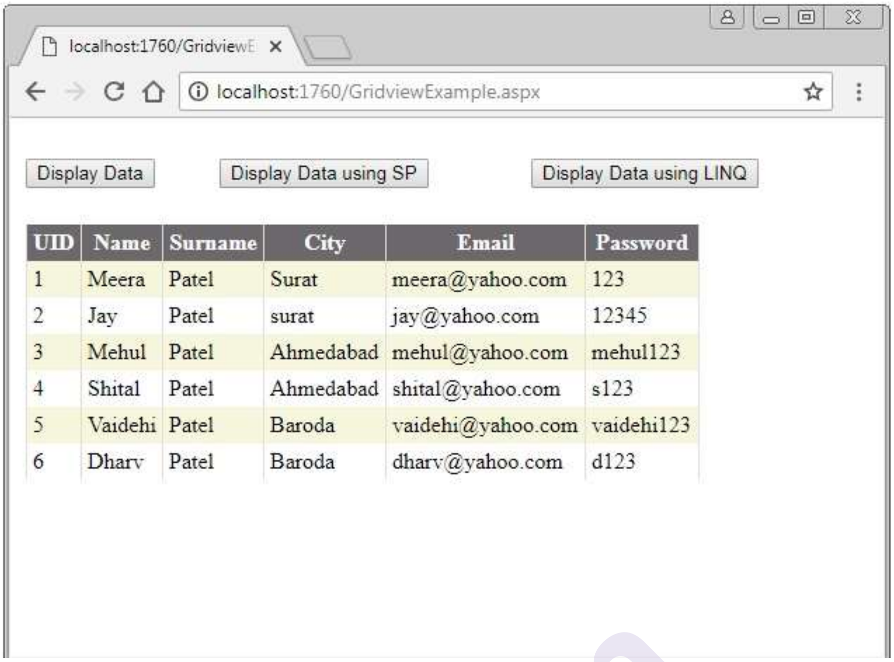
Step 5: Bind Data using LINQ method.

Write below code on ‘Display Data using LINQ’ button.

Here, we use LINQ method to fetch data from sql server and Display data in to GridView control.

```
protected void btnviewLINQ_Click(object sender, EventArgs e)
{
DataClassesDataContext Ctx = new DataClassesDataContext();
GridView1.DataSource = Ctx.USERMST_SELECT();
GridView1.DataBind();
}
```

Here is the result of above GridView example.



UID	Name	Surname	City	Email	Password
1	Meera	Patel	Surat	meera@yahoo.com	123
2	Jay	Patel	surat	jay@yahoo.com	12345
3	Mehul	Patel	Ahmedabad	mehul@yahoo.com	mehul123
4	Shital	Patel	Ahmedabad	shital@yahoo.com	s123
5	Vaidehi	Patel	Baroda	vaidehi@yahoo.com	vaidehi123
6	Dharv	Patel	Baroda	dharv@yahoo.com	d123

Full code of code behind page C# code.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Data.SqlClient;
using System.Data;
public partial class GridviewExample : System.Web.UI.Page
{
  protected void Page_Load(object sender, EventArgs e)
  {
  }
  protected void btnview_Click(object sender, EventArgs e)
  {
      SqlConnection SQLConn = new SqlConnection("Data Source=.\\SQLEXPRESS;Initial Catalog='Blog';Integrated Security=True");
      SqlDataAdapter SQLAdapter = new SqlDataAdapter("Select * from UserMst", SQLConn);
      DataTable DT = new DataTable();
      SQLAdapter.Fill(DT);
      GridView1.DataSource = DT;
```

```
        GridView1.DataBind();
    }
    protected void btnviewdataSP_Click(object sender, EventArgs e)
    {
        DS_USER.USERMST_SELECTDataTable     UDT     =     new
DS_USER.USERMST_SELECTDataTable();
        DS_USERTableAdapters.USERMST_SELECTTableAdapter
UAdapter                                  =                              new
DS_USERTableAdapters.USERMST_SELECTTableAdapter();
        UDT = UAdapter.SelectData();
        GridView1.DataSource = UDT;
        GridView1.DataBind();
    }
    protected void btnviewLINQ_Click(object sender, EventArgs e)
    {
        DataClassesDataContext Ctx = new DataClassesDataContext();
        GridView1.DataSource = Ctx.USERMST_SELECT();
        GridView1.DataBind();
    }
}
```

11.9 DETAILS VIEW :

DetailsView control used to display a single database record of table layout in ASP.Net. Means it works with a single data item at a time. DetailsView control used to display record, insert, update, and delete records from database.

DetailsView Control Syntax :

<asp:DetailsView ID=“DetailsView1“ runat=“server“></asp:DetailsView>

Now let’s take an asp.net example to bind data to DetailsView.

Here, we learn how to bind and display data in DetailsView control from sql server database in asp.net c#.

DetailsView Control Example in ASP.Net C#

Step 1 – Open the Visual Studio –> Create a new empty Web application.

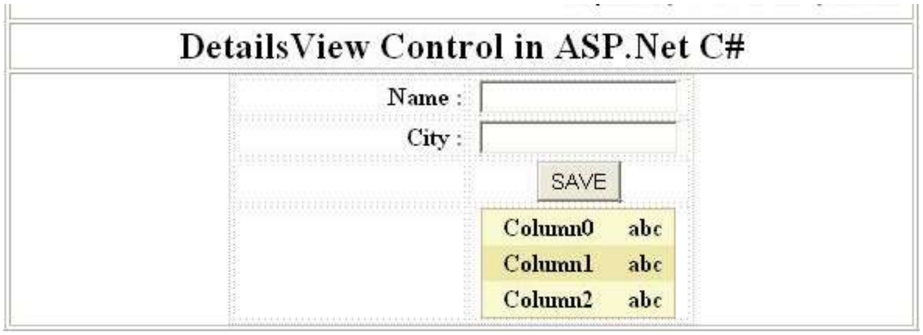
Step 2 – Create a New web page.

Step 3 – Drag and drop DetailsView Control on web page from toolbox.

Step 4 – Create New Database in SQL Server

Step 5 – Make connection between Database and web application.

Step 6 – Bind and Display data to DetailsView.



DetailsView Control in ASP.Net C#

Name :

City :

Column0	abc
Column1	abc
Column2	abc

Code:

```
<table>
<tr>
<td align="right">
Name :</td>
<td>
<asp:TextBox ID="txtname" runat="server"></asp:TextBox>
</td>
</tr>
<tr>
<td align="right">
City :</td>
<td>
<asp:TextBox ID="txtcity" runat="server"></asp:TextBox>
</td>
</tr>
<tr>
<td>
&nbsp;</td>
<td>
<asp:Button  ID="btnSave"  runat="server"  onclick="btnSave_Click"
Text="SAVE" />
</td>
</tr>
<tr>
<td>
&nbsp;</td>
<td>
```

```
<asp:DetailsView ID="DetailsView1" runat="server"
BackColor="LightGoldenrodYellow" BorderColor="Tan"
BorderWidth="1px" CellPadding="2" DataKeyNames="ID"
ForeColor="Black" GridLines="None" Height="50px"
onitemdeleting="DetailsView1_ItemDeleting"
onitemupdating="DetailsView1_ItemUpdating"
onmodechanging="DetailsView1_ModeChanging"
onpageindexchanging="DetailsView1_PageIndexChanging"
Width="125px">
<FooterStyle BackColor="Tan" />
<PagerStyle BackColor="PaleGoldenrod" ForeColor="DarkSlateBlue"
HorizontalAlign="Center" />
<HeaderStyle BackColor="Tan" Font-Bold="True" />
<EditRowStyle BackColor="DarkSlateBlue" ForeColor="GhostWhite"
/>
<AlternatingRowStyle BackColor="PaleGoldenrod" />
</asp:DetailsView>
</td>
</tr>
</table>
```

Now, After doing this next step to bind the DetailsView control from Database.

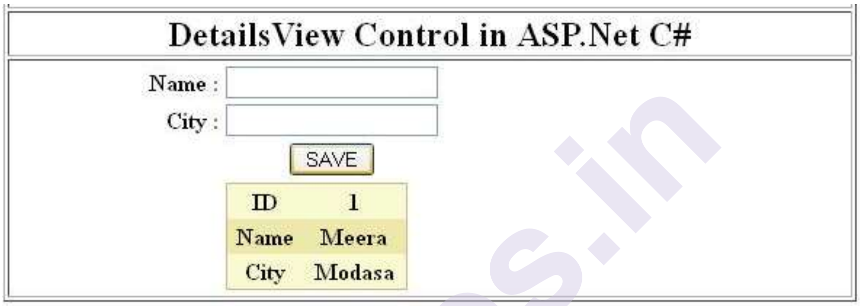
Here, we first insert record in to database and fetch record from database and display it to DetailsView control in ASP.Net.

Write below code on SAVE button for insert and bind data to DetailsView in ASP.net.

```
protected void btnSave_Click(object sender, EventArgs e)
{
SqlConnection SQLConn = new SqlConnection("Data
Source=COMPUTER_1\\SQLEXPRESS;Initial
Catalog=BOOK;Integrated Security=True");
SqlDataAdapter SQLAdapter = new SqlDataAdapter("insert into
UserMst values ('"+ txtname.Text +"','"+ txtcity.Text +"')", SQLConn);
DataTable DT = new DataTable();
SQLAdapter.Fill(DT);
BindDetailsView();
}
private void BindDetailsView()
{
```

```
SqlConnection SQLConn = new SqlConnection("Data Source=COMPUTER_1\\SQLEXPRESS;InitialCatalog=BOOK;Integrated Security=True");
SqlDataAdapter SQLAdapter = new SqlDataAdapter("select * from UserMst", SQLConn);
DataTable DT = new DataTable();
SQLAdapter.Fill(DT);
DetailsView1.DataSource = DT;
DetailsView1.DataBind();
}
```

The Output of DetailsView control example is :



DetailsView Control in ASP.Net C#

Name :

City :

ID	1
Name	Meera
City	Modasa

Note : If you use DetailsView control in your website, then it can display a single record at a time, so you must have do paging with DetailsView control for show all the records one by one.

11.10 FORM VIEW:

The Form View control is used to display a single record at a time from a data source. When you use the Form View control, you create templates to display and edit data-bound values. The templates contain controls, binding expressions, and formatting that define the look and functionality of the form.

Step 1 - Create database table in SQL server

```
CREATE TABLE [dbo].[StudentsResult](
   [RollNumber] [int] IDENTITY(10000,1) NOT NULL,
   [StudentName] [nvarchar](50) NULL,
   [Hindi] [int] NULL,
   [English] [int] NULL,
   [Physics] [int] NULL,
   [Chemistry] [int] NULL,
   [Biology] [int] NULL,
   [Mathematics] [int] NULL,
   [Total_Marks] [int] NULL,
   [Percentage] [float] NULL,
 CONSTRAINT [PK_StudentsResult] PRIMARY KEY CLUSTERED
(
   [RollNumber] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
CREATE PROCEDURE [dbo].[spGetStudentResult]
AS
BEGIN
SELECT*FROM [dbo].[StudentsResult]
END
```

Step 2: Open Visual Studio click on New Project and create an empty web application project.

Step 3 : Drag and drop a FormView control on the web form. Design FormView control.

Complete web form code:

```
<!DOCTYPE html>
  <html>
<head runat="server">
   <title>FormView</title>
   <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.1.0/css/bootstrap.min.
css">
   <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></s
cript>
   <script
src="https://maxcdn.bootstrapcdn.com/bootstrap/4.1.0/js/bootstrap.min.js"
></script>
</head>
<body>
   <form id="form1" runat="server">
      <div class="container py-4">
         <h5 class="text-center text-uppercase">FormView control in
asp.net</h5>
         <asp:FormView CssClass="container" ID="FormView1"
AllowPaging="true"OnPageIndexChanging="FormView1_PageIndexCha
nging" runat="server">
```

```
<ItemTemplate>
    <table class="table table-bordered table-striped">
        <tr>
            <td>Roll Number</td>
            <td><%#Eval("RollNumber") %></td>
        </tr>
        <tr>
            <td>Student Name</td>
            <td><%#Eval("StudentName") %></td>
        </tr>
        <tr>
            <td>Hindi</td>
            <td><%#Eval("Hindi") %></td>
        </tr>
        <tr>
            <td>English</td>
            <td><%#Eval("English") %></td>
        </tr>
        <tr>
            <td>Physics</td>
            <td><%#Eval("Physics") %></td>
        </tr>
        <tr>
            <td>Chemistry</td>
            <td><%#Eval("Chemistry") %></td>
        </tr>
        <tr>
            <td>Biology</td>
            <td><%#Eval("Biology") %></td>
        </tr>
        <tr>
            <td>Mathematics</td>
            <td><%#Eval("Mathematics") %></td>
        </tr>
        <tr>
            <td>Total Marks</td>
            <td><%#Eval("Total_Marks") %></td>
        </tr>
        <tr>
```

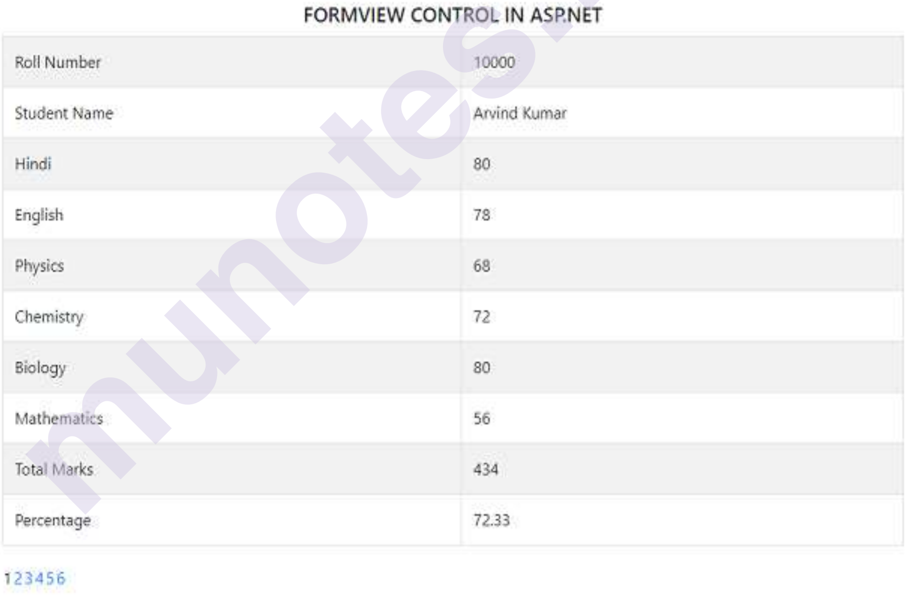
```
                    <td>Percentage</td>
                    <td><%#Eval("Percentage")%></td>
                  </tr>
                </table>
              </ItemTemplate>
            </asp:FormView>
          </div>
      </form>
</body>
</html>
```

Step 4 : Right click and view web form code. Write the following code.

```
using System;
using System.Configuration;
using System.Data;
using System.Data.SqlClient;
namespace BindDataControl_Demo
{
    public partial class FormView : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            if (!IsPostBack)
            {
                BindFormView();
            }
        }

        private void BindFormView()
        {
            string                                CS                                =
ConfigurationManager.ConnectionStrings["DBCS"].ConnectionString;
            using (SqlConnection con = new SqlConnection(CS))
            {
        SqlCommand cmd = new SqlCommand("spGetStudentResult", con);
                cmd.CommandType = CommandType.StoredProcedure;
                SqlDataAdapter sda = new SqlDataAdapter(cmd);
                con.Open();
                DataTable dt = new DataTable();
                sda.Fill(dt);
            FormView1.DataSource = dt;
            FormView1.DataBind();
          }
        }
 protected void FormView1_PageIndexChanging(object sender, System.Web.UI.WebControls.FormViewPageEventArgs e)
        {
          FormView1.PageIndex = e.NewPageIndex;
          this.BindFormView();
        }
    }
}
```

Step 5: Run the project ctrl+F5

Screenshot:



FORMVIEW CONTROL IN ASP.NET

Roll Number	10000
Student Name	Arvind Kumar
Hindi	80
English	78
Physics	68
Chemistry	72
Biology	80
Mathematics	56
Total Marks	434
Percentage	72.33

123456

11.10 SUMMARY:

- GridView can used for binding to data source controls, such as SqlDataSource.
- It has Built-in sort, update and delete, paging, row selection capabilities.
- BoundField column provides a DataFormatString property that you can use to configure the numbers and dates using a format string.
- Use TemplateFields when we wish to display ASP.Net controls in a GridView column.

- DetailsView control used to display a single database record of table.
- The FormView control is used to display a single record at a time from a data source.

11.11 REFERENCES :

Reference Books:

- Beginning ASP.NET 4.5 in C# by Apress.
- Murach's ASP.NET 4.6 Web Programming in C#2015 by SPD
- ASP.NET 4.0 programming by Tata McGrawHill.
- Programming ASP.NET by Microsoft Press

Web References:

- https://www.c-sharpcorner.com/
- https://www.aspsnippets.com/
- http://asp.net-informations.com/

11.12 UNIT END EXERCISES :

i. Explain the GridView control in detail.

ii. Explain any five properties of GridView control.

iii. What are different style properties of GridView control?

iv. How to sort the data with GridView control?

v. How to include paging in GridView control?

vi. What is used of CommandField in GridView control?

vii. What are the different types of Template in GridView control?

viii. How to use DetailView control?

ix. How to display data in FormView control?

❄❄❄❄❄❄❄

12

EXTENSIBLE MARKUP LANGUAGE (XML)

Unit Structure

12.0 OVERVIEW

In this unit you will be able to learn some concepts about XML. A few of the concepts are:

- What is XML and how it is important
- What are the classes in XML and why they are important?
- Different types of validations in XML
- How to display XML

The XML Security standards include XML Digital Signature for integrity and signing solutions, XML Encryption for confidentiality, XML Key Management (XKMS) for public key registration, location and validation, Security Assertion Markup Language (SAML) for conveying authentication, authorization and attribute assertions, XML Access Control Markup Language (XACML) for defining access control rules, and Platform for Privacy Preferences (P3P) for defining privacy policies and preferences. Major use cases include securing Web Services (WS-Security) and Digital Rights Management (eXtensible Rights Markup Language 2.0 – XrML).

These days, most of the web applications are using AJAX concepts to create better and more responsive applications. AJAX reduces the traffic between client and server and also, makes the response time faster which directly increases the performance of an application.

12.1 XML EXPLAINED

XML is a software- and hardware-independent tool for storing and transporting data. It is stands for e**X**tensible **M**arkup **L**anguage. In simple words it's a markup language same as your HTML. The tags in XML are used to store and organize the data rather than how to display the data. In HTML you can see tags are used to display the data and with the help of CSS, you are stylizing the data the way you want. There is no comparison between HTML and XML and neither the party is going to replace them, but one can adopt many successful features of HTML.

XML is a markup language that defines set of rules for encoding documents in a format that is both human-readable and machine-readable. In simple words, a markup language is a set of symbols that can be placed in the text of a document to demarcate and label the parts of that document. It also uses Document Type Definitions (DTD) to define the XML document structure. Unlike HTML tags, XML tags are self-descriptive. It's an open format. The filename extension of XML document is .xml.

Below is simple example how XML markup looks:

```
<message>

        <text>Hello, world!</text>

</message>
```

In the above example, <message> and <text> are tags. Here, tags <message> and </message> mark the start and end of the XML code fragment. The tags <text> and </text> surround the text Hello, world!

Let's dive into some of the key features of XML:

- It is extensible and human-readable.
- It is platform and language independent.
- It preserves white space.
- Overall simplicity.
- Self-descriptive nature.
- It separates data from HTML.
- XML tags are not predefined. You need to define your customized tags.
- XML was designed to carry data, not to display that data.
- Mark-up code of XML is easy to understand for a human.

- Well-structured format is easy to read and write from programs.
- XML is an extensible markup language like HTML.

12.2 THE XML CLASSES

The XML file may be loaded at the beginning of the program or at a later stage. In the same way as you can load a model from a file, you can also load the corresponding data from a file. The example below shows how to describe the dynamic classes Event and Alarm in XML format. These classes will be created when the XML file is loaded. The Event class has the ID attribute of type string. The Alarm class is a subclass of the Event class that has two attributes, PerceivedSeverity, of type String, and Acknowledged, a Boolean attribute that defaults to false. The Alarm class inherits the ID attribute from the Event class.

```
<classes xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"

xsi:noNamespaceSchemaLocation = "ilog/cpl/schema/model.xsd">

<classes>

 <class>

  <name>Event</name>

  <attribute>

   <name>ID</name>

   <javaClass>java.lang.String</javaClass>

  </attribute>

 </class>

 <class>

  <name>Alarm</name>

  <superClass>Event</superClass>

  <attribute>

   <name>PerceivedSeverity</name>

   <javaClass>java.lang.String</javaClass>

  </attribute>

  <attribute>

   <name>Acknowledged</name>

   <javaClass>java.lang.Boolean</javaClass>
```

```
      <defaultValue>false</defaultValue>
    </attribute>
  </class>
</classes>
```

The dynamic classes created from an XML file can inherit from existing classes.

12.3 XML VALIDATION

An XML document is referred as ‘well formed’ if it has a correct syntax. Below are some of the points that validates an XML document:

i. It must have a root element

ii. It should contain a closing tag

iii. XML tags are case sensitive

iv. Attribute values must be quoted

v. XML elements must be properly nested

If your XML document contains error, it will stop your XML applications. Unlike HTML documents, XML will display errors in browser if it contains error. In HTML, browsers will display HTML documents with error (like missing end tags). In another words, with XML errors are not allowed.

There are numbers of XML validators available online. One can test their XML markup using one of these validators.

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
<to>John</to>
<from>Loffer</from>
<heading>Reminder Note</pheading>
<body>We are catching tomorrow</body>
</note>
```

In the above example, line no. 5 contains error. Closing tag should be </heading> instead of </pheading>. This closing tag is mismatching with its opening tag.

A "well formed" XML document is not the same as a "valid" XML document.

A "valid" XML document must be well formed. In addition, it must conform to a document type definition.

If an XML document is well-formed and has an associated Document Type Declaration (DTD), then it is said to be a valid XML document. We will study more about DTD in the chapter XML - DTDs

12.4 XML DISPLAY AND TRANSFORMS

A raw XML file can be viewed in all major browsers. You cannot expect XML files to be displayed as HTML pages. Most browsers will display an XML document with color-coded elements. It is rather displayed in a proper hierarchy with a plus (+) or minus (-) to the left of the elements. It can be clicked to expand or collapse the element structure. To view raw XML source, try to select "View Page Source" or "View Source" from the browser menu.

- XSLT stands for Extensible Stylesheet Language Transformation.
- XSLT is used to transform XML document from one form to another form.
- XSLT uses Xpath to perform matching of nodes to perform these transformation.
- The result of applying XSLT to XML document could be an another XML document, HTML, text or any another document from technology perspective.
- The XSL code is written within the XML document with the extension of (.xsl).
- In other words, an XSLT document is a different kind of XML document.

Below is the example explaining XML Transformation

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

<xsl:stylesheet version="1.0"

xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">

<html>

<body>

<h1 align="center">Students' Basic Details</h1>

<table border="3" align="center" >

<tr>

        <th>Name</th>
        <th>Branch</th>
        <th>Age</th>
        <th>City</th>
</tr>
        <xsl:for-each select="student/s">
<tr>
        <td><xsl:value-of select="name"/></td>
        <td><xsl:value-of select="branch"/></td>
        <td><xsl:value-of select="age"/></td>
        <td><xsl:value-of select="city"/></td>
</tr>
        </xsl:for-each>
        </table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

12.5 UNDERSTANDING SECURITY REQUIREMENTS

Security has always been vitally important in the business world to ensure the integrity of content and transactions, to maintain privacy and confidentiality, and to make sure information is used appropriately. However, in today’s web-based business environment, the means for providing that security have changed. Using physical security, no longer works as well as it did in the past when all the computing resources were locked in a central computing room with all jobs submitted locally.

An essential requirement of new security standards is that they work naturally with content created using eXtensible Markup Language (XML). XML is being adopted widely for a growing variety of applications and types of content. It is also forming the basis for distributed system protocols to integrate applications across the Internet, such as Web Services protocols. XML languages are text based and designed to be extended and combined. It should be natural to provide integrity, confidentiality and other security benefits to entire XML documents or portions of these documents in a way that does not prevent further processing by standard XML tools [XMLRef]. XML Security therefore must be integrated with XML in such a way as to maintain the advantages and capabilities of XML while adding necessary security capabilities. This is especially important in XML-based protocols, such as XML Protocol (XMLProt, Simple Object Access Protocol, SOAP), that are explicitly designed to allow intermediary processing and modification of messages.

Older security technologies provide a set of core security algorithms and technologies that can be used in XML Security, but the actual formats used to implement security requirements are inappropriate for most XML Security applications. One reason is that these standards use binary formats that require specialized software for interpretation and use, even for extracting portions of the security information. A second reason is that these standards are not designed for use with XML and do not support common XML technical approaches for managing content, such as specifying content with uniform resource identifier strings (URIs) or using other XML standard definitions for locating portions of XML content (like XPath [XPath]). In addition, some existing security technologies assume that security-specific software will be integrated with applications to enable security. In practice, this is not always the case due to the details of custom integration.

XML Security addresses these issues by defining a common framework and processing rules that can be shared across applications using common tools, avoiding the need for extensive customization of applications to add security. XML Security reuses the concepts, algorithms and core technologies of legacy security systems while introducing changes necessary to support extensible integration with XML. This allows interoperability with a wide range of existing infrastructures and across deployments.

XML Security reduces barriers to adoption by defining the minimum modular mechanisms to obtain powerful results. By employing existing technologies and enabling use of XML paradigms and tools, XML Security minimizes the need to modify applications to meet security requirements.

12.6 AUTHENTICATION AND AUTHORIZATION

Authentication is the process of verifying that an individual, entity or website is whom it claims to be. Authentication in the context of web applications is commonly performed by submitting a username or ID and one or more items of private information that only a given user should know.

Session Management is a process by which a server maintains the state of an entity interacting with it. This is required for a server to remember how to react to subsequent requests throughout a transaction. Sessions are maintained on the server by a session identifier which can be passed back and forward between the client and server when transmitting and receiving requests. Sessions should be unique per user and computationally very difficult to predict. The Session Management Cheat Sheet contains further guidance on the best practices in this area.

Let's dive into the SAML which is used in the XML authentication.

What is SAML?

SAML is an acronym used to describe the Security Assertion Markup Language (SAML). Its primary role in online security is that it enables you to access multiple web applications using one set of login credentials. It works by passing authentication information in a particular format between two parties, usually an identity provider (idP) and a web application.

What SAML is and how it works

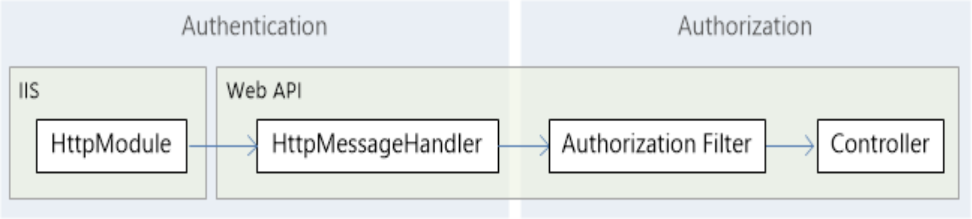
SAML is an open standard used for authentication. Based upon the Extensible Markup Language (XML) format, web applications use SAML to transfer authentication data between two parties - the identity provider (IdP) and the service provider (SP).

The technology industry created SAML to simplify the authentication process where users needed to access multiple, independent web applications across domains. Prior to SAML, single sign-on (SSO) was achievable but relied on cookies that were only viable within the same domain. It achieves this objective by centralizing user authentication with an identity provider. Web applications can then leverage SAML via the identity provider to grant access to their users. This SAML authentication approach means users do not need to remember multiple usernames and passwords. It also benefits service providers as it increases security of their own platform, primarily by avoiding the need to store (often weak and insecure) passwords and not having to address forgotten password issues.

Forms Authentication

Web App assumes that authentication happens in the host, such as IIS, which uses HTTP modules for authentication. One can configure the project to use any of the authentication modules built into IIS or ASP.NET, or write your own HTTP module to perform custom authentication.

When the host authenticates the user, it creates a principal, which is an IPrincipal object that represents the security context under which code is running. For example, Web API authentication and authorization the process could be like this:



Authentication Types:

- Windows Authentication
- Forms Authentication
- Passport Authentication
- None

- **Windows Authentication**, IIS performs the authentication, and the authenticated token is forwarded to the ASP.NET worker process.

- **Forms Authentication**: authenticates the user by inspecting the forms authentication ticket, which is typically included in the user's cookies collection. If no form of authentication ticket is present, the user is anonymous.

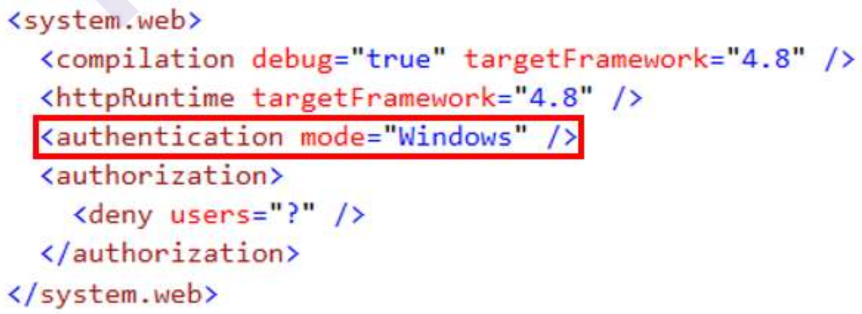
- **Passport Authentication**: Centralized authentication service provided by Microsoft that offers single logon and core profile services for member sites.

- **None**: No Authentication provided. This is the default Authentication mode.

12.7 WINDOWS AUTHENTICATION

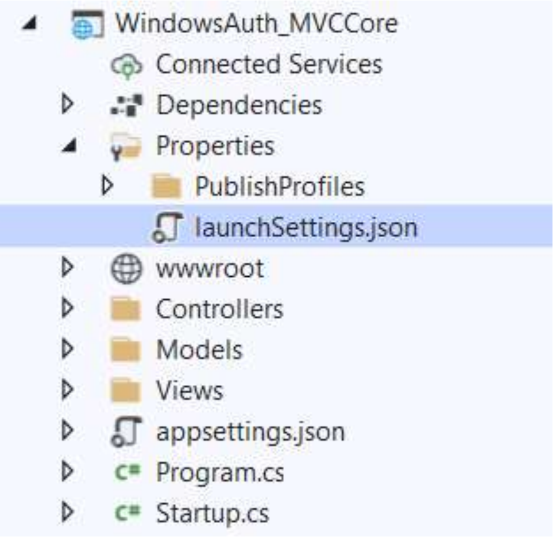
In Windows authentication, IIS performs the authentication, and the authenticated token is forwarded to the ASP.NET worker process. The advantage of using Windows authentication is that it requires minimal coding. One may want to use Windows authentication to impersonate the Windows user account that IIS authenticates beforehand off the request to ASP.NET.

The following is ASP.NET app Windows Authentication configuration in **web.config** file:

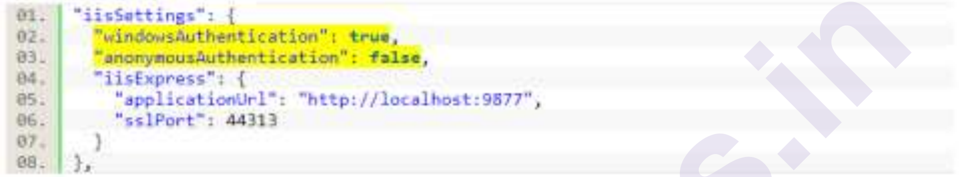


```
<system.web>
  <compilation debug="true" targetFramework="4.8" />
  <httpRuntime targetFramework="4.8" />
  <authentication mode="Windows" />
  <authorization>
    <deny users="?" />
  </authorization>
</system.web>
```

For .NET Core, configuration is in the launchSettings.json file under Profiles folder:



Such as,



```
"iisSettings": {
  "windowsAuthentication": true,
  "anonymousAuthentication": false,
  "iisExpress": {
    "applicationUrl": "http://localhost:9877",
    "sslPort": 44313
  }
},
```

12.8 UNDERSTANDING AJAX

What is AJAX?

Asynchronous JavaScript and XML (AJAX) is a development technique used to create interactive web applications or rich internet applications. AJAX uses a number of existing technologies together, including: XHTML, CSS, JavaScript, Document Object Model, XML, XSLT, and the XML Http Request object.

With AJAX, web applications can retrieve data from the server asynchronously, in the background, without reloading the entire browser page. The use of AJAX has led to an increase in interactive animation on web pages.

Advantages

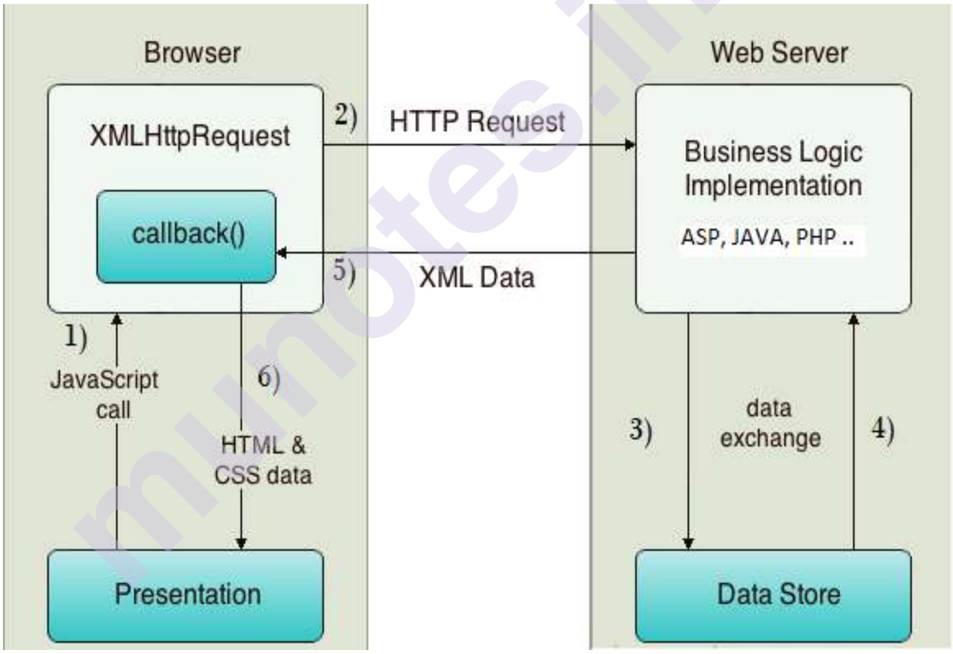
- Reduces the traffic travels between the client and the server.
- No cross browser pain.
- Better interactivity and responsiveness.
- With AJAX, several multipurpose applications and features can be handled using a single web page(SPA).
- API's are good because those work with HTTP method and JavaScrtipt.

Disadvantages

- Search engines like Google would not be able to index an AJAX application.
- It is totally built-in JavaScript code. If any user disables JS in the browser, it won't work.
- The server information cannot be accessed within AJAX.
- Security is less in AJAX applications as all the files are downloaded at client side.
- The data of all requests is URL-encoded, which increases the size of the request.

How AJAX works

AJAX communicates with the web server using XMLHttpRequest object. The following diagram illustrates how AJAX communicates between Client and Server.



12.9 USING PARTIAL REFRESHES

As name suggests, it gives a flexibility to web application to refresh the objects like tables, divisions or sections without refreshing the whole page. For an example, if your web application contains a table with several records, and you want to delete a specific record from it. Upon deleting a specific record, it gets partially refreshed the table and the database without refreshing the whole page. That’s the beauty of AJAX.

Let’s check below example:

Partial View Code:

```
<div id="wrapper">
@model IEnumerable<MyAppModels.StudentModel>
    <table class="table">
    <tr>
        <th>
            @Html.DisplayNameFor(model => model.FName)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.LName)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.Email)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.address.Details)
 </th>
        <th>
            @Html.DisplayNameFor(model => model.address.Country)
</th>
        <th>
            @Html.DisplayNameFor(model => model.address.State)
        </th>
        <th></th>
    </tr>
    @foreach (var item in Model)
    {
        <tr>
            <td>
```

XML

```
                <input type="hidden" name="stid" id="stid" value="@item.Id" />
                @Html.DisplayFor(modelItem => item.FName)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.LName)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Email)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.address.Details)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.address.Country)
        </td>
            <td>
                @Html.DisplayFor(modelItem => item.address.State)
            </td>
            <td>
          <button id="btnDelete" data-Id="@item.Id">Delete</button>
              <button id="btnEdit" data-Id="@item.Id">Edit</button>
        <button id="btnDetails" data-Id="@item.Id">Details</button>
            </td>
          </tr>
    }
</table>
</div>
<div id="EditArea"></div>
<div id="DetailArea"></div>
```

Controller Code:

```
public ActionResult GetStudents()
{
var res = stude.GettAllStudents();
return View(res);
}
```

Ajax/JS Code

```
$("#Edit").click(function () {
            var id = $(this).attr("data-Id");
            $.ajax({
                method: "POST",
                url: "Home/Edit/" + id,
                contentType: "application/json; charset=utf-8",
                cache: false,
               success: function (data) {
                   $("#EditArea").html(data);
            },
                error: function (err) {
                    console.log('Failed to get data' + err);
           }
            });
     });
```

Edit Action:

```
public ActionResult Edit(int id)
{
var res = stude.GettAllStudents().Where(x=>x.Id==id);
return partiaView(res);
}
```

Partial View for Edit:

```
 @model MyAppModels.StudentModel
<form action="/home/update">
Name:<input type="text" value="@model.Name">
.......other fields.....
<button type="submit">Update<button>
<form>
```

Ajax for Delete:

```
$("#Delete").click(function () {
          var id = $(this).attr("data-Id");
           $.ajax({
                 method: "POST",
                 url: "Home/Delete/" + id,
                 contentType: "application/json; charset=utf-8",
                 cache: false,
                success: function (data) {
                    $("#wrapper").Html(data);
              },
                 error: function (err) {
                    console.log('Failed to get data' + err);
             }
              });
     });
```

Action for Delete:

```
public ActionResult Delete(int id)
{
var Student= stude.GettAllStudents().Where(x=>x.Id==id);
_context.Students.Remove(Student);
_context.SaveChanges();
return View("GetStudents",res);
}
```

When you clicked the delete link, it deletes the record and loads the View as you want but the issue is that When you clicked on another record then it shows an error" record not found" and this is because of the view overload on the First view instead of refreshing it. You don't know how to refresh it, instead of overload. From overload, you mean that the ajax code load a Partial view on the already loaded view, and because of that the error occurs and stays until you refresh the page.

12.10 USING PROGRESS NOTIFICATION

ASP.NET includes another control that can help—the UpdateProgress control. The UpdateProgress control works in conjunction with the UpdatePanel. Essentially, the UpdateProgress control allows you to show a message while a time-consuming update is under way.

The UpdateProgress control is slightly misnamed. It doesn’t actually indicate progress; instead, it provides a wait message that reassures the user that the page is still working and the last request is still being processed.

When you add the UpdateProgress control to a page, you get the ability to specify some content that will appear as soon as an asynchronous request is started and disappear as soon as the request is finished. This content can include a fixed message, but many people prefer to use an animated GIF, because it more clearly suggests that the page is still at work. Often, this animated GIF simulates a progress bar.

The markup for this page defines an UpdatePanel followed by an UpdateProgress:

```
<asp:UpdatePanel ID="UpdatePanel1" runat="server">
  <ContentTemplate>
    <div style="background-color:#FFFFE0;padding: 20px">
      <asp:Label ID="lblTime" runat="server" Font-Bold="True"></asp:Label>
      <br /><br />
      <asp:Button ID="cmdRefreshTime" runat="server"
       Text="Start the Refresh Process" />
    </div>
  </ContentTemplate>
</asp:UpdatePanel>
<br />

<asp:UpdateProgress ID="updateProgress1" runat="server">
```

```
  <ProgressTemplate>
    <div style="font-size: xx-small">
      Contacting Server ... <img src="wait.gif" alt="Waiting..." />
    </div>
  </ProgressTemplate>
</asp:UpdateProgress>
```

12.11 IMPLEMENTING TIMED REFRESHES

Using the two controls you've seen so far—the UpdatePanel and UpdateProgress controls—you can create self-contained regions on your page that refresh themselves when certain actions take place. Of course, in order for this technique to work, the user needs to initiate an action that would ordinarily cause a postback, such as clicking a button, selecting an item in an AutoPostBack list, checking an AutoBostBack check box, and so on.

In some situations, you might want to force a full or partial page refresh without waiting for a user action. For example, you might create a page that includes a stock ticker, and you might want to refresh this ticker periodically (say, every five minutes) to ensure it doesn't become drastically outdated. ASP.NET includes a Timer control that allows you to implement this design easily.

The Timer control is refreshingly straightforward. You simply add it to a page and set its Interval property to the maximum number of milliseconds that should elapse before an update. For example, if you set Interval to 60000, the timer will force a postback after one minute elapses.

To use the timer with partial rendering, wrap the updateable portions of the page in UpdatePanel controls with the UpdateMode property set to Conditional. Then, add a trigger that forces the UpdatePanel to update itself whenever the Timer.Tick event occurs. Here's the markup you need:

```
<asp:UpdatePanel ID="UpdatePanel1" runat="server" UpdateMode="Conditional">
  <ContentTemplate>
    ...
  </ContentTemplate>
  <Triggers>
    <asp:AsyncPostBackTrigger ControlID="Timer1" EventName="Tick" />
  </Triggers>
</asp:UpdatePanel>
```

12.12 WORKING WITH THE ASP.NET AJAX CONTROL TOOLKIT

Working with the ASP.NET AJAX Control Toolkit

Ajax Control Toolkit is an open source library for web development. The ASP.net Ajax Control toolkit contains highly rich web development controls for creating responsive and interactive AJAX enabled web applications. ASP.Net Ajax Control Toolkit contains 40 + ready controls which is easy to use for fast productivity. Controls are available in the Visual Studio Toolbox for easy drag and drop integration with your web application. Some of the controls are like AutoComplete, Color Picker, Calendar, Watermark, Modal Popup Extender, Slideshow Extender and more of the useful controls.

14.6 Summary:

Extensible Markup Language (XML) is a markup language used to describe the content and structure of data in a document. It is a simplified version of Standard Generalized Markup Language (SGML). XML is an industry standard for delivering content on the Internet. Because it provides a facility to define new tags, XML is also extensible. XSLT is used to transform XML into other formats, such as HTML. XMLHttpRequest object provides a page load after communicating with the server the way.

XML security refers to standard security requirements of XML documents such as confidentiality, integrity, message authentication, and non-repudiation. The need for digital signature and encryption standards for XML documents prompted the World Wide Web Consortium (W3C) to put forth an XML Signature standard and an XML Encryption standard. Using security fundamentals, one can validate and authenticate various applications.

Asynchronous JavaScript and XML (AJAX) is a development technique used to create interactive web applications or rich internet applications. AJAX uses a number of existing technologies together, including: XHTML, CSS, JavaScript, Document Object Model, XML, XSLT, and the XMLHttpRequest object. With AJAX, web applications can retrieve data from the server asynchronously, in the background, without reloading the entire browser page. The use of AJAX has led to an increase in interactive animation on web pages. Using AJAX one can create simple to complex web applications.

12.13 QUESTIONS:

- How XML is different from HTML?
- What is a valid XML document?
- What is the XML data binding?

- What is XML Namespace?
- What is authentication and authorization in XML?
- What are the different types of authentication in XML?
- What is XPath in XML?
- Why security is essential?
- What are different types of authentication?
- What is SAML?
- What is ASP.NET AJAX?
- Which is the current version of ASP.NET AJAX Control Toolkit?
- What is the use of partial refreshes in ASP.NET using AJAX?
- What is the role of UpdatePanel in ASP.NET AJAX?
- What are the limitations of AJAX?

12.14 BIBLIOGRAPHY:

- Javapoint – https://www.javatpoint.com/xml-tutorial
- Tutorialspoint - https://www.tutorialspoint.com/xml/index.htm
- C-SharpCorner - https://www.c-sharpcorner.com/article/ajax-in-asp-net/

❄❄❄❄❄❄❄