UNIT I

Introduction to .NET Framework

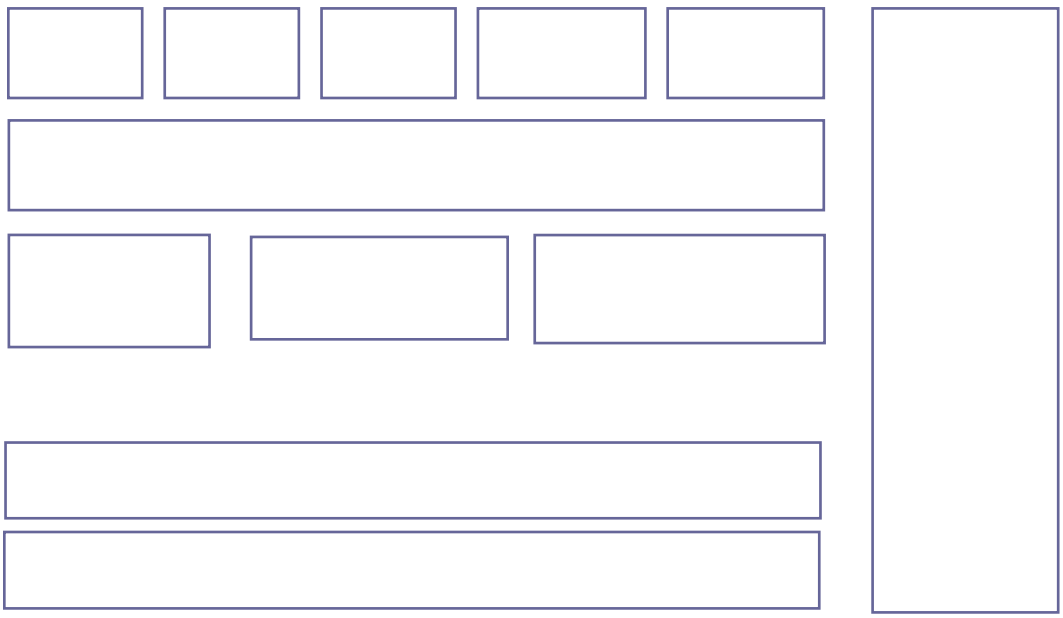
.NET is a software framework which is designed and developed by Microsoft. The first version of .Net framework was 1.0 which came in the year 2002. In easy words, it is a virtual machine for compiling and executing programs written in different languages like C#,VB.Net,etc.

It is used to develop Form-based applications, Web-based applications, and Web services. There is a variety of programming languages available on the .Net platform, VB.Net and C# being the most common ones are . It is used to build applications for Windows, phone, web etc. It provides a lot of functionalities and also supports industry standards.

.NET Framework supports more than 60 programming languages in which 11 programming languages are designed and developed by Microsoft. The remaining **Non-Microsoft Languages** which are supported by .NET Framework but not designed and developed by Microsoft.

Programming Languages which are designed and developed by Microsoft are:

C#.NET,VB.NET,C++.NET,J#.NET,F#.NET,JSCRIPT.NET,WINDOWS POWERSHELL



.NET is tiered, modular, and hierarchal. Each tier of the .NET Framework is a layer of abstraction. .NET languages are the top tier and the most abstracted level. The common language runtime is the bottom tier, the least abstracted, and closest to the native environment. This is important since the common language runtime works closely with the operating environment to manage .NET applications. The .NET Framework is partitioned into modules, each with its own distinct responsibility.

Common Language Specification

Common Type System

Every programming language has its own data type system, so CTS is responsible for the understanding all the data type system of .NET programming languages and converting them into CLR understandable format which will be a common format.

There are 2 Types of CTS that every .NET programming language have :

a. **Value Types:** Value Types will directly store the value directly into the memory location. These types work with stack mechanism only. CLR allots memory for these at Compile Time.
b. **Reference Types:** Reference Types will contain a memory address of value because the reference types won't store the variable value directly in memory. These types work with Heap mechanism. CLR allots memory for these at Runtime.

The following is the code for declaring an integer in C# and Visual Basic .NET. Either syntax maps to a System.Int32 object.

```
In C#integer
    int nVar=0;
In VB..NET
 dim nVar as integer=0
```

ASP.NET

It is a web application framework developed by Microsoft to build dynamic websites,web application and web services.

It is the successor to ASP. While IIS 5 and 6 support side-by-side execution of ASP and ASP.NET, ASP.NET is not merely an upgrade of ASP, as evidenced by the lack of upward compatibility.

Web forms are main building block for web applications and are contained in files with an .aspx extension.

Web services are components on a web server that a client application can call by making HTTP requests across the web.

WINDOWS FORMS:

Windows forms are a graphical user interface(GUI) class library which is bundled in .net framework.

The main purpose of this is to provide an easier interface to develop the applications for desktop.

ADO.NET:

It provides consistent access to data sources such as Microsoft SQL Server and data sources exposed through OLEDB and XML.

Application can use ADO.NET to connect to these data sources and retrieve ,manipulate and update data.

BASE CLASS LIBRARY:

It is a standard library available to all languages using .net framework.

It encapsulates a large number of common functions such as file reading,file

writing,database interaction and etc.,

BCL is a part of FCL (Framework Class Library) and FCL contains Microsoft.CSharp, Microsoft.VisualBasic libraries.

Example for BCL in C# languages are System is for console application,System.windows.forms is for windows application and System.web is for web application .

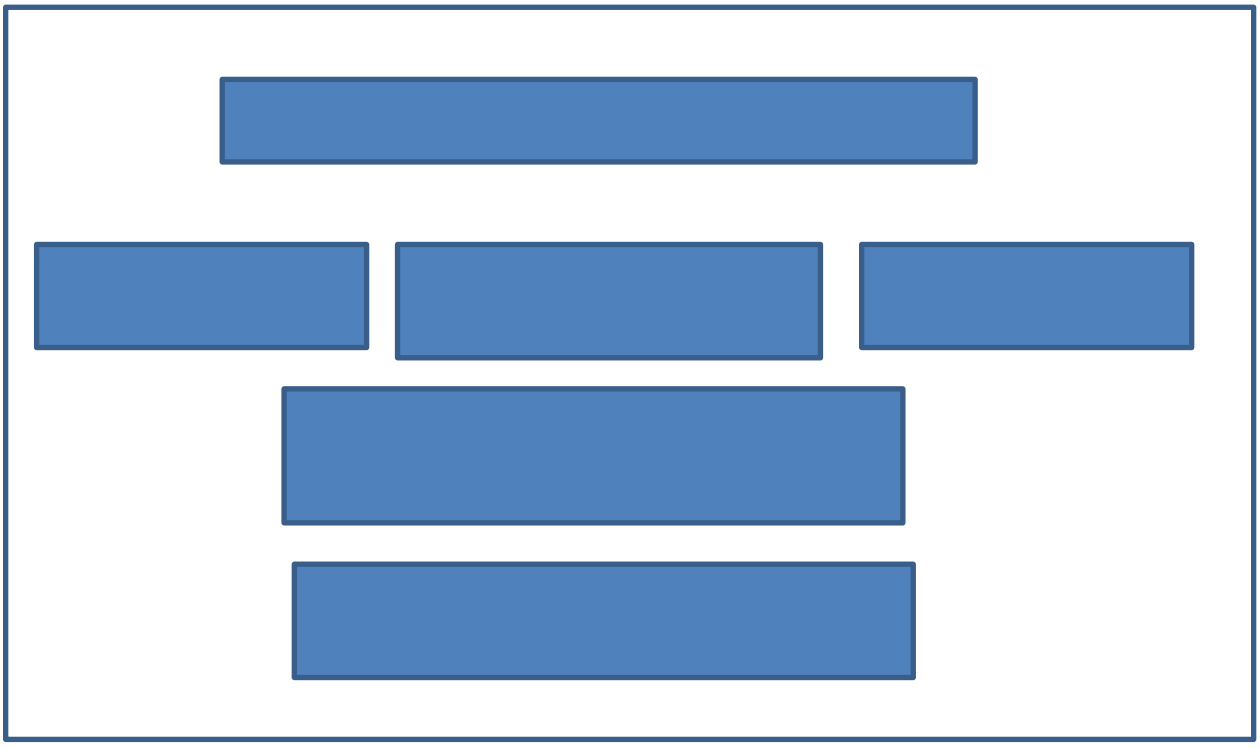
CLR:

CLR is the basic and Virtual Machine component of the **.NET Framework**.

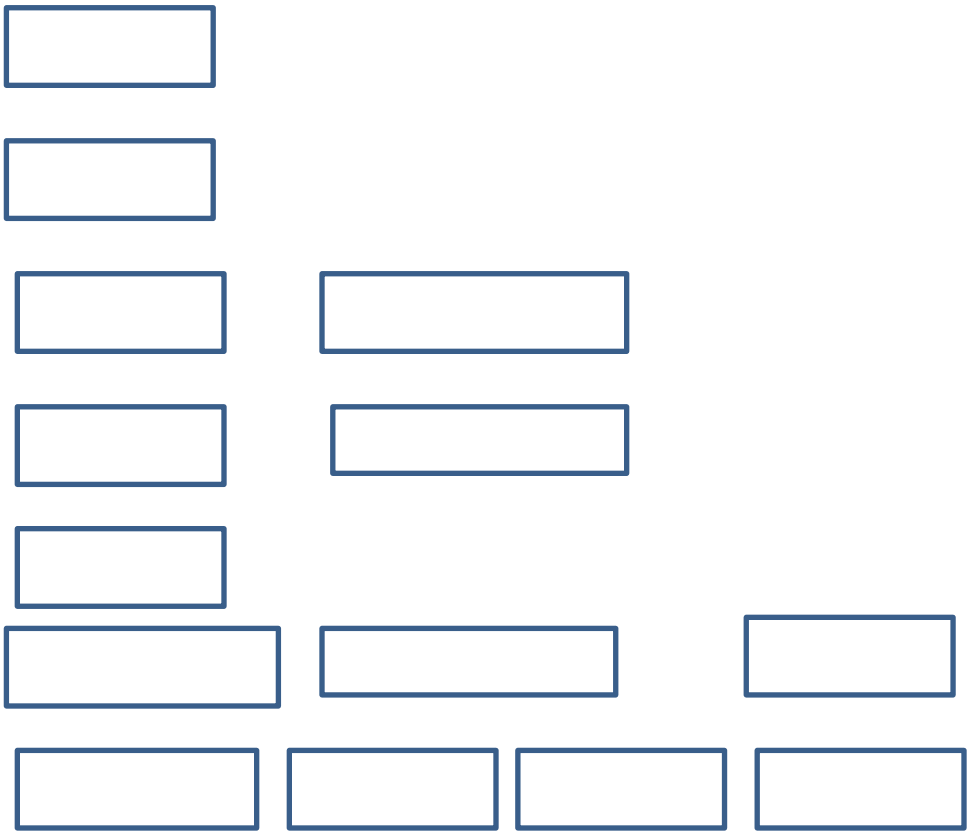
It is the **run-time enviornment in the .NET Framework** that runs the codes and helps in making the development process easier by providing the various services.

Basically, it is responsible for managing the execution of *.NET programs* regardless of any *.NET* programming language.

CLR Architecture:



- It is the life line of .net framework.
- **Runtime** is an environment in which programs are executed.
- The CLR is an environment in which user can run our .net applications that have been compiled to IL.
- The above diagram shows various components of the CLR.
- CTS -**Common Type System** is responsible for interpreting the data types into a common format.
- The second component ,the **IL compiler** takes in the IL code and converts it to the host machine language.
- The **execution support** is similar to the language runtime (DLL file/EXE file).
- **Security** component in the CLR ensures that the assembly (Ready to execute) has permissions to execute certain functions.
- **Garbage Collector** is used to provide the *Automatic Memory Management* feature. Suppose if there is no garbage collector then programmers have to write the memory management codes which will be a kind of overhead on programmers.
- **Class Loader** component is to load the classes needed by the executing application.(ex.System,Sytem.windows.forms etc.,)



The above diagram shows how the source code is executed in CLR component.

The source code is compiled to IL code while the meta data engine creates meta data information.

IL& Meta Data are linked with other native code if required and resultant IL code is saved.

During execution, the IL code and any requirement from the base class library are brought together by the class loader.

The combined code is tested for type safety and then compiled by JIT compiler to produce native machine code, which is sent to the runtime manager for execution.

Microsoft Intermediate Language (MSIL):

- A dot net programming language (C#,VB.net,VC++)does not compile into executable code instead it compiles into IL called MSIL.
- Programmer need not worry about the syntax of MSIL, because source code can be automatically converted to MSIL.
- The MSIL code is then send to the CLR that converts the code to Machine Language which is then run on the host machine.

Meta Data

- Meta data is binary information describing user program that is stored either in a CLR portable executable file (PE) or in memory.
- When user compile the code into a PE file, meta data is inserted into one portion of the file,while the user code is converted to MSIL and inserted into another portion of the file.
- Every type and member defined in a module or assembly is described within meta data.
- When code is executed ,the runtime loads meta data into memory and references it to discover information about classes, members and etc.,

Just-In Time(JIT):

- JIT is a part of dot net which is responsible for managing the execution of dot net programs.
- A language specific compiler converts the source code to the IL.
- This IL is then converted into the machine code by the JIT compiler.

Types of JIT:

1. Pre JIT
2. Normal JIT
3. Econo JIT

Pre JIT:

- The complete source code is converted into native code in a single cycle.
- This is done at the time of application deployment.
- In dot net it is called ngen.exe(Native Image Generator)

Normal JIT:

- The compiler compiles only those methods that are called at runtime.
- After executing this method, compiled method are stored in memory cache.

Econo JIT:

- The compiler compiles only those methods that are called at runtime.
- After executing this method, compiled method are removed from memory.

.NET supports two kind of coding

1. Managed Code
2. Unmanaged Code

Managed Code

- The resource, which is with in your application domain is, managed code. The resources that are within domain are faster.
- The code, which is developed in .NET framework, is known as managed code. This code is directly executed by CLR with help of managed code execution. Any language that is written in .NET Framework is managed code.
- Managed code uses CLR which in turns looks after your applications by managing memory, handling security, allowing cross - language debugging, and so on

Unmanaged Code

- The code, which is developed outside .NET Framework is known as unmanaged code.
- Applications that do not run under the control of the CLR are said to be unmanaged, and certain languages such as C++ can be used to write such applications, which, for example, access low - level functions of the operating system. Background compatibility with code of VB, ASP and COM are examples of unmanaged code.
- Unmanaged code can be unmanaged source code and unmanaged compile code.
- Unmanaged code is executed with help of wrapper classes.

```
class Program
  {

    static unsafe void Main(string[] args)
    {
       int var = 20;
       int* p = &var;
       Console.WriteLine("Data is: {0} ", var);
       Console.WriteLine("Address is: {0}", (int)p);
       Console.ReadKey();
    }  }
```

Benefits of the Dot Net approach

- Simple and faster system development.
- It provides rich object model.
- Enhanced built-in functionality.
- Integration of different languages into one platform.
- Easy deployment and execution.
- Wide range of scalability.
- Interoperability with existing applications.
- Simple and easy to build development tools.
- Potentially better performance.
- Fewer bugs.

UNIT-II

Data types specify the type of data that a valid C# variable can hold. C# is a strongly typed programming language because in C#, each type of data (such as integer, character, float, and so forth) is predefined as part of the programming language and all constants or variables defined for a given program must be described with one of the data types.

Data types in C# is mainly divided into three categories

- Value Data Types
- Reference Data Types
- Pointer Data Type

Value Data Types : In C#, the Value Data Types will directly store the variable value in memory and it will also accept both signed and unsigned literals. The derived class for these data types are System.ValueType. Following are different Value Data Types in C# programming language :

- Signed & Unsigned Integral Types : There are 8 integral types which provide support for 8-bit, 16-bit, 32-bit, and 64-bit values in signed or unsigned form.

Data type	CTS	Type	Size(bits)	Range	Def val
sbyte	System.Sbyte	signed integer	8	-128 to 127	0
short	System.Int16	signed integer	16	-32768 to 32767	0
int	System.Int32	signed integer	32	-2^{31} to $2^{31}-1$	0
long	System.Int64	signed integer	64	-2^{63} to $2^{63}-1$	0L
byte	System.Byte	unsigned integer	8	0 to 255	0
ushort	System.UInt16	unsigned integer	16	0 to 65535	0
uint	System.UInt32	unsigned integer	32	0 to 2^{32}	0
ulong	System.UInt64	unsigned integer	64	0 to 2^{63}	0

Floating Point Types : There are 2 floating point data types which contain the decimal point.

- **Float**: It is **32-bit single-precision** floating point type. It has 7 digit Precision. To initialize a float variable, use the suffix f or F. Like, float x = 3.5F. If the suffix F or f will not use then it is treated as double.
- **Double**:It is **64-bit double-precision** floating point type. It has 14 – 15 digit Precision. To initialize a double variable, use the suffix d or D. But it is not mandatory to use suffix because by default floating data types are the double type.
- **Decimal Types :** The decimal type is a 128-bit data type suitable for financial and monetary calculations. It has 28-29 digit Precision. To initialize a decimal variable, use the suffix m or M. Like as, decimal x = 300.5m;. If the suffix m or M will not use then it is treated as double.

Data type	CTS	Size(bits)	Range	Def val
float	System.Single	32	$3.4 \times 10^{38}$to $+3.4 \times 10^{38}$	0.0F
double	System.Double	64	$\pm 5.0 \times 10^{-324}$ to $\pm 1.7 \times 10^{308}$	0.0D
decimal	System.Decimal	128	$-7.9 \times 10^{28}$to 7.9×10^{28}	0.0M

Character Types : The character types represents a UTF-16 code unit or represents the 16-bit Unicode character.

Data type	CTS	Size(bits)	Range	Def val

char	System.Char	16	U +0000 to U +ffff	'\0'

Boolean Types : It has to be assigned either true or false value. Values of type bool are not converted implicitly or explicitly (with casts) to any other type. But the programmer can easily write conversion code.

Data type	CTS	values
bool	System.Boolean	True / False

Reference Data Types : The Reference Data Types will contain a memory address of variable value because the reference types won't store the variable value directly in memory. The built-in reference types are **string, object.**

String : It represents a sequence of Unicode characters and its type name is **System.String**. So, string and String are equivalent.
Example :

- string s1 = "hello"; // creating through string keyword
- String s2 = "welcome"; // creating through String class
- **Object :** In C#, all types, predefined and user-defined, reference types and value types, inherit directly or indirectly from Object. So basically it is the base class for all the data types in C#. Before assigning values, it needs type conversion. When a variable of a value type is converted to object, it's called **boxing**. When a variable of type object is converted to a value type, it's called **unboxing**. Its type name is **System.Object**.

Type conversion:

Type conversion is converting one type of data to another type. It is also known as Type Casting. In C#, type casting has two forms −

- **Implicit type conversion** − these conversions are performed by C# in a type-safe manner.

Eg:

byte a=23;

int b=a;

- **Explicit type conversion** – these conversions are done explicitly by users using the pre-defined functions. Explicit conversions require a cast operator.

  Eg:

  int a=23;

  byte b=(byte)a;

  Using parsing method:

  The parsing method is used to convert string type data to primitive value type.

  Eg:

  int a=Int32.Parse(Console.readLine());

  Using convert class:

  The convert class is used to convert from one primitive data type to another primitive data type.

  This class contains different static methods like toint16(),toint32(),tostring(),todatetime() etc.,

  Eg:

```
        int a = 23;
float b = Convert.ToSingle(a);
```

Boxing and Unboxing:

boxing is a mechanism in which value type is converted into reference type.

It is implicit conversion process in which object type is used.

Unboxing is a mechanism in which reference type is converted into value type.

It is explicit conversion process.

Eg:

```
        int a = 23;
        string s=a.tostring();//boxing
object o = a;//boxing
int j = (int)o;//unboxing
```

Variables:

A variable is a name given to a memory location and all the operations done on the variable effects that memory location.

In C#, all the variables must be declared before they can be used.

It is the basic unit of storage in a program.

The value stored in a variable can be changed during program execution.

Types of Variable

- **Local variables**

```
using System;
class StudentDetails {

  // Method
  public void StudentAge()
  {

    // local variable age
    int age = 0;

    age = age + 10;
    Console.WriteLine("Student age is : " + age);
  }

  // Main Method
  public static void Main(String[] args)
  {

    // Creating object
    StudentDetails obj = new StudentDetails();

    // calling the function
    obj.StudentAge();
  }
}
```

- Instance variables or Non – Static Variables

```
using System;

class Marks {

   // These variables are instance variables.
   // These variables are in a class and
   // are not inside any function
   int engMarks;
   int mathsMarks;
   int phyMarks;

   // Main Method
   public static void Main(String[] args)
   {

      // first object
      Marks obj1 = new Marks();
      obj1.engMarks = 90;
      obj1.mathsMarks = 80;
      obj1.phyMarks = 93;

      // second object
      Marks obj2 = new Marks();
      obj2.engMarks = 95;
      obj2.mathsMarks = 70;
      obj2.phyMarks = 90;

      // displaying marks for first object
      Console.WriteLine("Marks for first object:");
      Console.WriteLine(obj1.engMarks);
      Console.WriteLine(obj1.mathsMarks);
      Console.WriteLine(obj1.phyMarks);

      // displaying marks for second object
      Console.WriteLine("Marks for second object:");
      Console.WriteLine(obj2.engMarks);
      Console.WriteLine(obj2.mathsMarks);
      Console.WriteLine(obj2.phyMarks);
   }
}
```

- Static Variables or Class Variables

```
using System;
class Emp {

   // static variable salary
   static double salary;
   static String name = "Aks";
```

```
    // Main Method
    public static void Main(String[] args)
    {

        // accessing static variable
        // without object
        Emp.salary = 100000;

        Console.WriteLine(Emp.name + "'s average salary:"
                                + Emp.salary);
    }
}
```

- ## Constant Variables

```
using System;
class Program {

    // instance variable
    int a = 10;

    // static variable
    static int b = 20;

    // constant variable
    const float max = 50;

    // Main Method
    public static void Main()
    {

        // creating object
        Program obj = new Program();

        // displaying result
        Console.WriteLine("The value of a is = " + obj.a);
        Console.WriteLine("The value of b is = " + Program.b);
        Console.WriteLine("The value of max is = " + Program.max);
    }
}
```

- ## Readonly Variables

```
using System;
class Program {

    // instance variable
    int a = 80;
```

```
    // static variable
    static int b = 40;

    // Constant variables
    const float max = 50;

    // readonly variables
    readonly int k;

    // Main Method
    public static void Main()
    {

        // Creating object
        Program obj = new Program();

        Console.WriteLine("The value of a is = " + obj.a);
        Console.WriteLine("The value of b is = " + Program.b);
        Console.WriteLine("The value of max is = " + Program.max);
        Console.WriteLine("The value of k is = " + obj.k);
    }
}
```

Defining Variables

Syntax for variable definition in C# is −

<data_type> <variable_list>;

Here, data_type must be a valid C# data type including char, int, float, double, or any user-defined data type, and variable_list may consist of one or more identifier names separated by commas.

Some valid variable definitions are shown here −

int i, j, k;

char c, ch;

float f, salary;

double d;

Initializing Variables

Variables are initialized (assigned a value) with an equal sign followed by a constant expression. The general form of initialization is −

variable_name = value;

```
int d = 3, f = 5;

byte z = 22;

double pi = 3.14159;

char x = 'x';
```

Rules:

- It must not begin at a digit.
- Uppercase and lowercase letters are distinct.
- It should not be a keyword.
- Whitespace is not allowed.
- Variable names can be of any length.

Arrays

- An array is a group of like-typed variables that are referred to by a common name. And each data item is called an element of the array.
- The data types of the elements may be any valid data type like char, int, float etc. and the elements are stored in a contiguous location.
- The variables in the array are ordered and each has an index beginning from 0.

Declaring Arrays

To declare an array in C#, use the following syntax

datatype[] arrayName;

where,

- *datatype* is used to specify the type of elements in the array.
- *[ ]* specifies the size of the array.
- *arrayName* specifies the name of the array.

For example,

double[] balance;

Initializing an Array

Declaring an array does not initialize the array in the memory. When the array variable is initialized, user can assign values to the array.

Array is a reference type, so user need to use the **new** keyword to create an instance of the array. For example,

```
double[] balance = new double[10];
```

Assigning Values to an Array

User can assign values to individual array elements, by using the index number, like –

```
double[] balance = new double[10];

balance[0] = 4500.0;
```

User can assign values to the array at the time of declaration, as shown –

```
double[] balance = { 2340.0, 4523.69, 3421.0};
```

alternate,

```
int [] marks = new int[5]  { 99,  98, 92, 97, 95};
```

(or)

```
int [] marks = new int[]  { 99,  98, 92, 97, 95};
```

user can copy an array variable into another target array variable.

```
int [] marks = new int[]  { 99,  98, 92, 97, 95};

int[] score = marks;
```

Accessing Array Elements

```
using System;

   class MyArray {

      static void Main(string[] args) {

         int []  n = new int[10];
```

```
        int i,j;


        for ( i = 0; i < 10; i++ )
          n[ i ] = i + 100;
        for (j = 0; j < 10; j++ )
          Console.WriteLine("Elements”+ n[j]);
```

(Or)

```
Foreach(int j in n)
Console.writeline(j);
        Console.ReadKey();
    }
  }
```

Array Types

Arrays can be divided into the following four categories.

- Single-dimensional arrays
- Multidimensional arrays or rectangular arrays
- Jagged arrays
- Mixed arrays.

Single Dimensional Arrays

Single-dimensional arrays are the simplest form of arrays. These types of arrays are used to store number of items of a predefined type. All items in a single dimension array are stored contiguously starting from 0 to the size of the array -1.

Eg:

```
        int[] intArray;
        intArray = new int[3];
```

Multi-Dimensional Arrays

A multi-dimensional array, also known as a rectangular array is an array with more than one dimension. The form of a multi-dimensional array is a matrix.

Declaring a multi-dimensional array

A multi-dimension array is declared as follows:

string[,] mutliDimStringArray;

Initializing multi-dimensional arrays

```
int[,] numbers = new int[3, 2] { { 1, 2 }, { 3, 4 }, { 5, 6 } };
class Program
  {
    static void Main()
     {
        int[,] twod = new int[2, 3];
        int count = 0;
        for (int i = 0; i < 2; i++)
           for (int j = 0; j < 3; j++)
         twod[i, j] = count++;
        for (int i = 0; i < 2; i++)
           for (int j = 0; j < 3; j++)
              Console.WriteLine(twod[i, j]);
           Console.ReadKey();
     }
  }
```

Jagged Arrays

Jagged arrays are arrays of arrays. The elements of a jagged array are other arrays.

Declaring Jagged Arrays

Declaration of a jagged array involves two brackets. For example,

int [][] scores;

int[][] scores = new int[5][];

for (int i = 0; i < scores.Length; i++) {

  scores[i] = new int[4];

}

User can initialize a jagged array as –

int[][] scores = new int[2][]{new int[]{92,93,94},new int[]{85,66,87,88}};

Using Array class:

The Array class gives methods for creating, manipulating, searching, and sorting arrays.

```
class GFG {

  // Main Method
  public static void Main()
  {

    // declares an 1D Array of string.
    string[] topic;

    // allocating memory for topic.
    topic = new string[] {"Array, ", "String, ",
                "Stack, ", "Queue, ",
                "Exception, ", "Operators" };

    // Displaying Elements of
    // the array before reverse
    Console.WriteLine("Topic of C# before reverse:");
    Console.WriteLine();
    foreach(string ele in topic)
    {
      Console.WriteLine(ele + " ");
    }
    Console.WriteLine();

    // using Reverse() method to
    // reverse the given array
```

```
        Array.Reverse(topic);

        // Displaying Elements of array after reverse
        Console.WriteLine("Topic of C# after reverse:");
        Console.WriteLine();
        foreach(string val in topic)
        {
            Console.WriteLine(val + " ");
        }
    }
}
```

Array.sort(),Array.reverse(),

ArrayList: ArrayList represents an ordered collection of an object that can be indexed individually. It is basically an alternative to an array. It also allows dynamic memory allocation, adding, searching and sorting items in the list.

```
using System;
using System.Collections;

class GFG {

    // Main Method
    public static void Main(string[] args)
    {

        // Create a list of strings
        ArrayList al = new ArrayList();
        al.Add("Ajay");
        al.Add("Ankit");
        al.Add(10);
        al.Add(10.10);

        // Iterate list element using foreach loop
        foreach(var names in al)
        {
            Console.WriteLine(names);
        }
    }
}
```

Properties

- One of the design goals of object oriented system is not to permit any direct access to data members (private).
- Properties are the special type of class members that provides a flexible mechanism to read, write, or compute the value of a private field.

- So, to access data members, c# provides special methods called accessor method.
- **Accessors:** The block of “set” and “get” is known as “Accessors”.
- **Set** property is used to write a value or store a value.
- **Get** property is used to read or retrieve a value.

There are different types of properties based on the “get” and set accessors:

- **Read and Write Properties:** When property contains both get and set methods.
- **Read-Only Properties:** When property contains only get method.
- **Write Only Properties:** When property contains only set method.

```
Syntax:
<access_modifier> <return_type> <property_name>
{
    get { // body }
    set { // body }
}
```

Get Accessor: It specifies that the value of a field can access publicly. It returns a single value and it specifies the *read-only property*.

Example:

```
class Geeks {

private int roll_no;

public int Roll_no

{

  get

   {

    return roll_no;
```

}}}

Set Accessor: It will specify the assignment of a value to a private field in a property. It returns a single value and it specifies the *write-only property*.

Example:

```
class Geeks {

private int roll_no;

public int Roll_no

 {

   set

    {

       roll_no = value;     }}}
```

example:

```
public class Student {

     private string name;

   public string Name
   {

     get
     {
        return name;
     }

     set
     {
        name = value;
     }
   }
}

class TestStudent {

   // Main Method
   public static void Main(string[] args)
   {
```

```
        Student s = new Student();

         s.Name = "BCA";

        Console.WriteLine("Name: " + s.Name);
    }
}
```

Namespace:

Grouping of logical related classes or collections of logical related classes are called namepace.

The general form is

Defining a Namespace

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

namespace name

{

Type declarations

}

Where, name can be any legal identifier.

Type declarations can declare one or more of the following types

Class,interface and etc.,

Example:

namespace bca

{

class test

{

//class code

}

```
}
```

Creating namespace:

Using class library file:

```
Namespace john
{
public class Class1
 {
    public void disp()
    {
       Console.WriteLine("welcome to c# namespace");
    }
 }
}
```

Step 1:

Build solution.

Step 2:

Create new console application

Step 3:

Call Add reference (john.dll) using solution explorer window.

Step 4:

Implementation:

using john;

```
class Program
 {
    static void Main(string[] args)
    {
       Class1 ob = new Class1();
       ob.disp();
       Console.ReadKey();
    }
 }
```

Nested namespace:

user can also define a namespace into another namespace which is termed as the nested namespace. To access the members of nested namespace user has to use the

dot(.) operator.

Creating nested namespace:

```
namespace john
{
   namespace john1
   {
      public class Class1
      {
         public void disp()
         {
            Console.WriteLine("welcome to c# namespace");
         }
      }
   }
}
```

implemenation

```
using john.john1;
namespace namespaceexample

{
   class Program
   {
      static void Main(string[] args)
      {
         Class1 ob = new Class1();
         ob.disp();
         Console.ReadLine();
      }
   }
}
```

Methods:

A method is a group of statements that together perform a task. Every C# program has at least one class with a method named Main.

To use a method, you need to −

- Define the method
- Call the method

Defining Methods in C#

When you define a method, you basically declare the elements of its structure. The syntax for defining a method in C# is as follows −

<Access Specifier> <Return Type> <Method Name>(Parameter List)

{

Method Body

}

Following are the various elements of a method −

- **Access Specifier** − This determines the visibility of a variable or a method from another class.
- **Return type** − A method may return a value. The return type is the data type of the value the method returns. If the method is not returning any values, then the return type is **void**.
- **Method name** − Method name is a unique identifier and it is case sensitive. It cannot be same as any other identifier declared in the class.
- **Parameter list** − Enclosed between parentheses, the parameters are used to pass and receive data from a method. The parameter list refers to the type, order, and number of the parameters of a method. Parameters are optional; that is, a method may contain no parameters.
- **Method body** − this contains the set of instructions needed to complete the required activity.

  Ex:

  public void disp()

  {

  Console.writeline("welcome to c# method");

  }

Passing Parameters to a Method

When method with parameters is called, you need to pass the parameters to the method. There are four ways that parameters can be passed to a method

1. Value parameters

This method copies the actual value of an argument into the formal parameter of the

function. In this case, changes made to the parameter inside the function have no effect on the argument.

Default parameter is value parameter.

```
class Program
  {
     static void dis(int x)
     {
        x= x + 10;
        Console.WriteLine("the value is"+x);

     }
     static void Main(string[] args)
     {
        int x = 4;
        dis(x);
        Console.WriteLine(x);
        Console.Read();
     }
  }
```

2 Reference parameters

This method copies the reference to the memory location of an argument into the formal parameter. This means that changes made to the parameter affect the argument.

To implement this, use **ref** keyword.

```
static void dis(ref int x)
     {
        x= x + 10;
        Console.WriteLine("the value is"+x);

     }
     static void Main(string[] args)
     {
        int x = 4;
        dis(ref x);
        Console.WriteLine(x);
        Console.Read();
     }
  }
```

3 Output parameters

Output parameters are used to pass result back to the calling method.this is achieved by declaring the parameter with an **out** keyword.

```
class Program
```

```
{
   static void dis( int x,out int res)
   {
      res= x + 10;
      Console.WriteLine("the value is"+res);

   }
   static void Main(string[] args)
   {
      int x = 4,m;
      dis( x,out m);
      Console.WriteLine(m);
      Console.Read();
   }
}
```

4.parameter array

In c#, user can define methods that can handle variable with number of arguments called parameter array.this is achieved by **params** keyword.

```
class Program
  {
     static void dis( params int[] x)
     {
        foreach (int a in x)
           Console.WriteLine(a);

     }
     static void Main(string[] args)
     {
        int[] x = { 11, 22, 33, 44, 55 };
        dis(x);

        Console.Read();
     }
  }
```

Delegation

A delegate is a type that references a method.

Once a delegate is assigned a method, it behaves exactly like that method.

The delegate method can be used like any other method, with parameters and return a value.

Example:

public delegate int delegatename(int x,int y);

class bca {

```
public delegate void addnum(int a, int b);
public delegate void subnum(int a, int b);

   public void sum(int a, int b)
   {
      Console.WriteLine("(100 + 40) = {0}", a + b);
   }

   public void subtract(int a, int b)
   {
      Console.WriteLine("(100 - 60) = {0}", a - b);
   }

public static void Main(String []args)
{


   bca ob = new bca();


   addnum del_obj1 = new addnum(ob.sum);
   subnum del_obj2 = new subnum(ob.subtract);


   del_obj1(100, 40);
   del_obj2(100, 60);
 }
}
```

Delegation ability to multicast:

Delegate's ability to multicast means that a delegate object can maintain a list of methods to call,rather than a single method.if user want to add a method to the invocation list of a object ,user simply make use of the overloaded += operator,and if user want to remove a method from the invocation list,user make use of the overloaded operator -=.

Example:

```
class bca {
public delegate void addnum(int a, int b);

   public void sum(int a, int b)
   {
      Console.WriteLine("(100 + 40) = {0}", a + b);
   }

   public void subtract(int a, int b)
   {
      Console.WriteLine("(100 - 60) = {0}", a - b);
   }

public static void Main(String []args)
{
```

```
    bca ob = new bca();


    addnum del_obj1 = new addnum(ob.sum);

  del_obj1 +=new addnum(ob.subtract);
del_obj1 -=new addnum(ob.sum);

    del_obj1(100, 40);
    del_obj2(100, 60);
  }
}
```

Interfaces:

Like a class, ***Interface*** can have methods, properties, events, and indexers as its members. But interfaces will contain only the declaration of the members. The implementation of interface's members will be given by class who implements the interface implicitly or explicitly.

- Interfaces can't have private members.
- By default all the members of Interface are public and abstract.
- The interface will always defined with the help of keyword '***interface***'.
- Interface cannot contain fields because they represent a particular implementation of data.
- *Multiple inheritance* is possible with the help of Interfaces but not with classes.

Syntax:

```
interface  <interface_name >
{
      // declare methods
    // declare properties
}
```

Syntax for Implementing Interface:

class class_name : interface_name

To declare an interface, use *interface* keyword. It is used to provide total abstraction.

That means all the members in the interface are declared with the empty body and are public and abstract by default.

A class that implement interface must implement all the methods declared in the interface.

```
interface bca
{

   void display();
}

class Program : bca
{


   public void display()
   {
      Console.WriteLine("welcome to III BCA-A");
   }


   public static void Main (String []  args)
   {

      // Creating object
      Program ob = new Program();

      // calling method
      ob.display();
   }
}
```

Extending interface:

Like classes,interfaces can also be extended

```
Interface bca
{
Members;
}

Interface bca1:bca
{
Members;
}
```

Multiple inheritance:

```
interface bca
{
void add(int x,int y);
}

interface bca1
{
void sub(int x,int y);
}

class test : bca,bca1
{
public void add(int x,int y)
{
Console.WriteLine(x+y);
}

public void sub(int x,int y)
{
Console.WriteLine(x-y);
}

class program
{
static void Main(String[] arg)
{
test ob=new test();
ob.add(4,2);
ob.sub(4,3);
}
}
```

UNIT-III

ASP.NET

Introduction:

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

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

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

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

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

Difference between ASP and ASP.NET:

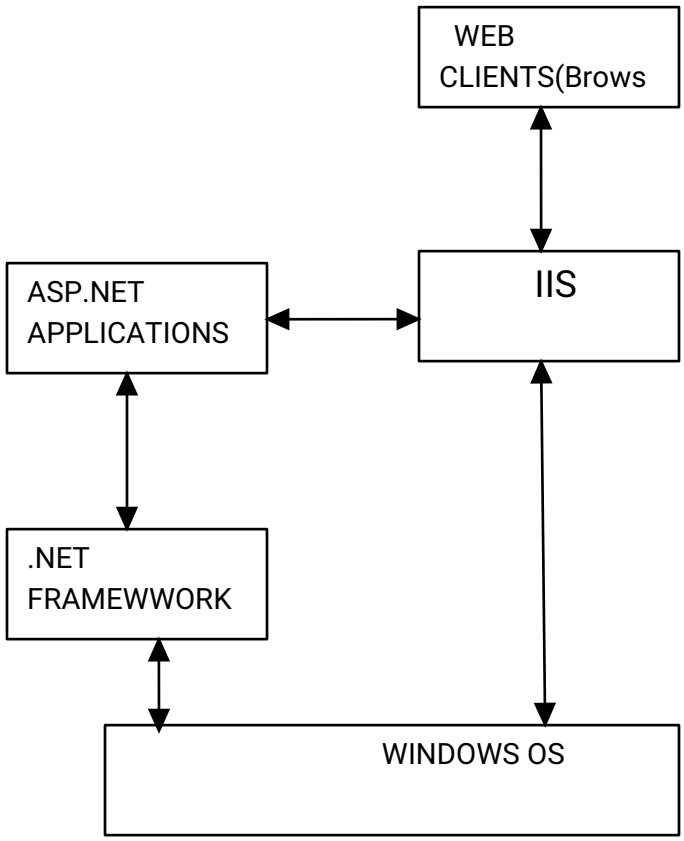
ASP is interpreted whereas, ASP.NET is compiled. This implies that since ASP uses VBScript; therefore, when an ASP page is executed, it is interpreted. On the other hand, ASP.NET uses .NET languages, such as C# and VB.NET, which are compiled to Microsoft Intermediate Language (MSIL).

Difference between ASP and ASP.NET:

ASP	ASP.NET
ASP is the interpreted languages.	ASP.NET is the compiled language.
ASP uses **ADO** (ActiveX Data Objects) technology to connect and work with databases.	ASP.NET uses **ADO.NET** to connect and work with database.
ASP is partially object oriented.	ASP.NET is fully object oriented.

ASP	ASP.NET
ASP Pages have the file extension **.asp**.	ASP.NET Pages have the file extension **.aspx**.
ASP doesn't have the concept of inheritance.	ASP.NET inherits the class written in code behind.
ASP pages use scripting language.	ASP.NET use full fledged programming language.
Error handling is very poor in ASP.	Error handling is very good in ASP.NET.
ASP has maximum four in-built classes i.e. Request, Response, Session and Application.	ASP.NET has more than 2000 in-built classes.

Architecture of ASP.NET



- This is an overview of the asp.net infrastructure and sub system relationships.
- In this diagram shows, all web clients communicate with asp.net applications through IIS.
- IIS interpreted and optionally authenticated the request.
- IIS also find the requested resource (asp.net app) and if the client is authorized ,returns the application resource.

Using asp.net configuration files

Every computing platform needs a configuration mechanism to control the behavior of the platform.

All configuration information for asp.net is contained in files named web.config and machine.config(C:\Windows\Microsoft.NET\Framework\v4.0.30319\Config).

Here windows provide an environment variable called path that controls the search path for executable programs.

In asp.net, a configuration file is depends on XML files(web.config&machine.config) .

The second way in which applications have configured themselves in the part is through the registry.

Machine.config file:

The default .net configuration for your machine is declared within a file named machine.config.

User can find machine.config within the directory
C:\Windows\Microsoft.NET\Framework\v4.0.30319\Config

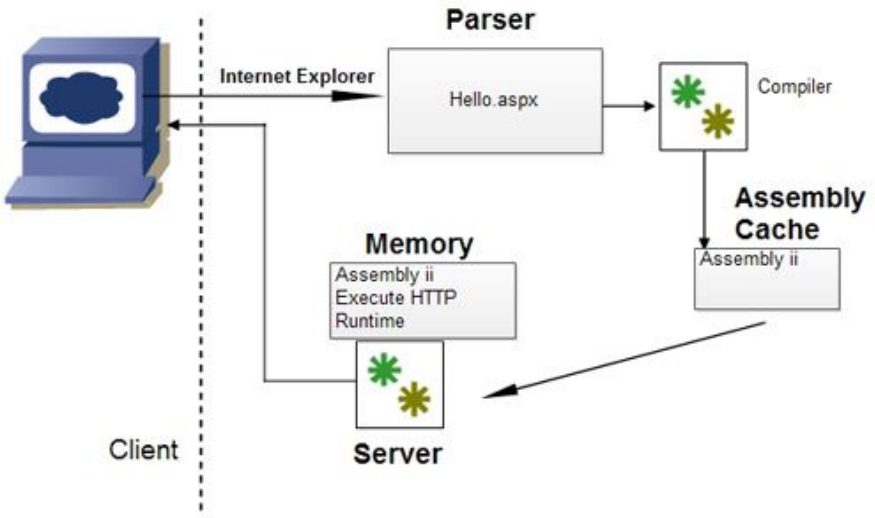
Execution Model

1. The client browser issues· a GET HTTP request to the server.

2. The ASP.NET parser interprets the source code.

3. If the code was not already compiled into a dynamic-link library (DLL), ASP.NET invokes the compiler.

4. Runtime loads and executes the Microsoft intermediate language (MSIL) code.



ASP Execution Model

When the user requests the same Web page for the second time, the following set of events take place:

1. The client browser issues a GET HTTP request to the server.

2. Runtime loads and immediately executes the MSIL code that was already compiled during the user's first access attempt.

Difference between ASPX file and Code behind File

Examining ASPX File:

A **Page** directive (in an ASPX file a **directive** is delimited by **<%@**and **%>**) to specify information needed by ASP.NET to process this file.

The **Language** attribute of the Page directive specifies the language of the code-behind file as C#.

The code-behind file (i.e., the **CodeFile**).Note that a code-behind file name usually consists of the full ASPX file name (e.g., WebTime.aspx) followed by the .cs extension.

The **AutoEventWireup** attribute determines how Web Form events are handled. When AutoEventWireup is set to true, ASP.NET determines which methods in the class are called in response to an event generated by the Page. For example, ASP.NET will call methods Page_Load in the code-behind file to handle the Page's Load.

The **Inherits** attribute specifies the class in the code-behind file from which this ASP.NET class inherits.

Example for aspx file

```
<%@ 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>
<%Response.Write(DateTime.Now); %>

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

Examining a Code-Behind File

Code Behind File:

Code Behind refers to the code for an ASP.NET Web page that is written in a separate class file that can have the extension of .aspx.cs or .aspx.vb depending on the language used.

Here the code is compiled into a separate class from which the .aspx file derives. User can write the code in a separate .cs or .vb code file for each .aspx page.

One major point of Code Behind is that the code for all the Web pages is compiled into a DLL file that allows the web pages to be hosted free from any Inline Server Code.

Inline Code

Inline Code refers to the code that is written inside an ASP.NET Web Page that has an extension of .aspx. It allows the code to be written along with the HTML source code using a <Script> tag.

It's major point is that since it's physically in the .aspx file it's deployed with the Web Form page whenever the Web Page is deployed.

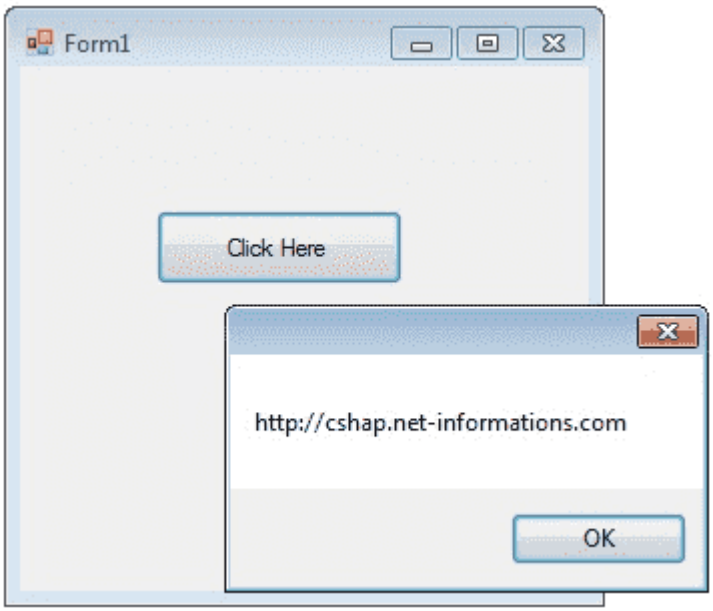
```
using System;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
```

```
public partial class _Default : System.Web.UI.Page
{    protected void Page_Load(object sender, EventArgs e)
    {
Response.Write(DateTime.Now); }    }
```

UNIT-IV

Button Control

Windows Forms controls are reusable components that encapsulate user interface functionality and are used in client side Windows applications. A button is a control, which is an interactive component that enables users to communicate with an application. The Button class inherits directly from the ButtonBase class. A Button can be clicked by using the mouse, ENTER key, or SPACEBAR if the button has focus.



When you want to change display text of the Button , you can change the Text property of the button.

```
button1.Text = "Click Here";
```

Similarly if you want to load an Image to a Button control , you can code like this

```
button1.Image = Image.FromFile("C:\\testimage.jpg");
```

The following C# source code shows how to change the button Text property while Form loading event and to display a message box when pressing a Button Control.

```
using System;
using System.Drawing;
using System.Windows.Forms;

namespace WindowsFormsApplication1
{
   public partial class Form1 : Form
```

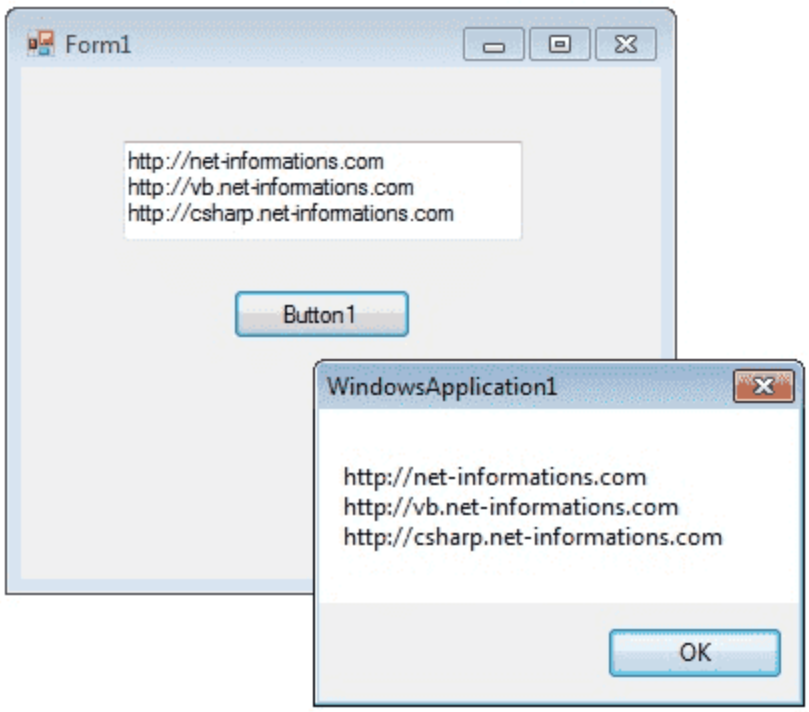
```
  {
    public Form1()
    {
      InitializeComponent();
    }

    private void Form1_Load(object sender, EventArgs e)
    {
      button1.Text = "Click Here";
    }

    private void button1_Click(object sender, EventArgs e)
    {
      MessageBox.Show("http://cshap.net-informations.com");
    }
  }
}
```

TextBox Control

A TextBox control is used to display, or accept as input, a single line of text. This control has additional functionality that is not found in the standard Windows text box control, including multiline editing and password character masking.



A text box object is used to display text on a form or to get user input while a C# program is running. In a text box, a user can type data or paste it into the control from the clipboard.

For displaying a text in a TextBox control , you can code like this

```
textBox1.Text = "http://csharp.net-informations.com";
```

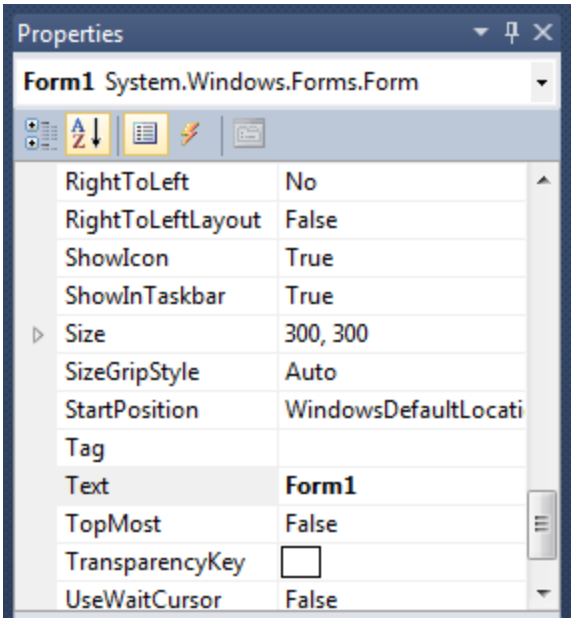
You can also collect the input value from a TextBox control to a variable like this way

```
string var;

var = textBox1.Text;
```

C# TextBox Properties

You can set TextBox properties through Property window or through program. You can open Properties window by pressing F4 or right click on a control and select Properties menu item



The below code set a textbox width as 250 and height as 50 through source code.

```
textBox1.Width = 250;
textBox1.Height = 50;
```

Background Color and Foreground Color

You can set background color and foreground color through property window and programmatically.

```
textBox1.BackColor = Color.Blue;
textBox1.ForeColor = Color.White;
```

Textbox BorderStyle

You can set 3 different types of border style for textbox, they are None, FixedSingle and fixed3d.

```
textBox1.BorderStyle = BorderStyle.Fixed3D;
```

TextBox Events

Keydown event

You can capture which key is pressed by the user using KeyDown event

e.g.

```
private void textBox1_KeyDown(object sender, KeyEventArgs e)
{
    if (e.KeyCode == Keys.Enter)
    {
        MessageBox.Show("You press Enter Key");

    }
    if (e.KeyCode == Keys.CapsLock)
    {
        MessageBox.Show("You press Caps Lock Key");
    }
}
```

TextChanged Event

When user input or setting the Text property to a new value raises the TextChanged event

e.g.

```
private void textBox1_TextChanged(object sender, EventArgs e)
{
    label1.Text = textBox1.Text;
}
```

Multiline TextBox

You can use the Multiline and ScrollBars properties to enable multiple lines of text to be displayed or entered.

```
textBox1.Multiline = true;
```

Textbox passowrd character

TextBox controls can also be used to accept passwords and other sensitive

information. You can use the PasswordChar property to mask characters entered in a single line version of the control

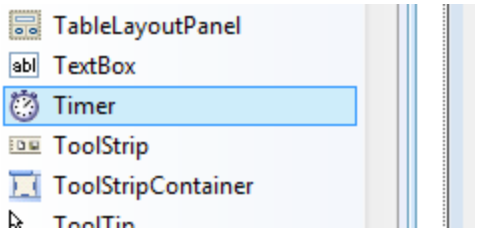
```
textBox1.PasswordChar = '*';
```

The above code set the PasswordChar to * , so when the user enter password then it display only * instead of typed characters.

TIMER CONTROL:

What is Timer Control ?

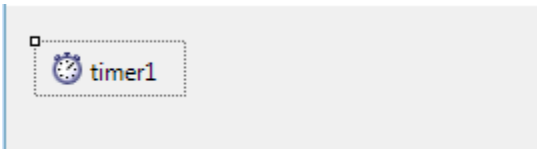
The Timer Control plays an important role in the development of programs both Client side and Server side development as well as in Windows Services. With the Timer Control we can raise events at a specific interval of time without the interaction of another thread.



Use of Timer Control

We require Timer Object in many situations on our development environment. We have to use Timer Object when we want to set an interval between events, periodic checking, to start a process at a fixed time schedule, to increase or decrease the speed in an animation graphics with time schedule etc.

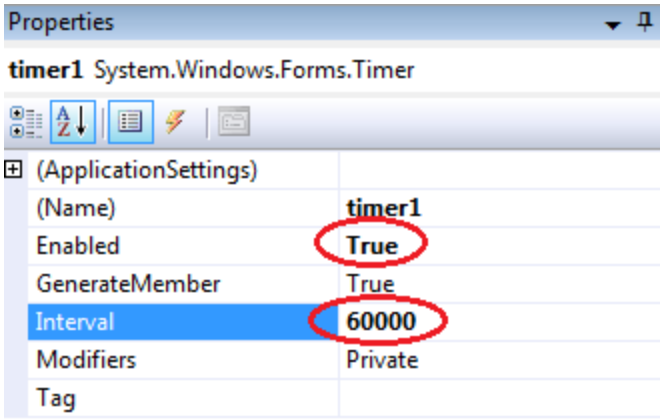
A Timer control does not have a visual representation and works as a component in the background.



How to Timer Control ?

We can control programs with Timer Control in millisecond, seconds, minutes and even in hours. The Timer Control allows us to set Intervel property in milliseconds. That is, one second is equal to 1000 milliseconds. For example, if we want to set an interval of 1 minute we set the value at Interval property as 60000, means 60x1000 .

By default the Enabled property of Timer Control is False. So before running the program we have to set the Enabled property is True , then only the Timer Control starts its function.



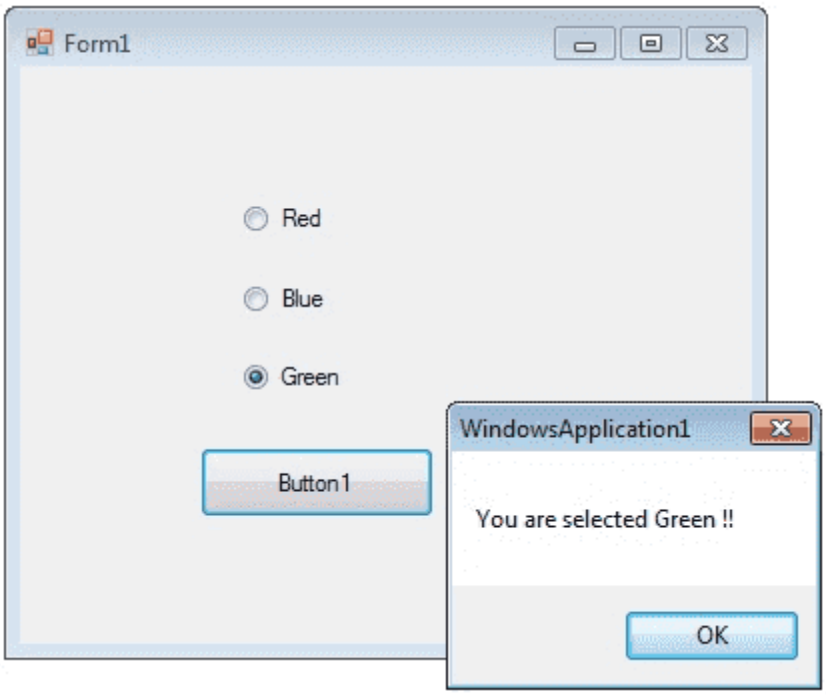
Timer example

In the following program we display the current time in a Label Control. In order to develop this program, we need a Timer Control and a Label Control. Here we set the timer interval as 1000 milliseconds, that means one second, for displaying current system time in Label control for the interval of one second.

```
using System;
using System.Windows.Forms;
namespace WindowsFormsApplication1
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
      }
      private void timer1_Tick(object sender, EventArgs e)
      {
         label1.Text = DateTime.Now.ToString();
      }
   }
}
```

RadioButton Control

A radio button or option button enables the user to select a single option from a group of choices when paired with other RadioButton controls. When a user clicks on a radio button, it becomes checked, and all other radio buttons with same group become unchecked



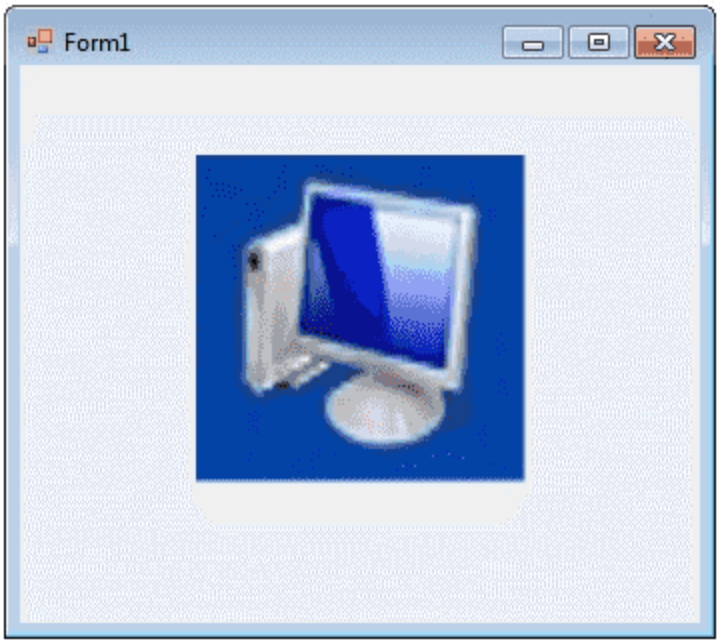
The RadioButton control can display text, an Image, or both. Use the Checked property to get or set the state of a RadioButton.

```
radioButton1.Checked = true;
```

The radio button and the check box are used for different functions. Use a radio button when you want the user to choose only one option. When you want the user to choose all appropriate options, use a check box. Like check boxes, radio buttons support a Checked property that indicates whether the radio button is selected.

PictureBox Control

The Windows Forms PictureBox control is used to display images in bitmap, GIF , icon , or JPEG formats.



You can set the Image property to the Image you want to display, either at design time or at run time. You can programmatically change the image displayed in a picture box, which is particularly useful when you use a single form to display different pieces of information.

```
pictureBox1.Image = Image.FromFile("c:\\testImage.jpg");
```

The SizeMode property, which is set to values in the PictureBoxSizeMode enumeration, controls the clipping and positioning of the image in the display area.

```
pictureBox1.SizeMode = PictureBoxSizeMode.StretchImage;
```

There are five different PictureBoxSizeMode is available to PictureBox control.

```
AutoSize    - Sizes the picture box to the image.
CenterImage      - Centers the image in the picture box.
Normal          - Places the upper-left corner of the image at upper
                                                    left in the picture box

StretchImage     - Allows you to stretch the image in code
```

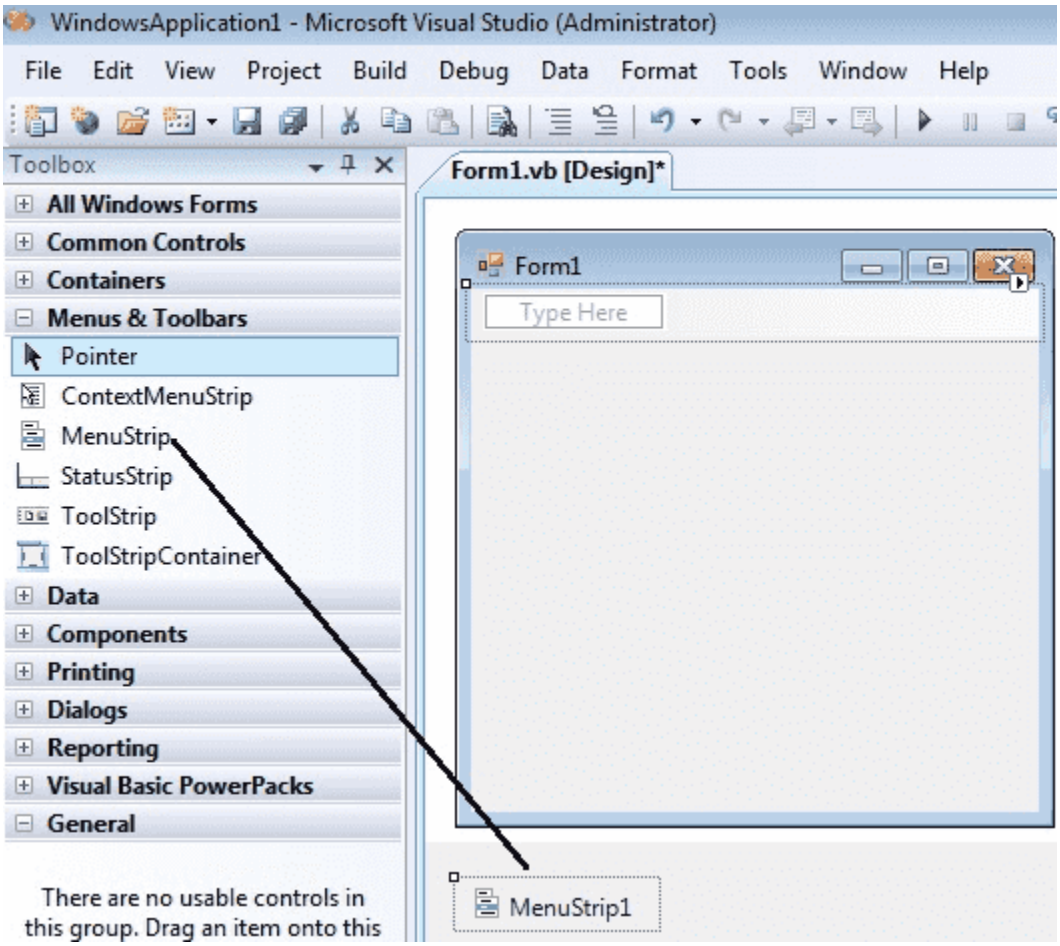
The PictureBox is not a selectable control, which means that it cannot receive input focus. The following C# program shows how to load a picture from a file and display it in streach mode.

Menu Control

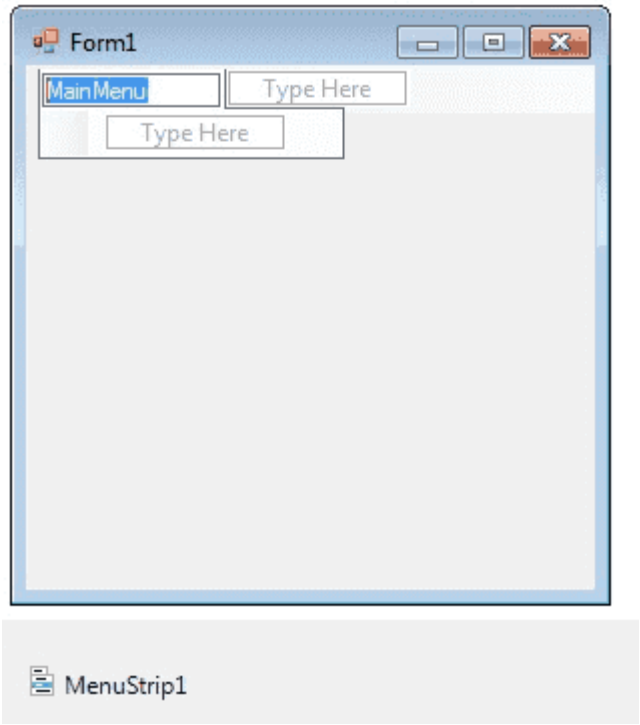
A Menu on a Windows Form is created with a MainMenu object, which is a collection

of MenuItem objects. MainMenu is the container for the Menu structure of the form and menus are made of MenuItem objects that represent individual parts of a menu.

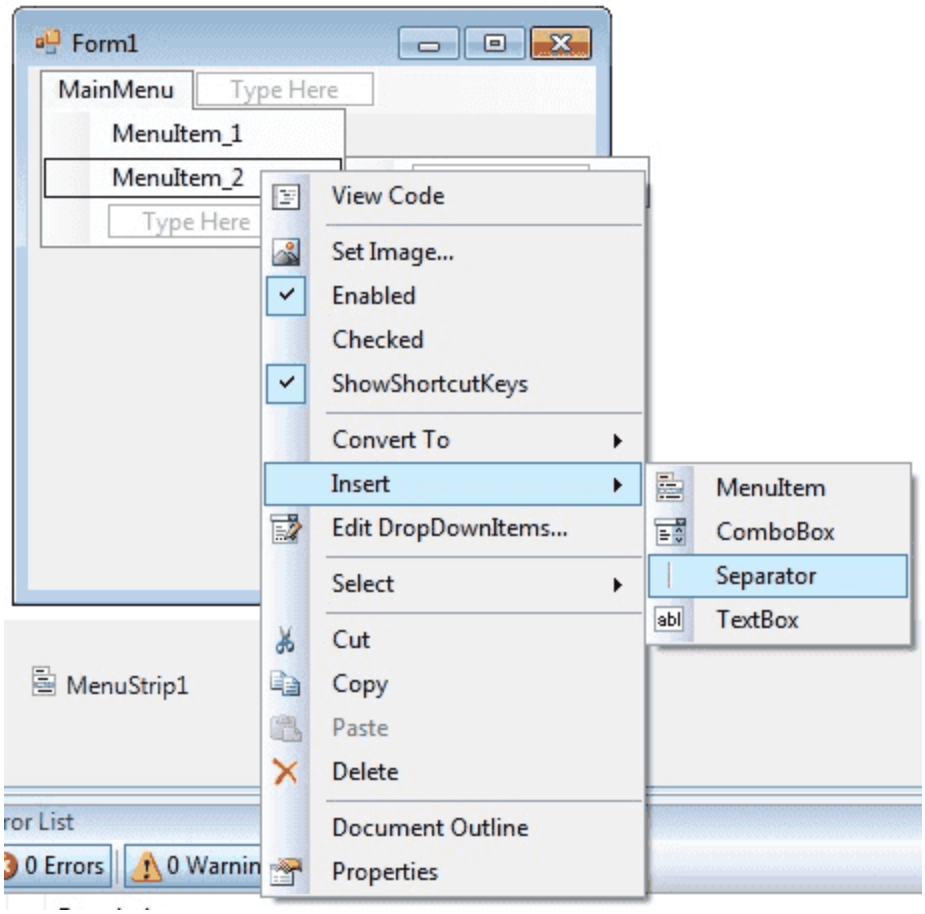
You can add menus to Windows Forms at design time by adding the MainMenu component and then appending menu items to it using the Menu Designer.



After drag the Menustrip on your form you can directly create the menu items by type a value into the "Type Here" box on the menubar part of your form. From the following picture you can understand how to create each menu items on mainmenu Object.



If you need a seperator bar , right click on your menu then go to insert->Seperator.



After creating the Menu on the form , you have to double click on each menu item and write the programs there depends on your requirements. The following C# program shows how to show a messagebox when clicking a Menu item.

ADROTATOR CONTROL

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

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

The basic syntax of adding an AdRotator is as follows:

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

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

The Advertisement File:

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

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

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

Following is an example of XML file:

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

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

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

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

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

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

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

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

Working with the AdRotator Control

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

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

</div>
</form>
```

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

Try to run the above application an dobserve that each time the page is reloaded, the ad is changed.

CALENDAR CONTROL:

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

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

The basic syntax for adding a calendar control is:

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

Properties and Events of the Calendar Control

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

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

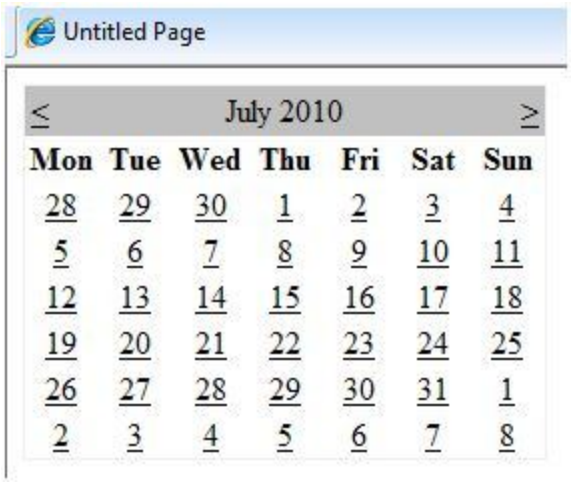
DayHeaderStyle	Gets the style properties for the section that displays the day of the week.
DayNameFormat	Gets or sets format of days of the week.
DayStyle	Gets the style properties for the days in the displayed month.
FirstDayOfWeek	Gets or sets the day of week to display in the first column.
NextMonthText	Gets or sets the text for next month navigation control; 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; default value is <
SelectedDate	Gets or sets the selected date.
SelectedDates	Gets a collection of DateTime objects representing the selected dates.
SelectedDayStyle	Gets the style properties for the selected dates.
SelectionMode	Gets or sets the selection mode that specifies whether the user can select a single day, a week or an entire month.
SelectMonthText	Gets or sets the text for the month selection element in the selector column.
SelectorStyle	Gets the style properties for the week and month selector column.
SelectWeekText	Gets or sets the text displayed for the week selection element in the selector column.
ShowDayHeader	Gets or sets the value indicating whether the heading for the days of the week is displayed.
ShowGridLines	Gets or sets the value indicating whether the gridlines would be shown.
ShowNextPrevMonth	Gets or sets a value indicating whether next and previous month navigation elements are shown in the title section.
ShowTitle	Gets or sets a value indicating whether the title section is displayed.
TitleFormat	Gets or sets the format for the title section.
Titlestyle	Get the style properties of the title heading for the Calendar control.
TodayDayStyle	Gets the style properties for today's date on the Calendar control.
TodaysDate	Gets or sets the value for today.s date.
UseAccessibleHeader	Gets or sets a value that indicates whether to render the table header <th> HTML element for the day headers instead of the table data <td> HTML element.
VisibleDate	Gets or sets the date that specifies the month to display.
WeekendDayStyle	Gets the style properties for the weekend dates on the Calendar control.

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

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

Working with the Calendar Control

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



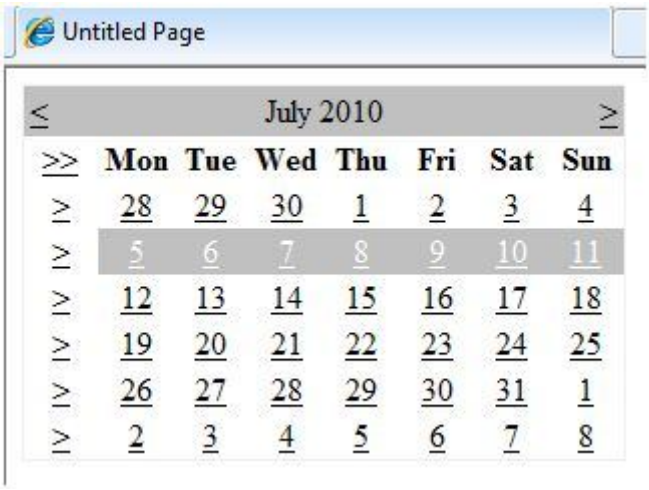
Calendar controls allow the users to select a single day, a week or an entire month. This is done by using the SelectionMode property. This property has the following values:

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

The syntax for selecting days:

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

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



Validation Control:

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

ASP.NET provides the following validation controls:

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

BaseValidator Class

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

Members	Description
ControlToValidate	Indicates the input control to validate.
Display	Indicates how the error message is shown.

EnableClientScript	Indicates whether client side validation will take.
Enabled	Enables or disables the validator.
ErrorMessage	Indicates error string.
Text	Error text to be shown if validation fails.
IsValid	Indicates whether the value of the control is valid.

RequiredFieldValidator Control

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

The syntax of the control is as given:

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

</asp:RequiredFieldValidator>
```

RangeValidator Control

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

It has three specific properties:

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

The syntax of the control is as given:

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

</asp:RangeValidator>
```

CompareValidator Control

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

It has the following specific properties:

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

The basic syntax of the control is as follows:

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

</asp:CompareValidator>
```

RegularExpressionValidator

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

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

Metacharacters	Description

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

The syntax of the control is as given:

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

</asp:RegularExpressionValidator>
```

CustomValidator

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

The basic syntax for the control is as given:

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

</asp:CustomValidator>
```

Example:

```
protected void CustomValidator1_ServerValidate(object source,
ServerValidateEventArgs args)

{

    if (TextBox1.Text == "bca")

       args.IsValid = true;

    else

       args.IsValid = false;

  }
```

ValidationSummary

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

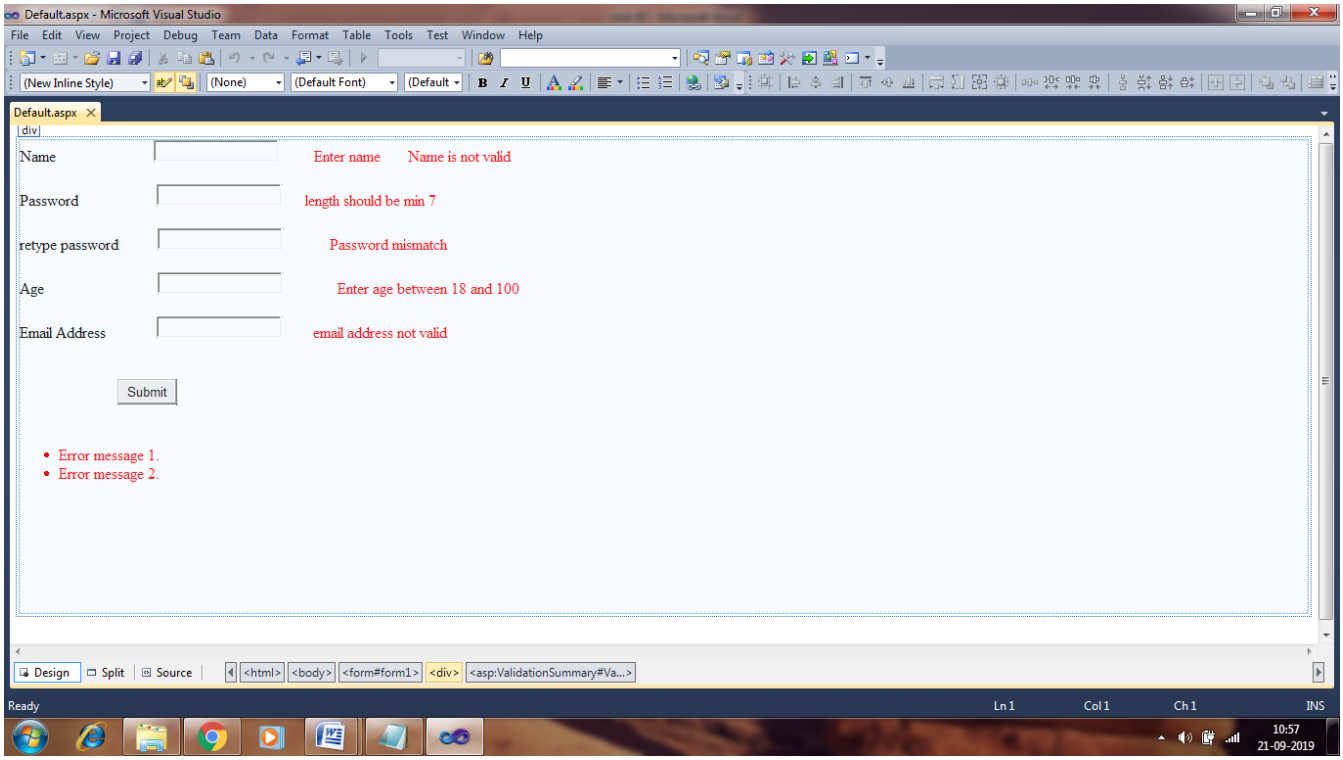
The following two mutually inclusive properties list out the error message:

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

The syntax for the control is as given:

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

Example :



UNIT-V

ADO.NET Object Model:

ADO.NET includes many objects you can use to work with data.

The SqlConnection Object

To interact with a database, you must have a connection to it.

The connection helps identify the database server, the database name, user name, password, and other parameters that are required for connecting to the data base.

A connection object is used by command objects so they will know which database to execute the command on.

The SqlCommand Object

The process of interacting with a database means that you must specify the actions you want to occur. This is done with a command object.

You use a command object to send SQL statements to the database. A command object uses a connection object to figure out which database to communicate with.

The SqlDataReader Object

Many data operations require that you only get a stream of data for reading. The data reader object allows you to obtain the results of a SELECT statement from a command object.

For performance reasons, the data returned from a data reader is a fast forward-only stream of data. This means that you can only pull the data from the stream in a sequential manner.

The DataSet Object

DataSet objects are in-memory representations of data. They contain multiple Datatable objects, which contain columns and rows, just like normal database tables.

The DataSet is specifically designed to help manage data in memory and to support disconnected operations on data, when such a scenario make sense.

The SqlDataAdapter Object

Sometimes the data you work with is primarily read-only and you rarely need to make changes to the underlying data source.

The data adapter makes it easy for you to accomplish these things by helping to manage data in a disconnected mode.

The data adapter fills a DataSet object when reading the data and writes in a single batch when persisting changes back to the database.

A data adapter contains a reference to the connection object and opens and closes the connection automatically when reading from or writing to the database.

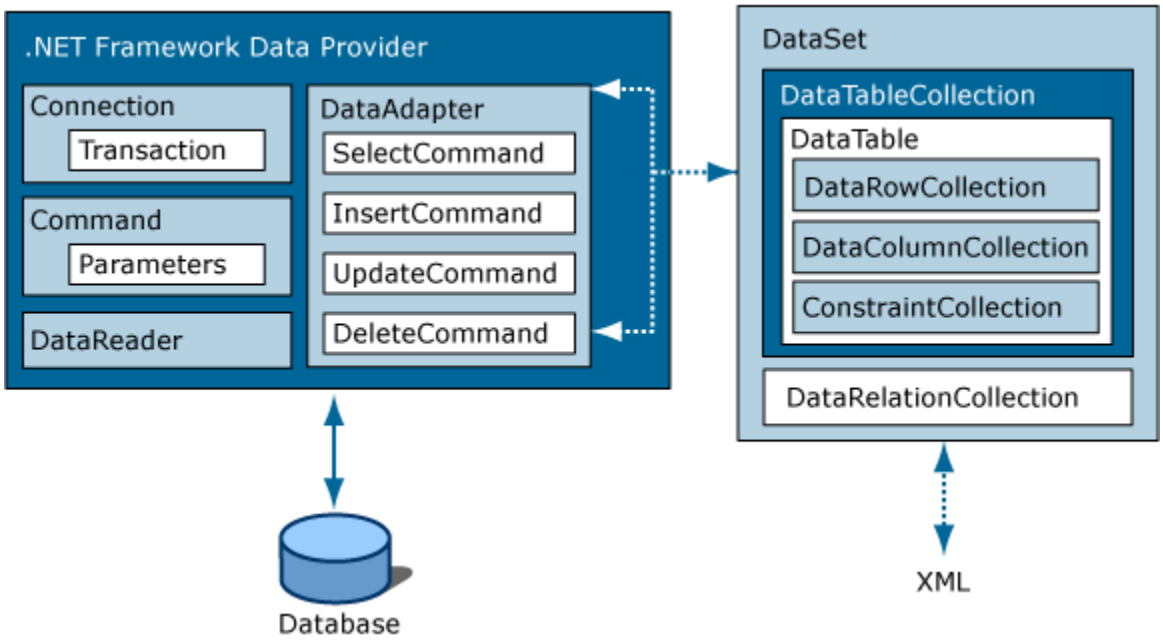
Additionally, the data adapter contains command object references for SELECT, INSERT, UPDATE, and DELETE operations on the data.

Architecture of Ado.net

- The .NET Framework Data Providers are components that have been explicitly designed for data manipulation and fast, forward-only, read-only access to data.
- The **Connection** object provides connectivity to a data source.
- The **Command** object enables access to database commands to return data, modify data, run stored procedures, and send or retrieve parameter information.
- The **DataReader** provides a high-performance stream of data from the data source.
- Finally, the DataAdapter provides the bridge between the **DataSet** object and the data source.
- The **DataAdapter** uses **Command** objects to execute SQL commands at the data source to both load the **DataSet** with data, and reconcile changes made to the data in the **DataSet** back to the data source.
- The ADO.NET DataSet is explicitly designed for data access independent of any data source.
- As a result, it can be used with multiple and differing data sources, used with XML data, or used to manage data local to the application.
- The **DataSet** contains a collection of one or more DataTable objects made up of rows and columns of data, as well as primary key, foreign key, constraint, and relation information about the data in the **DataTable** objects.

The following diagram illustrates the relationship between a .NET Framework data provider and a **DataSet**.

ADO.NET architecture



XML and ADO.NET

ADO.NET leverages the power of XML to provide disconnected access to data. ADO.NET was designed hand-in-hand with the XML classes in the .NET Framework; both are components of a single architecture.

ADO.NET and the XML classes in the .NET Framework converge in the **DataSet** object. The **DataSet** can be populated with data from an XML source, whether it is a file or an XML stream. The **DataSet** can be written as World-Wide Web Consortium (W3C) compliant XML, including its schema as XML Schema definition language (XSD) schema, regardless of the source of the data in the **DataSet**.

Introduction to ADO.NET

ADO.NET is an object-oriented set of libraries that allows you to interact with data sources. Commonly, the data source is a database, but it could also be a text file, an Excel spreadsheet, or an XML file.

As you are probably aware, there are many different types of databases available. For example, there is Microsoft SQL Server, Microsoft Access, Oracle,

Borland Interbase, and IBM DB2 etc.,

Data Providers

The ADO.NET allows us to interact with different types of data sources and different types of databases.

ADO.NET provides a relatively common way to interact with data sources, but comes in different sets of libraries for each way you can talk to a data source.

These libraries are called Data Providers and are usually named for the protocol or data source type they allow you to interact with.

The following lists some well known data providers, the API prefix they use, and the type of data source they allow you to interact with.

Provider Name	API prefix	Data Source Description
ODBCData provider	Odbc	Data Sources with an ODBC interface. Normally older data bases.
OleDbData Provider	OleDb	Data Sources that expose an OleDb interface, i.e. Access or Excel.
OracleData Provider	Oracle	For Oracle Databases.
SQLDataProvider	Sql	For interacting with Microsoft SQL Server.
Borland Data Provider	Bdp	Generic access to many databases such as Interbase, SQL Server, IBM DB2, and Oracle.

One of the first ADO.NET objects is the connection object, which allows you to establish a connection to a data source.

If we were using the OleDb Data Provider to connect to a data source that exposes an OleDb interface, we would use a connection object named OleDbConnection.

Similarly, the connection object name would be prefixed with Odbc or Sql for an OdbcConnection object on an Odbc data source or a SqlConnection object on a SQL Server database, respectively.

Syntax for Connection class:

Provider Name is OLEDB

OleDbConnection obj=new OleDbConnection(String ConnectionPath);

Ex:

```
OleDbConnection con=new
OleDbConnection(@"Provider=Microsoft.Jet.OLEDB.4.0;Data Source=D:\bca-
asp.net\login.mdb");
```

Syntax for Command class:

```
      OleDbCommand obj=new OleDbCommand(String SQL command,Connection
obj);
Eg:

OleDbCommand  cmd = new OleDbCommand("select *from stu where  uname='" +
textBox1.Text + "' and pwd='" + textBox2.Text + "' ", con);
```

Syntax for DataReader class:

```
 OleDbDataReader obj=commandclass_obj.ExecuteReader();
Eg:
OleDbDataReader dr=cmd.ExecuteReader();
```

Example using database.

```
using System;

using System.Windows.Forms;

using System.Data.OleDb;

namespace logincreation
{

   public partial class Form1 : Form
   {
      OleDbConnection con;
      OleDbCommand cmd;
      OleDbDataReader dr;
      public Form1()
      {
         InitializeComponent();
      }

      private void Form1_Load(object sender, EventArgs e)
      {
         con=new OleDbConnection(@"Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=D:\bca-asp.net\login.mdb");
```

```
    }

    private void button1_Click(object sender, EventArgs e)
    {
        con.Open();
        cmd = new OleDbCommand("select *from stu where  uname='" +
textBox1.Text + "' and pwd='" + textBox2.Text + "' ", con);
        dr = cmd.ExecuteReader();
        if (dr.Read())

            MessageBox.Show("successfully logged in");
        else
            MessageBox.Show("login failed");
        con.Close();

    }
  }
}
```

Data gridview control:

The GridView control provides many built-in capabilities that allow the user to sort, update, delete, select, and page through items in the control.

It is a commonly used control in **ASP.Net** web applications.

To use a **GridView** control a DataSource control has to be attached to the **GridView** control.

Example

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title></title>
</head>
<body>
  <form id="form1" runat="server">
  <div>

    <asp:GridView ID="GridView1" runat="server" AllowPaging="True"
      AllowSorting="True" DataSourceID="SqlDataSource1"
      AutoGenerateEditButton="True" DataKeyNames="uname"
      onrowupdating="GridView1_RowUpdating"
      onselectedindexchanged="GridView1_SelectedIndexChanged"
      AutoGenerateDeleteButton="True">
    </asp:GridView>
    <asp:SqlDataSource ID="SqlDataSource1" runat="server"
      ConnectionString="<%$ ConnectionStrings:loginConnectionString %>"
      ProviderName="<%$ ConnectionStrings:loginConnectionString.ProviderName %>"
      SelectCommand="SELECT [uname], [pwd] FROM [stu]"
       DeleteCommand="delete from stu where [uname]=@uname"
      UpdateCommand="UPDATE [stu] SET [pwd] = @pwd WHERE [uname] = @uname" >
```

```
            <UpdateParameters>
            <asp:Parameter Type="String" Name="pwd"></asp:Parameter>
                    </UpdateParameters>
          </asp:SqlDataSource>

    </div>
    </form>
```

Crystal Report

Crystal Report is a Reporting Application that can generate reports from various Data Sources like Databases , XML files etc..

The Visual Studio.NET Integrated Development Environment comes with Crystal Reports tools.

The Crystal Reports makes it easy to create simple reports, and also has comprehensive tools that you need to produce complex or specialized reports in csharp and other programming languages.

Crystal Reports is compatible with most popular development environments like **C#, VB.NET** etc.

You can use the Crystal Reports Designer in Visual Studio .NET to create a new report or modify an existing report.

```
using System;
using System.Windows.Forms;
using CrystalDecisions.CrystalReports.Engine;

namespace WindowsApplication1
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
      }

      private void button1_Click(object sender, EventArgs e)
      {
         ReportDocument cryRpt = new ReportDocument();
         cryRpt.Load(PUT CRYSTAL REPORT PATH HERE\\CrystalReport1.rpt");
         crystalReportViewer1.ReportSource = cryRpt;
         crystalReportViewer1.Refresh();
      }
   }
}
```