1. Explain .NET Framework Architecture.

- .NET framework consists several layers as shown below:

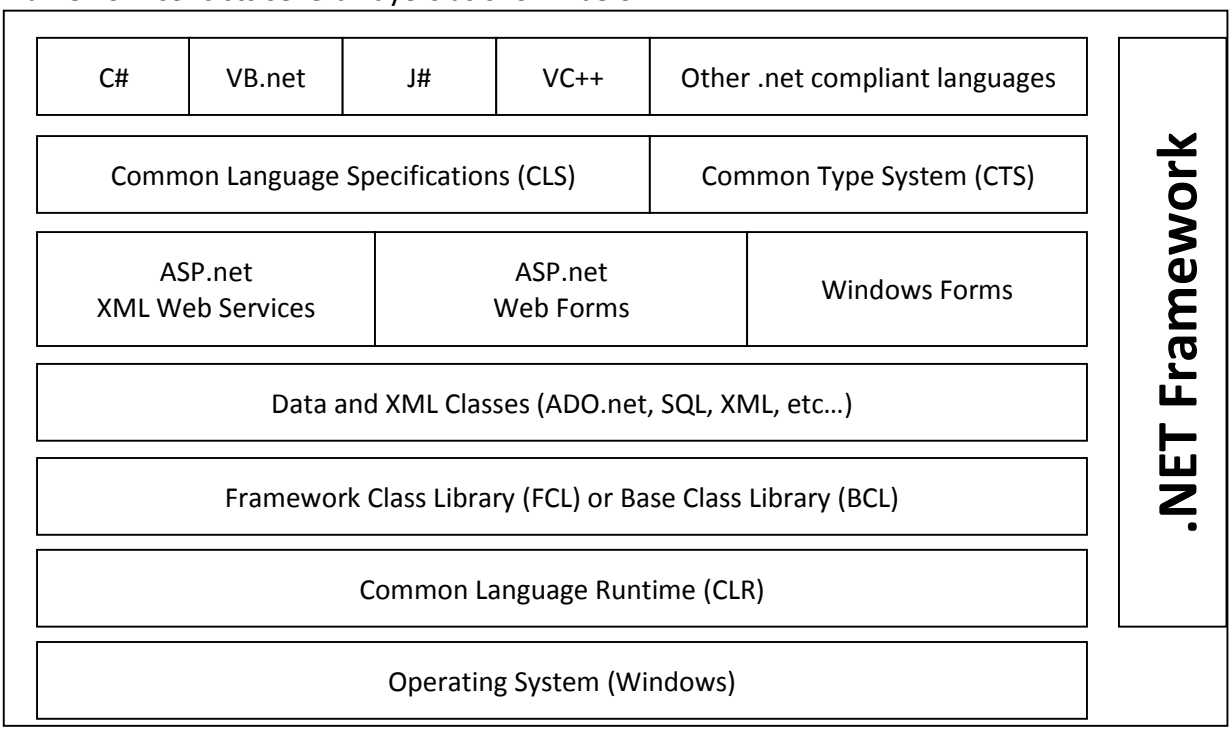


Figure1: .NET Framework Architecture Block Diagram

- .NET framework runs on different versions of windows operating system, starting from windows 98 to latest versions of windows 7 and windows 8.
- Operating system is not a part of .NET framework but generally it is shown as a base layer to indicate that .NET framework runs on operating system.

a) Common Language Runtime (CLR)

- The CLR is the execution engine for .NET applications and serves as the interface between .NET applications and the operating system.
- The CLR is the platform on which applications are hosted and executed.
- The CLR manages memory, Thread execution, Garbage Collection (GC), Exception Handling, Common Type System (CTS), code safety verifications, and other system services.

b) Framework Class Library (FCL) also known as Base Class Library (BCL)

- The .NET FCL is a collection of reusable types that is integrated with the CLR.
- The .NET FCL is object oriented.
- It contains more than 7000 classes and data types to read and write files, access databases, process XML, display a graphical user interface, draw graphics, use Web services, etc…
- The .NET FCL wraps much of the large, complex Win32 API into simpler .NET classes that can be used by

C# and other .NET programming languages.

c) Data and XML Classes (ADO.NET, SQL, XML, etc…)

- These classes extend the FCL to support data management and XML manipulation.
- ADO.NET provides improved support for the disconnected programming model.
- It also provides rich XML support.
- XML classes support various operations on XML data like searching, translations, modifying, etc…

d) Applications (Windows, Web, Web Services, etc…)

- Applications are the interface between users and computers that allows .NET to interact with the outside world.
- Commonly used application types are windows (Tally, MS Office, etc…), web (www.gtu.ac.in, www.google.co.in), web services (Internet payment systems), etc…

e) Common Language Specifications (CLS)

- CLS is a set of specifications to make languages as .NET compliant languages.
- CLS is an agreement among language designers and class library designers to use a common subset of basic language features that all languages have to follow.
- This is done in such a way, that programs written in any language (.NET compliant) can interoperate with other languages. This also can take full advantage of inheritance, polymorphism, exceptions, and other features.
- CLS makes use of CTS and CLR.

f) Common Type System (CTS)

- CTS describes how types are declared, used and managed in the runtime.
- It facilitates cross-language integration, type safety, and high performance code execution.
- Types are the mechanism by which code written in one programming language can talk to code written in a different programming language.
- CTS help developers to develop applications in different languages.
- The CTS also specifies the rules for type visibility and for access to the members of a type.

g) .NET compliant languages

- This is the top most layer, it consists of .NET compliant languages e.g. C#, VB.NET, J#, VC++, F#, etc…
- One of the imperative features of the .NET is the facility to program in multiple languages, which allows programmers to use their favorite languages.

2. Explain MSIL (Microsoft Intermediate Language).

- MSIL is a code that consists of CPU and platform independent set of instructions, which can be easily

converted to native code.

- It is also called as Intermediate Language.
- When you compile a .NET program, the CLR of .NET translates your source code into MSIL code that can be converted into CPU specific code with the help of JIT compiler.
- The MSIL code includes instruction to load, initialize and invoke methods on objects.
- It also includes the instructions for various operations on program code, such as arithmetic and logical operations, control flow, direct memory access, and exception handling.
- The program's source code is converted to MSIL code, which is equivalent to assembly language for CPU.
- The MSIL code is collected and assembled in the form of byte codes and is converted to a .NET assembly.
- The .NET assembly code is executed by the JIT compiler to generate native code.
- The native code is executed by the computer's processor.

3. Explain Common Language Runtime (CLR) OR Write a Short note on CLR.

- It works as a layer between operating system and the applications written in .NET languages that matches to the Common Language Specification (CLS).
- The main function of CLR is to convert the managed code into native code and then execute the code.

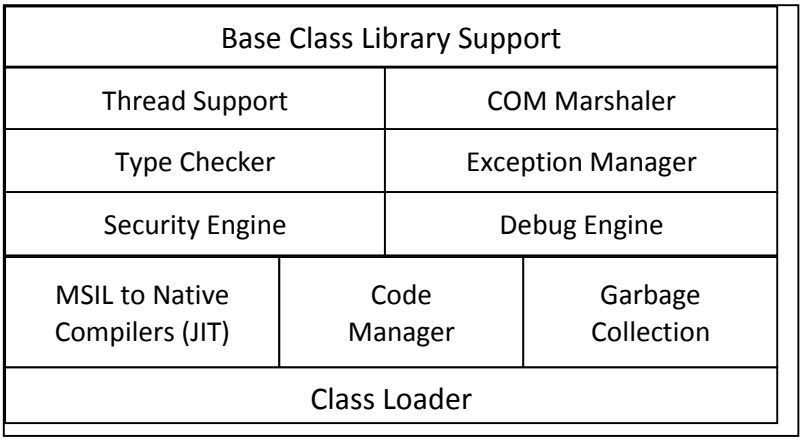


Figure2: CLR Modules

- The CLR's Just In Time (JIT) compilation converts Intermediate Language (MSIL) to native code on demand at run time.
- During the execution of the program, the CLR manages memory, thread execution, garbage collection (GC), exception handling, CTS, code safety verifications, and other system services.
- The CLR defines the CTS which is a standard type system used by all .NET languages.
- That means all .NET programming languages uses the same representation for common data types, thus CLR is a language-independent runtime environment.
- The CLR environment is also referred as a managed environment, because during the execution of a

program it also controls the interaction with the Operating System.

a) Thread Support

- The .NET Framework provides a number of threading and synchronization facilities to allow you to build high performance, multithreaded code.
- Your choice of threading approach and synchronization mechanism impacts application concurrency; hence, it also impacts scalability and overall performance.

b) COM Marshaler

- COM marshaler allows .NET applications to exchange data with COM applications.

c) Type Checker

- Type checker will verify types used in the application with CTS or CLS standards supported by CLR, this provides type safety.

d) Exception Manager

- The CLR supports structured exception handling to allow you to build robust, maintainable code.
- Use language constructs such as try/catch/finally to take advantage of structured exception handling.

e) Security Engine

- The .NET Framework provides code access security to ensure that code has the necessary permissions to perform specific types of operations such as accessing the file system, calling unmanaged code, accessing network resources, and accessing the registry.

f) Debug Engine

- Debug Manager Service will activate debugger utility to support line by line execution; the developer can make changes as per requirement without terminating application execution.

g) MSIL to Native Compilers (JIT)

- The just-in-time (JIT) compiler converts the Microsoft intermediate language (MSIL) into native machine code at run time.
- Methods that are never called, are not JIT-compiled.

h) Garbage Collection

- The garbage collector is responsible for allocating, freeing, and compacting memory.
- Garbage Collector will release memory of unused objects, this provides automatic memory management.

i) Code Manager

- Code manager invokes class loader for execution.

j) Class Loader

- The .NET Framework loader is responsible for locating and loading assemblies.

4. Explain Managed code and Unmanaged code

a) Manage Code

- Managed code is the code that is executed directly by the CLR.
- The applications that are created using managed code automatically have CLR services, such as type checking, security and automatic garbage collection.
- These services help to provide platform and language independence to managed code applications.
- The CLR compiles the applications to Microsoft Intermediate Language (MSIL) and not the machine code.
- This MSIL along with the metadata that describes the attributes, classes, and methods of the code reside in assembly.
- The compilation takes place in managed execution environment, which assures the working of the code.

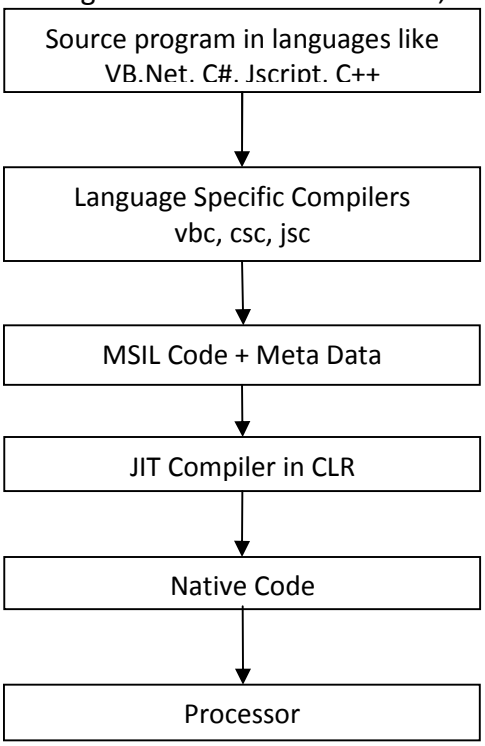


Figure3: Compilation of .Net code using CLR (Managed code)

- When you compile the code into managed code, the compiler converts the source code into MSIL, which is CPU-independent.

- Compilation of source code into MSIL, generates metadata.
- MSIL must be converted into CPU – specific code by the JIT compiler, before the execution of the code.t
- The metadata contains definition of types, member signature, the members in the code, and other details that the code uses at the time of execution.
- The runtime locates and extracts the metadata from the file during execution.
- While executing the application, a JIT compiler translates the MSIL into native code.
- After compiling the code is passed through a verification process that examines the MSIL and metadata to check whether the code is safe, such as it should be able to access only those memory, locations which it is authorized to access.

b)Unmanaged Code

- Unmanaged code directly compiles to the machine code and runs on the machine where it has been compiled.
- It does not have services, such as security or memory management, which are provided by the runtime.
- If code is not security-prone, it can be directly interpreted by any user, which can prove harmful.
- 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.
- Wrapper classes are of two types: CCW (COM Callable Wrapper) and RCW (Runtime Callable Wrapper).

5. Explain Common Language Specification (CLS).

- CLS is a set of basic language features that .NET Languages needed to develop Applications and Services, which are compatible with the .NET Framework.
- CLS ensures complete interoperability among applications, regardless of the language used to create the application.
- When there is a requirement to communicate objects written in different .NET Complaint languages, those objects must expose the features that are common to all the languages.
- CLS defines a subset of Common Type System (CTS) which describes a set of types that can use different .NET languages which ensure that objects written in different languages can interact with each other.
- Most of the members defined by types in the .NET Framework Class Library (FCL) are Common Language Specification (CLS) compliant Types.

6. Explain Namespace.

- Namespace is a grouping of logically related identifiers, classes, types etc…
- Namespace is used to avoid conflicts with the elements of an unrelated code which have the same

names.

- A namespace acts as a container—like a disk folder—for classes organized into groups usually based on functionality.
- All classes and types of .NET FCL are organized in namespaces.
- Implementing Namespaces in your own code is a good habit because it is likely to save you from problems later when you want to reuse some of your code.
- Namespaces don't correspond to file or directory names.

a) How to create namespace?

- Syntax: **namespace namespace_name { }**
- Example:

```
namespace GTU
{
    class Student
   {
       public static void Main()
      {
         Console.WriteLine("This is Student class");
      }
   }
   class Subject
  {
      public static void Main()
     {
         Console.WriteLine("This is Subject class");
     }
  }
}
```

b) How to use namespace?

- Example:

```
using GTU;
using System;
class Result
{
Subject s;
//Only class name, no need of namespace as we have used in the beginning
System.Windows.Forms.Button b = new System.Windows.Forms.Button ();
//Full qualifier name because we have not included namespace.
}
```

c) Alias for namespace:

- We can create alias of any namespace.
- Syntax: **using alias-name = namespace;**
- Example:

```
using Win = System.Windows;
class Test
{
        Win.SplashScreen ss = new Win.SplashScreen();
}
```

d) Nested Namespace

- Nested namespace is allowed in .NET. Like System namespace contains *System.Web* namespace.
- Example:

```
namespace Parent
{
  namespace Child
  {
    namespace Grandchild
    {
      class Test
      {
        public void ShowMessage()
        {
          Console.WriteLine ("This is a nested namespace!");
        }
      }
    }
  }
}
```

- Major Namespaces in .NET
  - ***System****:* The System namespace contains fundamental classes and base classes that define commonly-used value and reference data types, events and event handlers, interfaces, attributes, and processing exceptions.
  - It the root of all namespaces containing all other namespaces as subordinates.
  - It also contains the types that we felt to be the most fundamental and frequently use
  - ***System.Collections****:* Includes classes and interfaces that define various collection of objects such as list, queues, hash tables, arrays, etc.
  - ***System.Data***: Includes classes which lets us handle data from data sources.
  - ***System.Data.OleDb****:* Includes classes that support the OLEDB .NET provider.

- ***System.Data.SqlClient****:* Includes classes that support the SQL Server .NET provider.
- ***System.Diagnostics***: Includes classes that allow to debug our application and to step through our code.
- ***System.Drawing****:* Provides access to drawing methods.
- ***System.Globalization****:* Includes classes that specify culture-related information.
- ***System.IO****:* Includes classes for data access with Files.
- ***System.NET****:* Provides interface to protocols used on the internet.
- ***System.Reflection****:* Includes classes and interfaces that return information about types, methods and fields.
- ***System.Security****:* Includes classes to support the structure of common language runtime security system.
- ***System.Threading****:* Includes classes and interfaces to support multithreaded applications.
- ***System.Web****:* Includes classes and interfaces that support browser-server communication.
- ***System.Web.Services****:* Includes classes that let us build and use Web Services.
- ***System.Windows.Forms****:* Includes classes for creating Windows based forms.
- ***System.XML****:* Includes classes for XML support.

7. Explain Assembly Structure. Write code for create shared assembly and store assembly in GAC using tool.

- An assembly is a collection of types and resources that are built to work together and form a logical unit of functionality.
- Assemblies are the building blocks of .NET Framework applications.
- .NET assembly is the standard for components developed with the Microsoft.NET.
- Assembly physically exists as DLLs or EXEs.
- Assembly can include any file types like image files, text files etc. along with DLLs or EXEs.
- When you compile your source code, the EXE or DLL is generated and it is actually an assembly.
- One assembly may contain one or more files.

a) Assembly contains four major parts:

1. **Manifest: -** Additional information regarding assembly.
2. **Type metadata:-**Data about data or structure of classes.
3. **MSIL code: -** Containing business logics – Intermediate version of our program.
4. **Set of Resource: -** Information or resource of other assembly.

b) What is assembly manifest?

- Every assembly file contains information about itself. This information is called as Assembly Manifest.
- Assembly manifest is a data structure which stores information about an assembly.
- The information includes version information, list of files packed, and definition of types, security permissions, version control and metadata.

c) Types of Assembly

1) Static Assembly & Dynamic Assembly

- Static assemblies can include .NET Framework types (interfaces and classes), as well as resources for the assembly (bitmaps, JPEG files, resource files, and so on).
- Static assemblies are first stored on disk in portable executable (PE) files then run.
- Dynamic assemblies run directly from memory and are not saved to disk before execution.
- You can save dynamic assemblies to disk after they have executed.

2) Single File Assembly & Multi File Assembly

- A single file assembly contains all the required information (IL, Metadata, and Manifest) in a single package.
- A single file assembly does not implement version checking.
- A multi file assembly is spread in multiple files.
- A Multi file assembly is generally used for large applications with various types of resources
- A Multi file assembly implements version checking.
- Note: Multi file assemblies cannot be created with Visual Studio 2005 IDE for C# and VB.NET.
- You should use command line compilers to create multi-file assemblies.
- There are various options for grouping resources and code modules in to assemblies.

3) Private Assembly, Shared Assembly

- A Private assembly is used only by a single application.
- A Private assembly is stored in the application's directory or sub-directory.
- A shared assembly is normally used by more than one application.
- A shared assembly is stored in the global assembly cache (GAC) which is a repository of assemblies maintained by .NET runtime.
- The .NET Framework library is provided as shared assemblies.

4) Satellite Assembly

- A satellite assembly is an assembly that contains language specific resources.
- Sometimes, in an application, it is required to support multiple languages.
- There can be multiple satellite assemblies working side-by-side as they are installed in the language specific folders.
- Using satellite assemblies, you can place the resources for different languages in different assemblies, and the correct assembly is loaded into memory only if the user selects to view the application in that language.
- In general, assemblies should contain culture-neutral resources.
- If you want to localize your assembly (for example use different strings for different locales) you should use satellite assemblies.

d) Making a private assembly a global assembly

- A global assembly is a public assembly that is shared by multiple applications.
- Unlike private assembly, a global assembly is not copied to bin directory of each application that references it.

- Global assembly instead is placed in GAC (Global Assembly Cache) and it can be referenced anywhere within the system.
- So only one copy is stored, but many applications can use that single copy.
- In order to convert a private assembly to global assembly, we have to take the following steps.

1) Creating a strong name

- Any assembly that is to be placed in GAC must have a strong name. Strong name is a combination of **public key and private key**.
- The relationship between public and private keys are such, given one you cannot get the other, but any data that is encrypted with private key can be decrypted only with the corresponding public key.
- Take the following steps to invoke **SN** (Strong Name) tool to create strong name.

a) Go to command prompt using
Programs\Microsoft Visual Studio 2008\Visual Studio Tools\Visual Studio 2008 Command Prompt-> SDK Command prompt
Enter the following command.

```
sn -k HelloWorld.key
```

b) The above command writes private and public key pair into HelloWorld.key file. Location of HelloWorld.key File: C:\Program Files\Microsoft Visual Studio 9.0\VC

2) Associate strong name with assembly

- Once private and public keys are generated using SN tool, use the following procedure to sign **Hello World** with the key file.

1. Open **HelloWorld** project.
2. Select project properties using **Project -> HelloWorld properties**
3. Select **Signing** tab in project properties window
4. Check **Sign the assembly** check box
5. Select **HelloWorld.key** file using **Choose a strong name key file** combo box
6. Close properties window
7. Build the solution again using **Build->Build Solution**

- Now, **HelloWorld.dll** is associated with a public key and also digitally signed with private key**.**

3) Place assembly in GAC

- In order to make an assembly a global assembly, the assembly must be associated with a strong name and then placed in **Global Assembly Cache** (GAC).
- GAC is a folder with name **Assembly** in **windows** folder of your system. So, place **HelloWorld.dll** in GAC using **GACUTIL** tool as follows.
- Gacutil -i HelloWorld.dll
- After you install global assembly into GAC, you can see **HelloWorld.dll** in **windows/assembly**

- Once, you place an assembly in GAC, any reference to the assembly will not create a copy of the assembly in BIN directory of the application.
- Instead all application that reference the assembly use the same copy that is placed in GAC.

e) An assembly performs the following functions:

- **It contains code that the CLR executes.**
  - MSIL code in a portable executable (PE) file will not be executed if it does not have an associated assembly manifest.
- **It forms a security boundary.**
  - An assembly is the unit at which permissions are requested and granted.
- **It forms a type boundary.**
  - Every type's identity includes the name of the assembly in which it resides.
  - A type called MyType loaded in the scope of one assembly is not the same as a type called MyType loaded in the scope of another assembly.
- **It forms a reference scope boundary.**
  - The assembly's manifest contains assembly metadata that is used for resolving types and satisfying resource requests.
  - It specifies the types and resources that are exposed outside the assembly.
- **It forms a version boundary.**
  - The assembly is the smallest versionable unit in the CLR; all types and resources in the same assembly are versioned as a unit.
  - The assembly's manifest describes the version dependencies you specify for any dependent assemblies.
- **It forms a deployment unit.**
  - When an application starts, only the assemblies that the application initially calls must be present.
  - Other assemblies, such as localization resources or assemblies containing utility classes can be retrieved on demand.
  - This allows applications to be kept simple and thin when first downloaded.
  - It is the unit at which side-by-side execution is supported.

8. Explain Common Language Implementation.

- CLI is a standard that enables a .NET program written in any .NET compliant programming languages to be executed on any operating system using CLR, instead of a language specific runtime.
- It provides a virtual execution environment that uses a compiler convert the source code of a program in form of executable code, called bytecode.
- Later when program is executed its bytecode is translated to the native code, which is required by the machine code.

- CLI includes a component called the Common Type System (CTS).

a) Common Type System (CTS)

- The common type system defines how types are declared, used, and managed in the runtime, and is also an important part of the runtime's support for cross-language integration.
- The common type system performs the following functions:
- To enable cross-language integration, type safety, and high performance code execution.
- To provide an object-oriented model that supports the complete implementation of many programming languages.
- To define rules that languages must follow, which helps ensure that objects written in different languages can interact with each other.
- The CTS also defines the rules that ensure that the data types of objects written in various languages are able to interact with each other.
- The CTS also specifies the rules for type visibility and access to the members of a type, i.e. the CTS establishes the rules by which assemblies form scope for a type, and the CLR enforces the visibility rules.

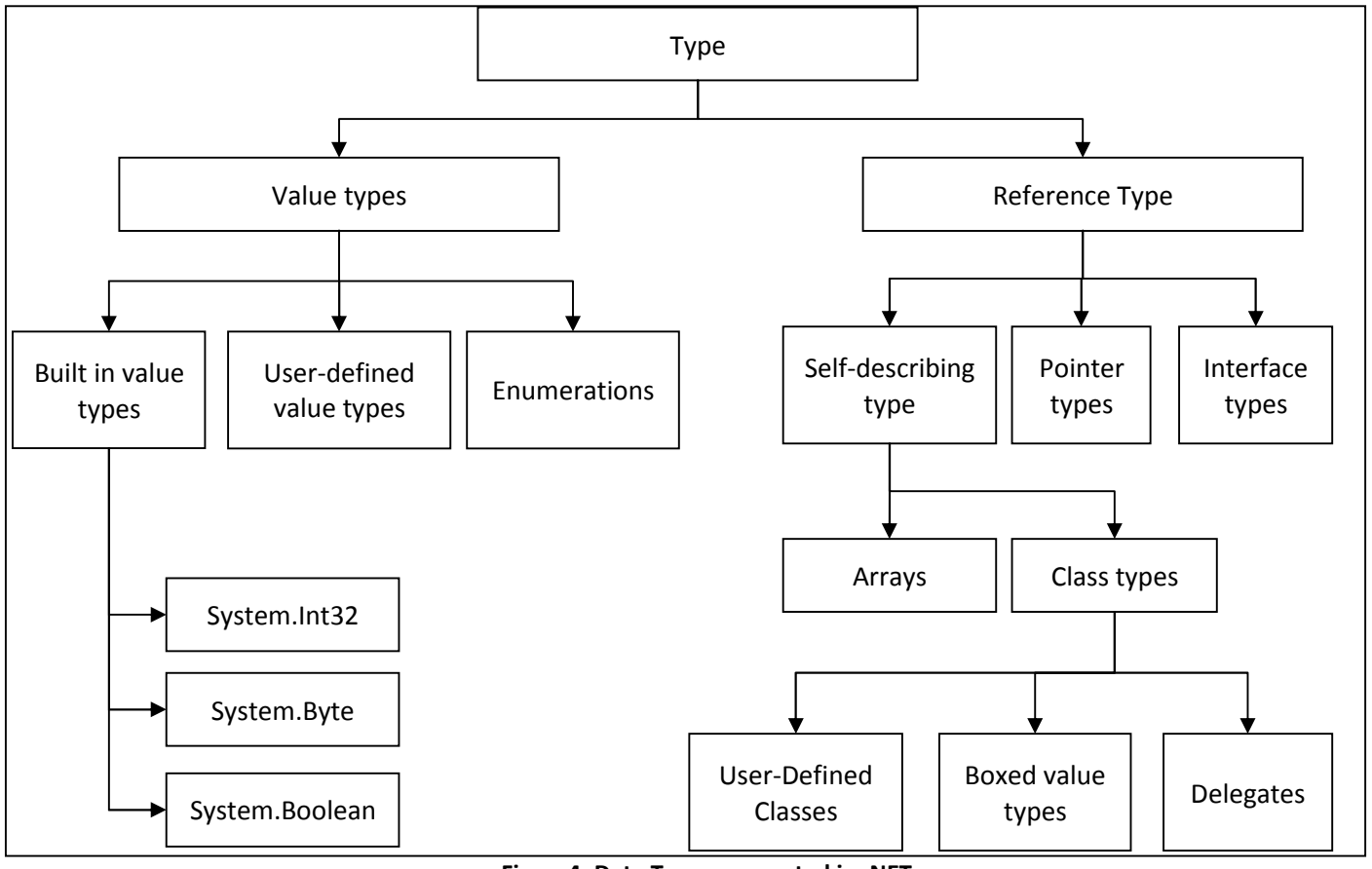


Figure4: Data Types supported in .NET

- The CTS defines the rules monitoring type inheritance, virtual methods and object lifetime.
- Languages supported by .NET can implement all or some common data types.
- The common type system supports two general categories of types, each of which is further divided into subcategories.

b) Value types

- Value types directly stores data and variables of value types are either stored on the stack or allocated inline in a structure.
- Value types can be built-in (*System.Int32*, *System.Boolean*), user-defined or enumerations.

c) Reference types

- Reference types store a reference to the value's memory address and are stored on the heap.
- Reference types can be self-describing types, pointer types, or interface types.
- The type of a reference type can be determined from values of self-describing types.
- Self-describing types are further split into arrays and class types. The class types are user-defined classes, boxed value types, and delegates.

9. Explain Metadata and intermediate Language.

- Metadata is a binary code that contains the self-description of your program.
- This binary code is stored in a portable executable (PE) file.
- When you compile a .NET program, the metadata is inserted in the one portion of PE file, while the program's code is converted to the IL code and inserted into other portion of file.
- When the program is executed, the CLR loads the metadata into memory and reference it to discover the information of your code.
- Metadata is the information that describes the types used in your code.
- Metadata eliminates the need for Interface Definition Language (IDL) files, header files, or any external method of component reference.
- Metadata stores the following information:
  - Description of the assembly.
    - Identity (name, version, culture, public key).
    - The types that are exported.
    - Other assemblies that this assembly depends on.
    - Security permissions needed to run.
  - Description of types.
    - Name, visibility, base class, and interfaces implemented.
    - Members (methods, fields, properties, events, nested types).
  - Attributes.
    - Additional descriptive elements that modify types and members

10. Explain Garbage Collection.

- The .Net Framework provides a new mechanism for releasing unreferenced objects from the memory (that is we no longer needed that objects in the program), this process is called Garbage Collection (GC).
- When a program creates an Object, the Object takes up the memory.
- Later when the program has no more references to that Object, the Object's memory becomes unreachable, but it is not immediately freed.
- The Garbage Collection checks to see if there are any Objects in the heap that are no longer being used by the application.
- If such Objects exist, then the memory used by these Objects can be reclaimed.
- So these unreferenced Objects should be removed from memory, then the other new Objects you create can find a place in the Heap.
- The reclaimed Objects have to be Finalized later.
- Finalization allows a resource to clean up after itself when it is being collected.
- This releasing of unreferenced Objects is happening automatically in .Net languages by the Garbage Collector (GC).
- The programming languages like C++, programmers are responsible for allocating memory for Objects they created in the application and reclaiming the memory when that Object is no longer needed for the program.
- In .Net languages there is a facility that we can call Garbage Collector (GC) explicitly in the program by calling *System.GC.Collect.*
- Following are the advantage of using Garbage Collector.
    - Allow us to develop an application without having worry to free memory.
    - Allocates memory for objects efficiently on the managed heap.
    - Reclaims the memory for no longer used objects and keeps the free memory for future allocations.
    - Provides memory safety by making sure that an object cannot use the content of another object.

11. Explain Versioning and Side-by-Side Execution

- Side-by-side execution is the ability to run multiple versions of an application or component on the same computer.
- You can have multiple versions of the common language runtime, and multiple versions of applications and components that use a version of the runtime, on the same computer at the same time.
- The below illustration shows several applications using two different versions of the runtime on the same computer.

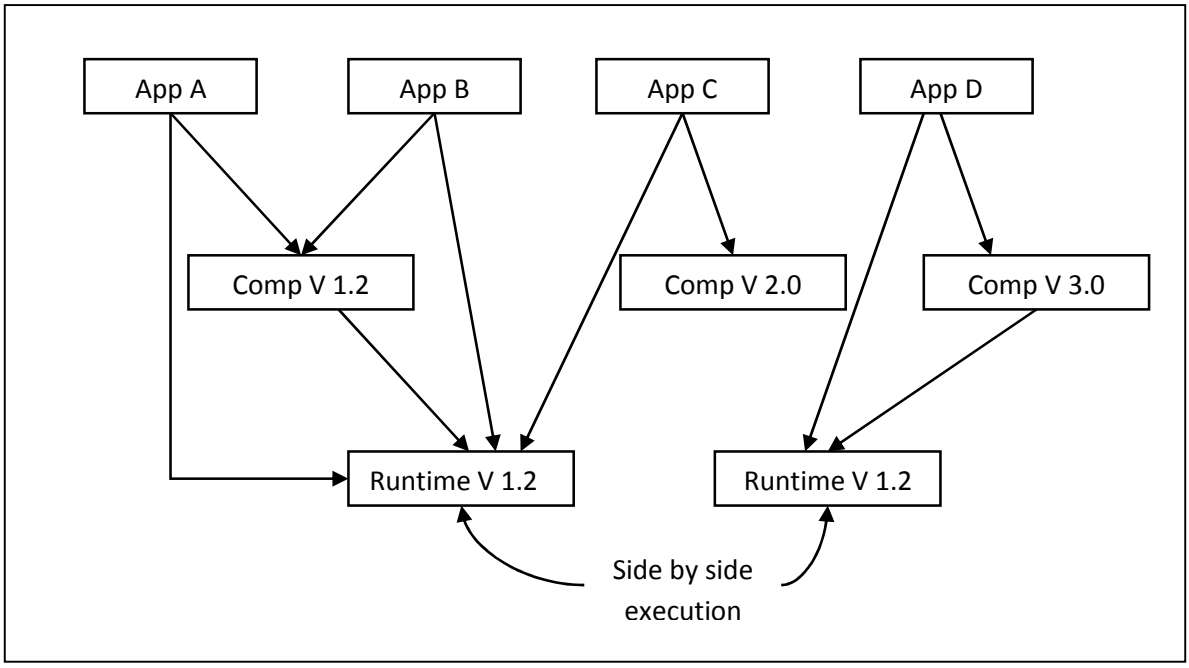


Figure5: Side by side execution

- Applications A, B, and C use runtime version 1.0, while application D uses runtime version 1.1.
- The .NET Framework consists of the common language runtime and a collection of assemblies that contain the API types.
- The runtime and the .NET Framework assemblies are versioned separately.
- Below are the benefits of side by side execution.
- Strong-named assemblies.
    - Side-by-side execution uses strong-named assemblies to bind type information to a specific version of an assembly.
    - This prevents an application or component from binding to an invalid version of an assembly.
    - Strong-named assemblies also allow multiple versions of a file to exist on the same computer and to be used by applications
- Version-aware code storage.
    - The .NET Framework provides version-aware code storage in the global assembly cache.
    - The global assembly cache is a computer-wide code cache present on all computers with the .NET Framework installed.
    - It stores assemblies based on version, culture, and publisher information, and supports multiple versions of components and applications.
- Isolation.
    - Using the .NET Framework, you can create applications and components that execute in isolation.
    - Isolation is an essential component of side-by-side execution.
    - It involves being aware of the resources you are using and sharing resources with confidence

among multiple versions of an application or component.
- Isolation also includes storing files in a version-specific way.

12. Explain end to DLL Hell.

- Earlier, before the release of .NET, the term DLL Hell, has been common in the world of software.
- DLL Hell refers to set of problems, which are caused when multiple applications try to share a common component, for instance a DLL file or a COM class.
- Suppose you install an application on your system.
- This application automatically updates a new version of the shared component that is not backward compatible with a version already on the machine.
- Although your new application that has just been installed works well, but existing application that depended on previous version of the shared component might no longer work.
- This is because the version information of different components of an application is not recorded by the system.
- Therefore, changes made by an application on the system affect other applications of the machine
- To end the problems of DLL Hell, Microsoft has introduced .NET along the concept of run once – run forever.
- This means that if a .NET application is installed and works then it will work forever, regardless of what other application, including .NET as well as non .NET are installed on the system.

Explain Constructor and Destructors in C#.

Constructors

- A class **constructor** is a special member function of a class that is executed whenever we create new objects of that class.
- A constructor will have exact same name as the class and it does not have any return type.
- By default C# creates default constructor internally.
- A default constructor does not have any parameter.
- A class can have any number of constructors.
- A constructor doesn't have any return type even void.
- Within a class you can create only one static constructor.
- Constructor with no arguments and nobody is called default constructor.
- If you need a constructor which can have parameters. Such constructors are called parameterized constructors. This technique helps you to assign initial value to an object at the time of its creation.
- The following example demonstrates the concept of Constructor:

```
using System;
using System.Collections;
namespace Constructor
{
    class Square
    {
        private int Side;
        //Default Constructor
        public Square()
        {
            this.Side = 1;
        }
        //Paramerized Constructor
        public Square(int side)
        {
            //A Constructor is used to initilize private fields of a class
            this.Side = side;
        }
        public int Area()
        {
            return this.Side * this.Side;
        }
    }
//Using Class in the program
    class MainClass
    {
        public static void Main()
        {
            //Calling Default Constructor
            Square squreobject = new Square();
            int Area = squreobject.Area();
            Console.WriteLine(Area);
```

```
            //Calling Parametrized Constructor
            Square MySquare = new Square(10);
            int myarea = MySquare.Area();
            Console.WriteLine(myarea);
            Console.Read();
        }
    }
}
```

Output:

1
100

Destructor

- A **destructor** is a special member function of a class that is executed whenever an object of its class goes out of scope.
- A destructor will have exact same name as the class prefixed with a tilde (~) and it can neither return a value nor can it take any parameters.
- Destructor can be very useful for releasing resources before coming out of the program like closing files, releasing memories etc. Destructors cannot be inherited or overloaded.
- A class can only have one destructor.
- Destructors cannot be inherited or overloaded.
- A destructor does not take modifiers or have parameters.
- The following example demonstrates the concept of Destructor:+

```
using System;
using System.Collections;
namespace Destructor
{
    class Example
    {
        public Example()
        {
            Console.WriteLine("Constructor")
        }
        ~Example()
        {
            Console.WriteLine("Destructor");
            Console.ReadLine();
        }
    }
    class Program
    {
        static void Main()
        {
            Example x = new Example();
        }
    }
}
```

Explain function overloading in C#.

Function Overloading

- The process of creating more than one method in a class with same name or creating a method in derived class with same name as a method in base class is called as method overloading.
- C# allows us to define multiple functions with the same name differing in the number type and order of arguments. This is termed as function overloading.
- There is no need to use any keyword while overloading a method either in same class or in derived class.
- While overloading methods, a rule to follow is the overloaded methods must differ either in number of arguments they take or the data type of at least one argument.
- In case of method overloading, compiler identifies which overloaded method to execute based on number of arguments and their data types during compilation itself. Hence method overloading is an example for compile time polymorphism.
- The following example demonstrates the concept of Function Overloading:

```
using System;
namespace Function_Overloading
{
    class Class1
    {
        public int Sum(int A, int B)
        {
            return A + B;
        }
        public float Sum(int A, float B)
        {
            return A + B;
        }
    }
    class Class2 : Class1
    {
        public int Sum(int A, int B, int C)
        {
            return A + B + C;
        }
    }
    class MainClass
    {
        static void Main()
        {
            Class2 obj = new Class2();
            Console.WriteLine(obj.Sum(10, 20));
            Console.WriteLine(obj.Sum(10, 15.70f));
            Console.WriteLine(obj.Sum(10, 20, 30));
            Console.Read();
      }
```

```
    }
}
```

Output: 30 25.7 60

- As shown in our example we has created two classes Class1 and Class2, now in our main method we has created object of Class2.
- Now with the help of that object we are calling method with different number of arguments, while passing the arguments if two methods have same number of arguments but data type of arguments are different then while passing the value we have to tell the compiler, that which type of the argument you are passing, as shown in our example "(10, 15.70f)", here we are telling the compiler that which method have the second parameter as a float that method we have to call.

Explain operator overloading in C#.

- Overloaded operators are functions with special names the keyword operator followed by the symbol for the operator being defined.
- Similar to any other function, an overloaded operator has a return type and a parameter list.
- For example, go through the following function:

```
public static Box operator +(Box b, Box c)
{
    Box box = new Box();
    box.length = b.length + c.length;
    box.breadth = b.breadth + c.breadth;
    box.height = b.height + c.height;
    return box;
}
```

- The above function implements the addition operator (+) for a user-defined class Box.
- It adds the attributes of two Box objects and returns the resultant Box object.
- The following program shows the complete implementation of Operator overloading:

```
using System;
namespace Operator_Overloading
{
    class Box
    {
        private double length,breadth,height;
              // Length, Breadth, Height of a box
        public double getVolume()
        {
            return length * breadth * height;
        }
        public void setLength(double len)
        {
            length = len;
        }
        public void setBreadth(double bre)
        {
            breadth = bre;
        }
       public void setHeight(double hei)
        {
            height = hei;
        }
        // Overload + operator to add two Box objects.
        public static Box operator +(Box b, Box c)
        {
            Box box = new Box();
            box.length = b.length + c.length;
            box.breadth = b.breadth + c.breadth;
            box.height = b.height + c.height;
            return box;
        }
    }
    class Tester
    {
        static void Main(string[] args)
        {
            Box Box1 = new Box();   // Declare Box1 of type Box
            Box Box2 = new Box();   // Declare Box2 of type Box
            Box Box3 = new Box();   // Declare Box3 of type Box
            double volume = 0.0;    // Store the volume of a box here
            // box 1 specification
            Box1.setLength(6.0);
            Box1.setBreadth(7.0);
            Box1.setHeight(5.0);
            // box 2 specification
            Box2.setLength(12.0);
            Box2.setBreadth(13.0);
            Box2.setHeight(10.0);
            // volume of box 1
            volume = Box1.getVolume();
            Console.WriteLine("Volume of Box1 : {0}", volume);
            // volume of box 2
            volume = Box2.getVolume();
            Console.WriteLine("Volume of Box2 : {0}", volume);
            // Add two object as follows:
            Box3 = Box1 + Box2;
            // volume of box 3
```

```
            volume = Box3.getVolume();
            Console.WriteLine("Volume of Box3 : {0}", volume);
            Console.ReadKey();
        }
    }
}
```

Output: 210 1560 5400

Explain Concept of Modifiers in C#.

- Access modifiers are keywords used to specify the declared accessibility of a member or a type.
- Access modifiers are an integral part of object-oriented programming.
- They support the concept of encapsulation, which promotes the idea of hiding functionality.
- Access modifiers allow you to define who does or doesn't have access to certain features.

Private Access modifiers

- Private members are accessible only within the body or scope of the class or the struct in which they are declared.
- We can declare private access modifier to a class, method, property, structure, interfaces. However if we do not specify any access modifier to a class, method, property, structure it will be assumed as private member only.

```
public class Employee
{
    private void DisplayEmployeeDetails()
    {
        Console.WriteLine("Employee Name --> Onlinebuff and Employee Code --> 009");
    }
}
```

- In above the code we have class "Employee" with "private" method "DisplayEmployeeDetails" which displays employee details.

Public Access Modifier:

- Public members are accessible anywhere in program or application.
- There is no restriction to access public access modifiers.
- We can declare public access modifier to a class, method, property, structure, interfaces.

```
public class Employee
{
    Public void DisplayEmployeeDetails()
    {
        Console.WriteLine("Employee Name --> Onlinebuff and Employee Code -->
009");
    }
}
```

- In above the code we have class "Employee" with "public" method "DisplayEmployeeDetails" which displays employee details.

Protected Access Modifier:

- Protected members are accessible within the body or scope of the class or the struct in which they are declared and if protected member is declared in base class (Parent class) then it is also accessible in derived class (child class) only if access takes place through derived class type.
- We can declare protected access modifier to a class, method, property, structure, interfaces.

```
public class Employee
{
    Protected void DisplayEmployeeDetails()
    {
        Console.WriteLine("Employee Name --> Onlinebuff and Employee Code --> 009");
    }
}
```

- In above code we have class "Employee" with "protected" method "DisplayEmployeeDetails" which displays employee details.

Internal Access Modifier:

- Internal members are accessible within the same namespace scope in which it is declared or it is accessible in same assembly or same project.
- We can declare internal access modifier to a class, method, property, structure, interfaces.

```
public class Employee
{
    internal void DisplayEmployeeDetails()
    {
        Console.WriteLine("Employee Name --> Onlinebuff and Employee Code --> 009");
    }
}
```

- In above code we have class "Employee" with "internal" method "DisplayEmployeeDetails" which displays employee details.

Protected Internal Access Modifier:

- Protected Internal members are accessible within the same namespace scope in which it is declared or it is accessible in same assembly or same project and it is also accessible in another assembly or project if protected internal member is declared in base classof first assembly then it is also accessible in derived class (child class) of other assembly if access takes place through derived class type.
- We can declare protected internal access modifier to a class, method, property, structure, interfaces.

```
public class Employee
{
    Protected internal void DisplayEmployeeDetails()
    {
        Console.WriteLine("Employee Name --> Onlinebuff and Employee Code --> 009");
    }
}
```

- In above code we have class "Employee" with "internal" method "DisplayEmployeeDetails" which displays employee details.

Explain Concept of Properties in C#.

- Properties are named members of classes, structures, and interfaces. Member variables or methods in a class or structures are called Fields.
- Properties are an extension of fields and are accessed using the same syntax. They use accessors through which the values of the private fields can be read, written or manipulated.
- Properties do not name the storage locations. Instead, they have accessors that read, write, or compute their values.
- For example, let us have a class named Student, with private fields for age, name, and code. We cannot directly access these fields from outside the class scope, but we can have properties for accessing these private fields.
- The following example demonstrates use of properties:

```
using System;
namespace Properties
{
    class Student
    {
        private string code = "N.A";
        private string name = "not known";
        // Declare a Code property of type string:
        public string Code
        {
            get
            {
                return code;
            }
            set
            {
                code = value;
            }
        }
        // Declare a Name property of type string:
        public string Name
        {
            get
            {
                return name;
            }
            set
            {
                name = value;
            }
        }
        public override string ToString()
        {
            return "Code = " + Code + ", Name = " + Name;
        }
    }
    class ExampleDemo
    {
        public static void Main()
        {
             // Create a new Student object:
```

```
            Student s = new Student();
            // Setting code and name of the student
            s.Code = "001";
            s.Name = "Zara";
            Console.WriteLine("Student Info: {0}", s);
            Console.ReadKey();
        }
    }
}
```

Output: Student Info: Code = 001, Name = Zara

Explain Concept of Indexers in C#.

- Indexer Concept is object act as an array.
- Indexer is an object to be indexed in the same way as an array.
- Indexer modifier can be private, public, protected or internal.
- The return type can be any valid C# types.
- Indexers in C# must have at least one parameter. Else the compiler will generate a compilation error.
- The following example demonstrates use of indexers:

```
using System;
using System.Collections;
namespace Indexers
{
    class ParentClass
    {
        private string[] range = new string[5];
        public string this[int indexrange]
        {
            get
            {
                return range[indexrange];
            }
            set
            {
                range[indexrange] = value;
            }
        }
    }
    /* The Above Class just act as array declaration using this pointer */
    class childclass
    {
        public static void Main()
        {
            ParentClass obj = new ParentClass();
            /* The Above Class ParentClass  create one object name is obj */
            obj[0] = "ONE";
            obj[1] = "TWO";
            obj[2] = "THREE";
            obj[3] = "FOUR ";
            obj[4] = "FIVE";
             Console.WriteLine("{0}\n,{1}\n,{2}\n,{3}\n,{4}\n", obj[0], obj[1],
             obj[2], obj[3], obj[4]);
```

```
            Console.ReadLine();
        }
    }
}
```

Output :

ONE
,TWO
,THREE
,FOUR
,FIVE

Properties	**Indexers**
Identified by its name.	Identified by its signature.
Accessed through a simple name or a member access.	Accessed through an element access.
Can be a static or an instance member.	Must be an instance member.
A get accessor of a property has no parameters.	A get accessor of an indexer has the same formal parameter list as the indexer.
A set accessor of a property contains the implicit value parameter.	A set accessor of an indexer has the same formal parameter list as the indexer, in addition to the value parameter.

Explain Attributes & Reflection API in C# with Example.

Attributes

- An attribute is a declarative tag that is used to convey information to runtime about the behaviors of various elements like classes, methods, structures, enumerators, assemblies etc.
- You can add declarative information to a program by using an attribute.
- A declarative tag is depicted by square ([ ]) brackets placed above the element it is used for.
- Attributes are used for adding metadata, such as compiler instruction and other information such as comments, description, methods and classes to a program.
- The .Net Framework provides two types of attributes: the pre-defined attributes and custom built attributes.
- Syntax for specifying an attribute is as follows:

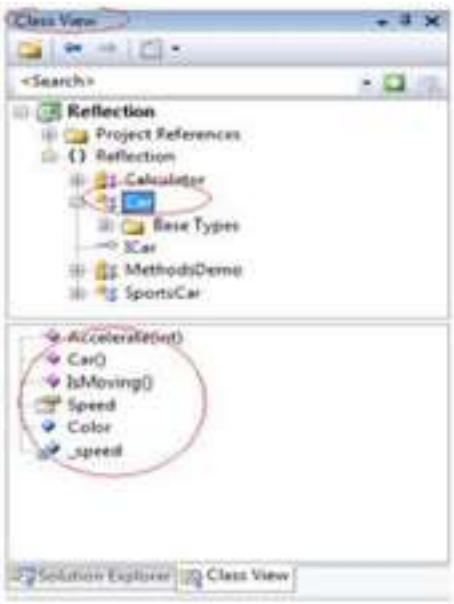
```
[attribute(positional_parameters, name_parameter = value, ...)]
{
        Element;
}
```

- Name of the attribute and its values are specified within the square brackets, before the element to which the attribute is applied.

Positional parameters specify the essential information and the name parameters specify the optional information.

Reflections

- Reflection objects are used for obtaining type information at runtime.
- .NET Framework's Reflection API allows fetching type (assembly) information at runtime programmatically.
- The classes that give access to the metadata of a running program are in theSystem.Reflection namespace.
- The System.Reflection namespace contains classes that allow you to obtain information about the application and to dynamically add types, values, and objects to the application.
- Applications of Reflections are as follows:
    - It allows view attribute information at runtime.
    - It allows examining various types in an assembly and instantiate these types.
    - It allows late binding to methods and properties.
    - It allows creating new types at runtime and then performs some tasks using those types.
- At runtime, the Reflection mechanism uses the PE file to read information about the assembly.
- Reflection enables us to use code that is not available at compile time.
- .NET Reflection allows an application to collect information about itself and also to manipulate on itself.
- It can be used effectively to find all types in an assembly and/or dynamically invoke methods in an assembly. This includes information about the type, properties, methods, and events of an object.
- With Reflection, we can dynamically create an instance of a type, bind the type to an existing object, or get the type from an existing object and invoke its methods or access its fields and properties.
- We can also access attribute information using Reflection.
- Using Reflection, we can get any kind of information which we can see in a class viewer; for example, information on the methods, properties, fields, and events of an object.

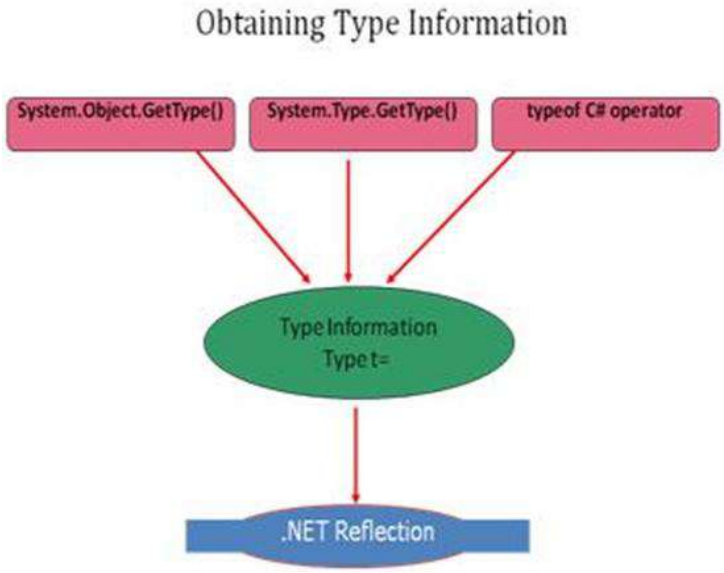


- The **System.Reflection** namespace contains the classes and interfaces that provide a managed view of loaded types, methods, and fields, with the ability to dynamically create and invoke types; this process is known as Reflection in .NET framework.
- Commonly Used classes

Class	Description
Assembly	Represents an assembly, which is a reusable, versionable, and self-describing building block of a Common Language Runtime application. This class contains a number of methods that allow you to load, investigate, and manipulate an assembly.
Module	Performs Reflection on a module. This class allows you to access a given module within a multi-file assembly.
AssemblyName	This class allows you to discover numerous details behind an assembly's identity. An assembly's identity consists of the following: Simple name Version number Cryptographic key pair Supported culture
EventInfo	This class holds information for a given event. Use the EventInfo class to inspect events and to bind to event handlers.
FieldInfo	This class holds information for a given field. Fields are variables defined in the class. FieldInfo provides access to the metadata for a field within a class, and provides dynamic set and get functionality for the field. The class is not loaded into memory until Invoke or get is called on the object.
MemberInfo	The MemberInfo class is the abstract base class for classes used to obtain information about all members of a class (constructors, events, fields, methods, and properties).
MethodInfo	This class contains information for a given method.
ParameterInfo	This class holds information for a given parameter.
PropertyInfo	This class holds information for a given property.

The System.Type Class

- The **System.Type** class is the main class for the .NET Reflection functionality and is the primary way to access metadata. The System.Type class is an abstract class and represents a type in the Common Type System (CLS).
- It represents type declarations: class types, interface types, array types, value types, enumeration types, type parameters, generic type definitions, and open or closed constructed generic types.
- Use the members of Type to get information about a type declaration, such as the constructors, methods, fields, properties, and events of a class, as well as the module and the assembly in which the class is deployed.
- There are three ways to obtain a Type reference:



Reflecting on Methods

- GetMethod() returns a reference to a System.Reflection.MethodInfo object, which contains details of a method. Searches for the public method with the specified name.
- GetMethods() returns an array of such references. The difference is that GetMethods() returns details of all the methods, whereas GetMethod() returns details of just one method with a specified parameter list.

Reflecting on Fields and Properties

- The behavior of Type.GetField() and Type.GetFields() is exactly similar to the GetMethods() methods, except Type.GetField() returns a reference of System.Reflection.MethodInfo and Type.GetFields() returns a reference of a System.Reflection.MethodInfo array. Similarly, Type.GetProperty() and Type.GetProperties() too.

Reflecting on Implemented Interfaces

- GetInterfaces() returns an array of System.Types! This should make sense given that interfaces are, indeed, types.

Reflecting on Method Parameters and Return Values

- To play with method parameters and return types, we first need to build a MethodInfo[] array using the GetMethods() function.
- The MethodInfo type provides the ReturnType property and the GetParameters() method for these very tasks.

Reflecting on Constructor

- The GetConstractors() function returns an array of ConstractorInfo elements which we can use to get more class constructor information.

The following example demonstrates use of reflection:

```
using System;
using System.Reflection;
namespace Reflection_API
{
    public class MyClass
    {
        public virtual int AddNumb(int numb1, int numb2)
        {
            int result = numb1 + numb2;
            return result;
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
              Console.WriteLine("\nReflection.MethodInfo");
              MyClass myClassObj = new MyClass();


        Type myTypeObj = myClassObj.GetType();
            MethodInfo myMethodInfo = myTypeObj.GetMethod("AddNumb");
            object[] mParam = new object[] { 5, 10 };
            Console.Write("\nFirst method - " + myTypeObj.FullName + " returns "
              +myMethodInfo.Invoke(myClassObj, mParam) + "\n");
        }
    }
}
```

Explain Console Input and Output in C# with Example.

- The Console is a window of the operating system through which users can interact with system programs of the operating system or with other console applications.
- The interaction consists of text input from the standard input or text display on the standard output. These actions are also known as input-output operations.
- Functions of Console class are

ReadLine()	This function is used to accept a string from input stream
Read()	This function is used to accept a string from input stream
ReadKey()	This function is used to wait program for a key press and it prevent the screen from running and closing quickly.

Write()	This function is used to print the output stream
WriteLine()	This function is used to print the output stream in new line

Example using above functions:

```
using System;

namespace ConsoleApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            int num;
            Console.WriteLine("Kindly enter value of num :");
            num = Convert.ToInt32(Console.ReadLine());
            Console.WriteLine("{0}{1}","The value of num is ",num);
            Console.ReadKey();
        }
    }
}
```

Output :

Kindly enter value of num :

100

The value of num is 100

Formatting Numerical Data

- If you require more elaborate formatting for numerical data, each placeholder can optionally contain various format characters. Following table shows the most common formatting options.

Format	Character Meaning in Life
C or c	Used to format currency. By default, the flag will prefix the local cultural symbol (a dollar sign [$] for U.S. English).
D or d	Used to format decimal numbers. This flag may also specify the minimum number of digits used to pad the value.
E or e	Used for exponential notation. Casing controls whether the exponential constant is uppercase (E) or lowercase (e).
F or f	Used for fixed-point formatting. This flag may also specify the minimum number of digits used to pad the value.
G or g	Stands for general. This character can be used to format a number to fixed or exponential format.
N or n	Used for basic numerical formatting (with commas).
X or x	X or x Used for hexadecimal formatting. If you use an uppercase X, your hex format will also contain uppercase characters.

Example demonstrates formatted output of Numeric Data:

```
using System;
namespace ConsoleApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("The value 99999 in various formats:");
            Console.WriteLine("c format: {0:c}", 99999);
            Console.WriteLine("d9 format: {0:d9}", 99999);
            Console.WriteLine("f3 format: {0:f3}", 99999);
            Console.WriteLine("n format: {0:n}", 99999);
            Console.WriteLine("E format: {0:E}", 99999);
            Console.WriteLine("e format: {0:e}", 99999);
            Console.WriteLine("X format: {0:X}", 99999);
            Console.WriteLine("x format: {0:x}", 99999);
            Console.ReadKey();
        }
    }
}
```

Output :

The value 99999 in various formats:
c format: $99,999.00
d9 format: 000099999
f3 format: 99999.000
n format: 99,999.00
E format: 9.999900E+004
e format: 9.999900e+004
X format: 1869F
x format: 1869f

Format Strings Data

- Console.Write() or Console.WriteLine() can be used to align string to the right or to the left. To align string to the left (spaces on the right) use formatting pattern with comma (,) followed by a negative number of characters: ("{0,–10}", text). To right alignment use a positive number: ("{0,10}", text)
- Following example shows how to format text to the table. Values in the first and second column are aligned to the left and the third column is aligned to the right.

Example demonstrates the formatted output of String Data:

```
using System;
namespace  Format_String
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("-------------------------------");
```

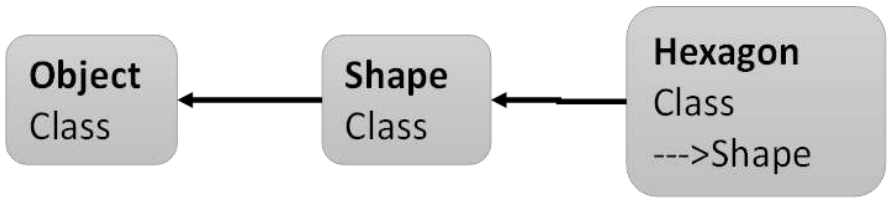
```
            Console.WriteLine("First Name | Last Name  |   Age");
            Console.WriteLine("-------------------------------");
            Console.WriteLine("{0,-10} | {1,-10} | {2,5}", "Amit", "Gandhi", 51);
            Console.WriteLine("{0,-10} | {1,-10} | {2,5}", "Suresh", "Parker", 104);
            Console.WriteLine("{0,-10} | {1,-10} | {2,5}", "Mahesh", "Mheta", 44);
            Console.WriteLine("-------------------------------");
            Console.ReadKey();

        }
    }
}
```

Output :

```
-------------------------------
First Name | Last Name  |   Age
-------------------------------
Amit       | Gandhi     |    51
Suresh     | Parker     |   104
Mahesh     | Mheta      |    44
-------------------------------
```

What is Inheritance? Explain with Example in C#.

- C# strongly supports the concept of OOPs in terms of reusability.
- The mechanism of deriving a new class from an old one is called inheritance (or derivation).
- The old class is referred to as the base class and new one is called the derived class or subclass.
- A class can also inherit properties from more than one class or from more than one level.
- For example, “A Hexagon is-a Shape that is-an Object.” When you have classes related by this form of inheritance, you establish “is-a” relationships between types. The “is-a” relationship is termed inheritance.



Inheritance - The “is-a” relationship

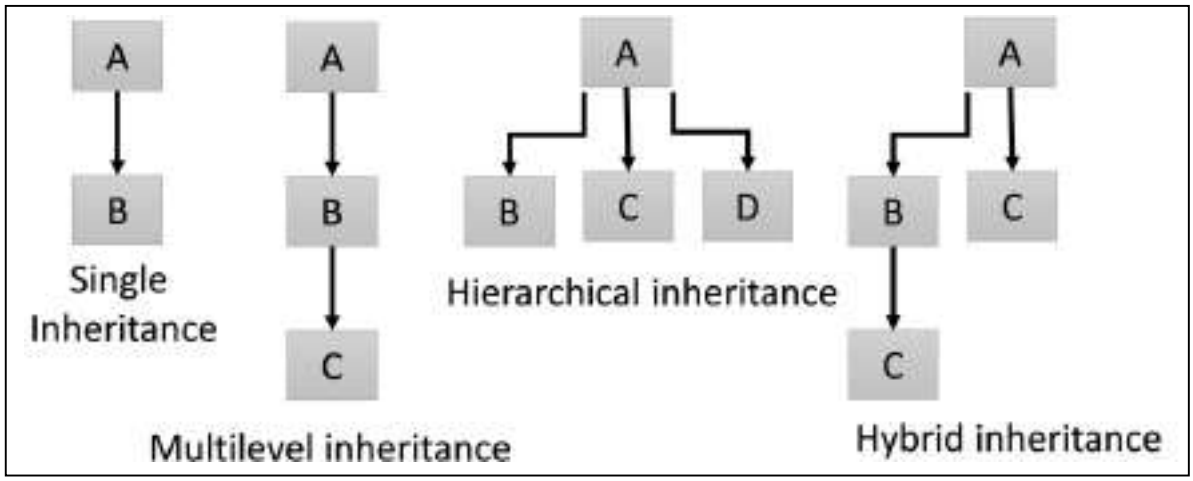
- If you want to declare that a class derives from another class, use the following syntax:

```
class MyDerivedClass : MyBaseClass
{
    // functions and data members here
}
```

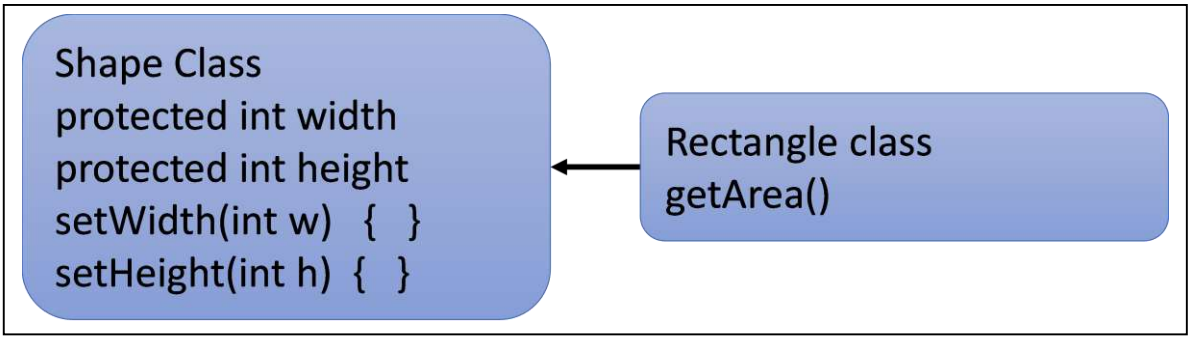
- **Important characteristics of inheritance include:**

1. A derived class extends its base class. That is, it contains the methods and data of its parent class, and it can also contain its own data members and methods.
2. The derived class cannot change the definition of an inherited member.
3. Constructors and destructors are not inherited. All other members of the base class are inherited.
4. The accessibility of a member in the derived class depends upon its declared accessibility in the base class.
5. A derived class can override an inherited member.

- **There are basic four types of inheritance.**
  1. Single Inheritance: A derived class with only one base class, is called single inheritance. As see in figure Class A is parent class and Class B is derived class.
  2. Multilevel inheritance: The mechanism of deriving a class from another derived class is called multilevel inheritance. As see in figure Class A is parent class of Class B and Class B is parent class of Class C.
  3. Hierarchical inheritance: One class may be inherited by more than one classes. This process is known as hierarchical inheritance. As see in figure Class A is parent class and Class B, Class C and Class D are derived classes.
  4. Hybrid inheritance: there could be situations where we need to apply two or more types of inheritance to design a program. As see in figure Class A is parent class of Class B and Class C, Class B is parent class of Class C. So applying two inheritance type hierarchical inheritance and multilevel inheritance.
- **Unlike C++, the C# does not support multiple inheritance.** For that in C# have facility of Interface. A class can be derived from more than Interface, which means that it can inherit data and functions from multiple base Interfaces. Interface inheritance means that a type inherits only the signatures/definitions of the functions, but does not inherit any implementations.



Let's take an example of "Rectangle is-a Shape" by using inheritance.



Consider a base class Shape and its derived class Rectangle:

```
using System;
namespace InheritanceApplication
{
    class Shape
    {
        public void setWidth(int w)
        {
            width = w;
        }
        public void setHeight(int h)
        {
            height = h;
        }
        protected int width;
        protected int height;
    }
    // Derived class
    class Rectangle : Shape
    {
        public int getArea()
        {
            return (width * height);
        }
    }
    class RectangleTester
    {
        static void Main(string[] args)
        {
            Rectangle Rect = new Rectangle();
            Rect.setWidth(5);
            Rect.setHeight(7);
            // Print the area of the object.
            Console.WriteLine("Total area: {0}", Rect.getArea());
            Console.ReadKey();
        }
    }
}
```

Output : Total area: 35

Features of .NET Programming Language

C#.NET is an Object-Oriented computer programming language that has been implemented on the .NET Framework. C#.NET also has some language fundamentals that includes identifiers, keywords, variables, constants, and control structures.

In C#.NET, a statement consists of the following parts:

- Keywords: Words reserved for performing specific tasks in C#.NET applications.
- Operators: symbols, such as +, - and * to perform addition, subtraction and multiplication respectively.
- Variables: stores the value used in program.
- Literal values: Refers to the simple value, like 5 or Hello World.
- Constants: similar to variable, except that the values contained in constants do not change.
- Expressions: Combinations of term and/or keywords that yield a value. For example, if the variable marks hold the value 58, then expression marks +4 yield the value 62.
- Declaration statement: help to create and name a variable, constant, or procedure and specify a data type, such as Boolean, Byte, Char and Date, for the elements.
- Executable statement: Performs an action. It can execute a method or loop through the code repetitively or act as an assignment statement that assigns a value or expression to a variable or constant, such as subject="Maths" and marks=82.

Identifiers and keywords:

- Every programming language uses a set of predefined words in coding. These words are language-specifics and are known as keywords, which provide specific predefine meaning for the compiler. C#.NET provides two types of keywords: reserved and unreserved. Reserved keywords are those keywords that cannot be as names for programming elements, such as variables, methods, and classes which unreserved keywords are those keywords that keywords that can be used as names for programming elements. However, you should avoid using keywords as the names of variables and procedures, as this leads to subtle errors in the code and makes it difficult to understand.
  - ✓ Example of Reserved keywords: bool, If, Each, Get, GoTo etc.
  - ✓ Example of Unreserved Keywords: Compare, Join, Mid etc.

Statement syntax:

- Syntaxes are conventions or rules that need to be followed while declaring or defining a variable, functions. Program cannot compile with mistake to write in the syntax.

Variables and constants:

- Declaring variable in C#.NET

  < access modifiers > <data type> <name> = <value>;
  For example:
  private string name = "John Doe";

  - ✓ There are 4 major access modifiers in C#. These are: Public, Private, Protected, Internal, Protected Internal

Data Types:

- The Data types supported by C#.NET are bool, byte, char, decimal, double, float, int, long, sbyte, short, uint, ulong, ushort. The following table lists the available value types in C#.NET:

Type	Represents	Range
bool	Boolean value	True or False
byte	8-bit unsigned integer	0 to 255
char	16-bit Unicode character	U +0000 to U +ffff
decimal	128-bit precise decimal values with 28-29 significant digits	$(-7.9 \times 10^{28}$ to $7.9 \times 10^{28}) / 10^{0 \text{ to } 28}$
double	64-bit double-precision floating point type	$(+/-)5.0 \times 10^{-324}$ to $(+/-)1.7 \times 10^{308}$
float	32-bit single-precision floating point type	-3.4×10^{38} to $+ 3.4 \times 10^{38}$
int	32-bit signed integer type	-2,147,483,648 to 2,147,483,647
long	64-bit signed integer type	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
sbyte	8-bit signed integer type	-128 to 127
short	16-bit signed integer type	-32,768 to 32,767
uint	32-bit unsigned integer type	0 to 4,294,967,295
ulong	64-bit unsigned integer type	0 to 18,446,744,073,709,551,615
ushort	16-bit unsigned integer type	0 to 65,535

Operators:

- An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations. C# has rich set of built-in operators and provides the following type of operators:

Arithmetic Operators

Following table shows all the arithmetic operators supported by C#. Assume variable A holds 10 and variable B holds 20 then:

Operator	Description	Example
+	Adds two operands	A + B = 30
-	Subtracts second operand from the first	A - B = -10
*	Multiplies both operands	A * B = 200
/	Divides numerator by de-numerator	B / A = 2
%	Modulus Operator and remainder of after an integer division	B % A = 0
++	Increment operator increases integer value by one	A++ = 11
--	Decrement operator decreases integer value by one	A-- = 9

Relational Operators

Following table shows all the relational operators supported by C#. Assume variable A holds 10 and variable B holds 20, then:

Operator	Description	Example

==	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

Logical Operators

Following table shows all the logical operators supported by C#. Assume variable A holds Boolean value true and variable B holds Boolean value false, then:

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non zero then condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands is non zero then condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is true.

Bitwise Operators

Bitwise operator works on bits and perform bit by bit operation. The truth tables for &, |, and ^ are as follows:

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) = 12, which is 0000 1100
|	Binary OR Operator copies a bit if it exists in either operand.	(A | B) = 61, which is 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) = 49, which is 0011 0001
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) = 61, which is 1100 0011 in 2's complement due to a signed binary number.
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 = 240, which is 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 = 15, which is 0000 1111

Assignment Operators

There are following assignment operators supported by C#:

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B assigns value of A + B into C

+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator	C &= 2 is same as C = C & 2
^=	bitwise exclusive OR and assignment operator	C ^= 2 is same as C = C ^ 2
|=	bitwise inclusive OR and assignment operator	C |= 2 is same as C = C | 2

Miscillaneous Operators

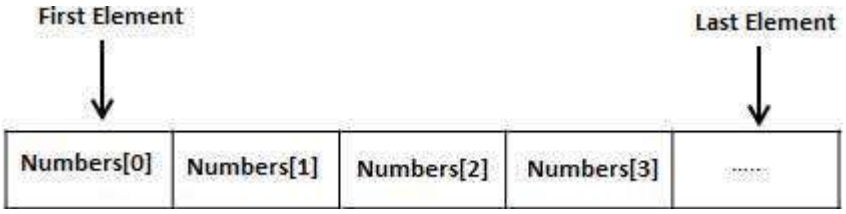
There are few other important operators including sizeof, typeof and ? :supported by C#.

Operator	Description	Example
sizeof()	Returns the size of a data type.	sizeof(int), returns 4.
typeof()	Returns the type of a class.	typeof(StreamReader);
&	Returns the address of an variable.	&a; returns actual address of the variable.
*	Pointer to a variable.	*a; creates pointer named 'a' to a variable.
? :	Conditional Expression	If Condition is true ? Then value X : Otherwise value Y
is	Determines whether an object is of a certain type.	If(Ford is Car) // checks if Ford is an object of the Car class.
as	Cast without raising an exception if the cast fails.	Object obj = new StringReader("Hello"); StringReader r = obj as StringReader;

Arrays:

- An array stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type stored at contiguous memory locations.

- Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables. A specific element in an array is accessed by an index.
- All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.



Declaring Arrays

- To declare an array in C#, you can use the following syntax:

datatype[] arrayName;
where,
datatype is used to specify the type of elements in the array.
[ ] specifies the rank of the array. The rank 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, you can assign values to the array.
- Array is a reference type, so you 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

- You can assign values to individual array elements, by using the index number, like:

```
double[] balance = new double[10];
balance[0] = 4500.0;
```

- You can assign values to the array at the time of declaration, as shown:

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

Enumerations

- An enumeration is a set of named integer constants. An enumerated type is declared using the enum keyword.
- C# enumerations are value data type. In other words, enumeration contains its own values and cannot inherit or cannot pass inheritance.

Declaring enum Variable

- The general syntax for declaring an enumeration is:

```
enum <enum_name>
{
  enumeration list
};
```

Where,

The enum_name specifies the enumeration type name.

The enumeration list is a comma-separated list of identifiers.

- Each of the symbols in the enumeration list stands for an integer value, one greater than the symbol that precedes it. By default, the value of the first enumeration symbol is 0. For example:

```
enum Days { Sun, Mon, tue, Wed, thu, Fri, Sat };
```

Strings

- In C#, you can use strings as array of characters, However, more common practice is to use the string keyword to declare a string variable. The string keyword is an alias for the System.String class.
- You can create string object using one of the following methods:
  - ✓ By assigning a string literal to a String variable
  - ✓ By using a String class constructor
  - ✓ By using the string concatenation operator (+)
  - ✓ By retrieving a property or calling a method that returns a string
  - ✓ By calling a formatting method to convert a value or an object to its string representation

For example

```
string[] sarray = { "Hello", "From", "C#", ".NET" };
string message = String.Join(" ", sarray);
Console.WriteLine("Message: {0}", message);
```

Output :

```
Message: Hello From C#.NET
```

Comments

- Comments are used for explaining code. Compilers ignore the comment entries. The multiline comments in C# programs start with /* and terminates with the characters */ as shown below:

```
/* This program demonstrates
The basic syntax of C# programming
Language */
```

- Single-line comments are indicated by the '//' symbol. For example,

```
}//end class Rectangle
```

Control flow statements

- Sometimes while programming we need to change the flow of the program control to skip the execution of specific statements in a sequence and execute other statements in the application. Change flow statements are used to change the flow of execution of a program.

Selection Statements

- Decision making structures requires the programmer to specify one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.
- An **if statement** can be followed by an optional else statement, which executes when the boolean expression is false.
- The syntax of an if...else statement in C# is:

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

- If the boolean expression evaluates to true, then the if block of code is executed, otherwise else block of code is executed.
- A **switch statement** allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each switch case.
- The syntax for a switch statement in C# is as follows:

```
switch(expression)
{
        case constant-expression  :
                statement(s);
          break; /* optional */
        case constant-expression  :
                statement(s);
          break; /* optional */

        /* you can have any number of case statements */
        default : /* Optional */
                statement(s);
}
```

Iteration Statements of loops

- There may be a situation, when you need to execute a block of code several number of times. In general, the statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on.
- Programming languages provide various control structures that allow for more complicated execution paths.

- A loop statement allows us to execute a statement or a group of statements multiple times and following is the general from of a loop statement in most of the programming languages:
  1. while loop: It repeats a statement or a group of statements while a given condition is true. It tests the condition before executing the loop body.
  2. for loop: It executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
  3. do...while loop: It is similar to a while statement, except that it tests the condition at the end of the loop body
  4. nested loops: You can use one or more loop inside any another while, for or do..while loop.
- For example

```
for (int a = 10; a < 20; a = a + 1)
{
        Console.WriteLine("value of a: {0}", a);
}
Console.ReadLine();
Output :
        value of a: 10
        value of a: 11
        value of a: 12
        value of a: 13
        value of a: 14
        value of a: 15
        value of a: 16
        value of a: 17
        value of a: 18
        value of a: 19
```

Jump Statements

- Branching is performed using jump statements, which cause an immediate transfer of the program control. The following keywords are used in jump statements:
  1. The **break statement** terminates the closest enclosing loop or switch statement in which it appears. Control is passed to the statement that follows the terminated statement, if any.
  2. The **continue statement** passes control to the next iteration of the enclosing while, do, for, or foreach statement in which it appears.
  3. The **return statement** terminates execution of the method in which it appears and returns control to the calling method. It can also return an optional value. If the method is a void type, the return statement can be omitted.
  4. The **throw statement** is used to signal the occurrence of an anomalous situation (exception) during the program execution.
  5. The **goto statement** transfers the program control directly to a labeled statement. A common use of goto is to transfer control to a specific switch-case label or the default label in a switch statement.
  6. The **exit statement** is used to terminates the loop in which it appears. In such a situation, the program control is transferred to the statement that follows the block definition or procedure call.

- **Example:** The conditional statement contains a counter that is supposed to count from 1 to 100; however, the break statement terminates the loop after 4 counts.

```
for (int i = 1; i <= 100; i++)
    {
      if (i == 5)
      {
        break;
      }
      Console.WriteLine(i);
    }
```

Output:

1
2
3
4

Explain the Base Class Library (BCL) in C#

- Almost all the capabilities of the .NET Framework are exposed via a set of managed types known as the Base Class Library (BCL). Because these types are CLS-compliant, they are accessible from almost any .NET language. BCL types are grouped logically by namespace and are exported from a set of assemblies (DLLs) that are part of the .NET platform. Using these types in a C# application requires you to reference the appropriate assembly when compiling.
- In order to work effectively in C# on the .NET platform, it is important to understand the general capabilities in the predefined class library. C#.NET support 4,500 types grouped into 120 namespaces and exported from 40 different assemblies.
- .NET supplies a library of base classes that we can use to implement application quickly. We can use them by simply instantiating them and invoking their methods or by inheriting them through derived classes, thus extending their functionality.
- Much of the functionality in the base framework classes resides in the vast namespace called System. We can use base classes in the system namespace for many different tasks including :
    - Input/output operation
    - Managing arrays,lists, maps etc
    - Accessing the registry
    - Windowing
    - Database management
    - Drawing
    - Connecting to the Internet
    - String handling
    - Accessing files and file system
    - Security
    - Windows messages
    - Evolution of mathematical functions
    - Managing errors and exceptions
    - And many more

What is Exception in C#? Explain Exception handling in C# with Example.

Exception

- An exception is a problem that arises during the execution 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.
    - **throw**: A program throws an exception when a problem shows up. This is done using a throw keyword.
- Assuming a block raises an exception, a method catches an exception using a combination of the try and catch keywords. A try/catch block is placed around the code that might generate an exception. Code within a try/catch block is referred to as protected code, and the syntax for using try/catch looks like the following:

```
try
{
   // statements causing exception
}
catch( ExceptionName e
 )
{
   // error handling code
}
finally
{
   // statements to be executed
}
```

- C# exceptions are represented by classes. The exception classes in C# are mainly directly or indirectly derived from the **System.Exception** class. Some of the exception classes derived from the System.Exception class are the **System.ApplicationException** and **System.SystemException** classes.
- The **System.ApplicationException** class supports exceptions generated by application programs. Hence the exceptions defined by the programmers should derive from this class.
- The **System.SystemException** class is the base class for all predefined system exception.
- The following table provides some of the predefined exception classes derived from the Sytem.SystemException class:

Exception Class	Description
System.IO.IOException	Handles I/O errors.
System.IndexOutOfRangeException	Handles errors generated when a method refers to an array index out of range.
System.ArrayTypeMismatchException	Handles errors generated when type is mismatched with the array type.
System.NullReferenceException	Handles errors generated from dereferencing a null object.
System.DivideByZeroException	Handles errors generated from dividing a dividend with zero.
System.InvalidCastException	Handles errors generated during typecasting.
System.OutOfMemoryException	Handles errors generated from insufficient free memory.
System.StackOverflowException	Handles errors generated from stack overflow.

Handling Exceptions

- Following is an example of throws an exception when dividing by zero condition occurs:

```
using System;
namespace ErrorHandlingApplication
{
    class DivNumbers
    {
        int result;
        DivNumbers()
        {
            result = 0;
        }
        public void division(int num1, int num2)
        {
         try
         {
            result = num1 / num2;
         }

         catch (DivideByZeroException e)
         {
            Console.WriteLine("Exception caught: {0}", e);
         }
       finally
         {
             Console.WriteLine("Result: {0}", result);
         }
       }

       static void Main(string[] args)
         {
            DivNumbers d = new DivNumbers();
            d.division(25, 0);
            Console.ReadKey();
         }
    }
}
```

- **Output:**
  Exception caught: System.DivideByZeroException: Attempted to divide by zero.

Explain Data Types of C# with proper Example

- Data types are the building block of programming language and are used to store data(values), such as numeric, Boolean, and string. In .NET you can use various set of types, known as data types, to store different types of values.
- In .NET Framework, Microsoft tried to standardize the data types by introducing limited and fixed set of data types.
- All the .NET programming language, such as C# and VB.NET uses the same set of data types without any conflict. This means that you can call the C# code from the VB.NET code and VB.NET to C# respectively.
- The following table of different data types in.NET:

Data Types	Storage Size	Value Range
Boolean	2 bytes	True or False
Byte	1 byte	0 through 255 (unsigned)
Char	2 bytes	0 through 65535 (unsigned)
Date	8 bytes	0:00:00 (midnight) on January 1, 0001 through 11:59:59 PM on December 31, 9999
Decimal	16 bytes	0 through +/- 79,228,162,514,264,337,593,543,950,335 (+/- 7.9...E+28) with no decimal point; 0 through +/- 7.9228162514264337593543950335 with 28 places to the right of the decimal
Double	8 bytes	-1.79769313486231570E+308 through -4.94065645841246544E-324, for negative values 4.94065645841246544E-324 through 1.79769313486231570E+308, for positive values
Integer	4 bytes	-2,147,483,648 through 2,147,483,647 (signed)
Long	8 bytes	-9,223,372,036,854,775,808 through 9,223,372,036,854,775,807(signed)
Object	4 bytes on 32-bit platform, 8 bytes on 64-bit platform	Any type can be stored in a variable of type Object
SByte	1 byte	-128 through 127 (signed)
Short	2 bytes	-32,768 through 32,767 (signed)
Single	4 bytes	-3.4028235E+38 through -1.401298E-45 for negative values;1.401298E-45 through 3.4028235E+38 for positive values
String	Depends on implementing platform	0 to approximately 2 billion Unicode characters
UInteger	4 bytes	0 through 4,294,967,295 (unsigned)
ULong	8 bytes	0 through 18,446,744,073,709,551,615 (unsigned)
User-Defined	Depends on implementing platform	Each member of the structure has a range determined by its data type and independent of the ranges of the other members

UShort	2 bytes	0 through 65,535 (unsigned)

String Manipulation

- The .NET Framework provides a String class that contains multiple built-in methods that allows you to manipulate string data.
- There are several string-handling function built into .NET. for example, you use Left, Mid and Right to divide a string into substrings, you find the length of string with Len function.
- Similarly, using these functions you can perform other manipulations in your string data, such as changing the data into upper and lower case, removing the extra space, or extract the characters from the string data.
- The following table of String Manipulating Methods .NET:

Methods	Description
public static int Compare()	Compares two specified string objects and returns an integer that indicates their relative position in the sort order.
public static string Concat()	Concatenates two string objects.
public bool Contains()	Returns a value indicating whether the specified String object occurs within this string.
public static string Copy()	Creates a new String object with the same value as the specified string.
public void CopyTo()	Copies a specified number of characters from a specified position of the String object to a specified position in an array of Unicode characters.
public bool EndsWith()	Determines whether the end of the string object matches the specified string.
public bool Equals()	Determines whether the current String object and the specified String object have the same value.
public static bool Equals()	Determines whether two specified String objects have the same value.
public static string Format()	Replaces one or more format items in a specified string with the string representation of a specified object.
public int IndexOf()	Returns the zero-based index of the first occurrence of the specified Unicode character/string in the current string.
public string Insert(int startIndex, string value)	Returns a new string in which a specified string is inserted at a specified index position in the current string object.

public static bool IsNullOrEmpty(string value)	Indicates whether the specified string is null or an Empty string.
public static string Join()	Concatenates all the elements of a string array, using the specified separator between each element.
public int LastIndexOf()	Returns the zero-based index position of the last occurrence of the specified Unicode character within the current string object.
public string Remove()	Removes all the characters in the current instance, beginning at a specified position and continuing through the last position, and returns the string.
public string Replace()	Replaces all occurrences of a specified character/string in the current string object with the specified character/string and returns the new string.
public string[] Split()	Returns a string array that contains the substrings in the current string object, delimited by elements of a specified Unicode character array.
public bool StartsWith()	Determines whether the beginning of this string instance matches the specified string.
public char[] ToCharArray()	Returns a Unicode character array with all the characters in the current string object.
public string ToLower()	Returns a copy of this string converted to lowercase.
public string ToUpper()	Returns a copy of this string converted to uppercase.
public string Trim()	Removes all leading and trailing white-space characters from the current String object.

- The following program of string comparison .NET:

```
using System;
namespace StringApplication
{
    class StringProg
    {
        static void Main(string[] args)
        {
            string str1 = "This is test";
            string str2 = "This is text";

            if (String.Compare(str1, str2) == 0)
            {
                Console.WriteLine(str1 + " and " + str2 + " are equal.");
            }
            else
            {
                Console.WriteLine(str1 + " and " + str2 + " are not equal.");
            }
            Console.ReadKey();
        }
    }
}
```

Output: This is test and This is text are not equal.

Files and I/O

- A file is one of the mediums to store information in proper format and design. In .NET, all the files should be properly handled to prevent the loss of information stored in them.
- File handling includes reading data from files, writing data to them, copying and moving files from one location to another, renaming and deleting files.
- In .NET, files handling is largely based on the system.IO namespace, which encloses a library that string, character, and file manipulation.
- The most commonly used classes in this namespace are FileStream, BinaryReader, BinaryWriter, StreamReader and StreamWriter. These classes include properties, methods, and events to create, copy, move and delete files.
- The following table of classes offered by System.IO namespace in .NET:

I/O Class	Description
BinaryReader	Reads primitive data from a binary stream.
BinaryWriter	Writes primitive data in binary format.
Directory	Helps in manipulating a directory structure.
DirectoryInfo	Used for performing operations on directories.
DriveInfo	Provides information for the drives.
File	Helps in manipulating files.
FileInfo	Used for performing operations on files.
FileStream	Used to read from and write to any location in a file.
StreamReader	Used for reading characters from a byte stream.
StreamWriter	Is used for writing characters to a stream.
DirectoryNotFoundException	Informs the use that a part of file or directory is missing.
FileFormatException	Raises an exception when the format of input data does not match the specified format of a file.
FileNotFoundException	Raises an exception when you attempt to access the file that does not exist in the hard-disk.

The FileStream Class

- The FileStream class allows you to access files and work with them. You can create, open, and share files with the help of the constructors of the FileStream class.
- This class can open a file either synchronously and asynchronously. By default, the FileStream class opens the files synchronously.
- After opening or creating a file, you can pass an object of the FileStream class to the StreamReader, StreamWriter, BinaryReader and BinaryWriter class to work with the data in the file.

- The syntax for creating a FileStream object is as follows:

  FileStream <object_name> = new FileStream(<file_name>, <FileMode Enumerator>, <FileAccess Enumerator>, <FileShare Enumerator>);

- For example,

  FileStream F = new FileStream("sample.txt", FileMode.Open, FileAccess.Read, FileShare.Read);

- The following table is methods of the FileStream class:

Name	Description
BeginRead()	Begins an asynchronous read operation.
BeginWrite()	Begins an asynchronous write operation.
Close()	Closes the current stream and releases any resources
CopyTo()	Reads the bytes from the current stream and writes them to another stream.
EndRead()	Waits for the pending asynchronous read operation to complete.
EndWrite()	Ends an asynchronous write operation and blocks until the I/O operation is complete.
Seek()	Sets the current position of this stream to the given value.

- The following program demonstrates use of the FileStream class:

```
using System;
using System.IO;

namespace FileIOApplication
{
    class Program
    {
        static void Main(string[] args)
        {
```

```
            FileStream F = new FileStream("test.txt", FileMode.OpenOrCreate,
            FileAccess.ReadWrite);
            for (int i = 1; i <= 20; i++)
            {
                F.WriteByte((byte)i);
            }

            F.Position = 0;
            for (int i = 0; i <= 20; i++)
            {
                Console.Write(F.ReadByte() + " ");
            }
            F.Close();
            Console.ReadKey();
        }
    }
}
```

- **Output:** 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 -1

The StreamWriter and StreamReader Class

You can use the StreamWriter and StreamReader Class to read and write data in files. The StreamReader Class is derived from abstract class called TextReader, which reads characters from stream. StreamWriter Class is derived from abstract class called TextWriter, which writes characters from stream. You can create an object of StreamWriter and StreamReader by passing a file name to their constructors.

- The following table is methods of the StreamReader class:

Name	Description
Close()	Closes the StreamReader object and the underlying stream, and releases any system resources associated with the reader.
Dispose()	Releases all resources used by the TextReader object.
Finalize()	Allows an object to try to free resources and perform other cleanup operations before it is reclaimed by garbage collection.
GetType()	Gets the Type of the current instance.
Read()	Reads the next character from the input stream and advances the character position by one character.
ReadLine()	Reads a line of characters from the current stream and returns the data as a string.
ReadToEnd()	Reads all characters from the current position to the end of the stream.
ToString()	Returns a string that represents the current object.

- The following table is methods of the StreamWriter class:

Name	Description
Close()	Closes the current StreamWriter object and the underlying stream.
Dispose()	Releases all resources used by the TextWriter object.
Finalize()	Allows an object to try to free resources and perform other cleanup operations before it is reclaimed by garbage collection.
Flush()	Clears all buffers for the current writer and causes any buffered data to be written to the underlying stream.
GetType()	Gets the Type of the current instance.
ToString()	Returns a string that represents the current object.
Write(Boolean)	Writes the text representation of a Boolean value to the text string or stream.
WriteLine()	Writes a line terminator to the text string or stream.

The BinaryReader and BinaryWriter Classes

- The BinaryReader and BinaryWriter classes read and write data in binary format, rather than text. You can use the Write() method of the BinaryWriter class to write binary data to a file. To read the binary data, you can use the Read() method of the BinaryReader class.
- The following table is methods of the BinaryReader class:

Name	Description

Close()	Closes the current BinaryReader and the underlying stream.
Dispose()	Releases all resources used by the current instance of the BinaryReader class.
Finalize()	Allows an object to perform cleanup operations.
GetType()	Gets the Type of the current instance.
Read()	Reads the next character from the input stream.
ToString()	Returns a string that represents the current object

- The following table is methods of BinaryWriter class:

Name	Description
Close()	Closes the current BinaryWriter and the underlying stream.
Dispose()	Releases all resources used by the current instance of the BinaryWriter class.
Finalize()	Allows an object to perform cleanup operations.
GetType()	Gets the Type of the current instance.
Flush()	Clears all buffers for the current writer and causes any buffered data to be written to the underlying device.
ToString()	Returns a string that represents the current object

The File Class

- The file class provides static methods for working with files. It also helps in creating the FileStream objects and getting and setting the file attributes. You can set date and time of creating, writing and accessing files by using this class.
- The following table is methods of File class:

Name	Description
AppendAllLines()	Appends lines to a file, and then closes the file. If the specified file does not exist, this method creates a file, writes the specified lines to the file, and then closes the file.
AppendAllText()	Opens a file, appends the specified string to the file, and then closes the file. If the file does not exist, this method creates a file, writes the specified string to the file, then closes the file.
AppendText()	Creates a StreamWriter that appends UTF-8 encoded text to an existing file, or to a new file if the specified file does not exist.

Copy()	Copies an existing file to a new file. Overwriting a file of the same name is not allowed.
Create()	Creates or overwrites a file in the specified path.
CreateText()	Creates or opens a file for writing UTF-8 encoded text.
Decrypt()	Decrypts a file that was encrypted by the current account using the Encrypt method.
Delete()	Deletes the specified file.
Encrypt()	Encrypts a file so that only the account used to encrypt the file can decrypt it.
Exists()	Determines whether the specified file exists.
GetAccessControl()	Gets a FileSecurity object that encapsulates the access control list (ACL) entries for a specified file.
GetAttributes()	Gets the FileAttributes of the file on the path.
GetCreationTime()	Returns the creation date and time of the specified file or directory.
GetCreationTimeUtc()	Returns the creation date and time, in coordinated universal time (UTC), of the specified file or directory.
GetLastAccessTime()	Returns the date and time the specified file or directory was last accessed.
GetLastAccessTimeUtc()	Returns the date and time, in coordinated universal time (UTC), that the specified file or directory was last accessed.
GetLastWriteTime()	Returns the date and time the specified file or directory was last written to.
GetLastWriteTimeUtc()	Returns the date and time, in coordinated universal time (UTC), that the specified file or directory was last written to.
Move()	Moves a specified file to a new location, providing the option to specify a new file name.
Open()	Opens a FileStream on the specified path with read/write access.
OpenRead()	Opens an existing file for reading.
OpenText()	Opens an existing UTF-8 encoded text file for reading.
OpenWrite()	Opens an existing file or creates a new file for writing.
ReadAllBytes()	Opens a binary file, reads the contents of the file into a byte array, and then closes the file.
ReadAllLines()	Opens a text file, reads all lines of the file, and then closes the file.
ReadAllText()	Opens a text file, reads all lines of the file, and then closes the file.
Replace()	Replaces the contents of a specified file with the contents of another file, deleting the original file, and creating a backup of the replaced file.
SetAccessControl()	Applies access control list (ACL) entries described by a FileSecurity object to the specified file.

SetAttributes()	Sets the specified FileAttributes of the file on the specified path.
SetCreationTime()	Sets the date and time the file was created.
SetCreationTimeUtc()	Sets the date and time, in coordinated universal time (UTC), that the file was created.
SetLastAccessTime()	Sets the date and time the specified file was last accessed.
SetLastAccessTimeUtc()	Sets the date and time, in coordinated universal time (UTC), that the specified file was last accessed.
SetLastWriteTime()	Sets the date and time that the specified file was last written to.
SetLastWriteTimeUtc()	Sets the date and time, in coordinated universal time (UTC), that the specified file was last written to.
WriteAllBytes()	Creates a new file, writes the specified byte array to the file, and then closes the file. If the target file already exists, it is overwritten.
WriteAllLines()	Creates a new file, writes a collection of strings to the file, and then closes the file.
WriteAllText()	Creates a new file, writes the specified string to the file, and then closes the file. If the target file already exists, it is overwritten.

The FileInfo Class

- The FileInfo class provides method to create, copy, delete, move and open files. It also helps to create FileStream objects.
- Some methods of the FileInfo class return input/output(I/O) types whenever you create or open any file, which can be used to manipulate the file. It is important to note that you cannot inherit the FileInfo class.
- The following table is methods of FileInfo class:

Name	**Description**
AppendText()	Creates a StreamWriter that appends text to the file represented by this instance of theFileInfo.
CopyTo(String)	Copies an existing file to a new file, disallowing the overwriting of an existing file.
Create()	Creates a file.
CreateText()	Creates a StreamWriter that writes a new text file.
Decrypt()	Decrypts a file that was encrypted by the current account using the Encrypt method.
Delete()	Permanently deletes a file.(Overrides FileSystemInfo.Delete().)
Encrypt()	Encrypts a file so that only the account used to encrypt the file can decrypt it.
GetAccessControl()	Gets a FileSecurity object that encapsulates the access control list (ACL) entries for the file described by the current FileInfo object.
MoveTo(String)	Moves a specified file to a new location, providing the option to specify a new file name.
Open(FileMode)	Opens a file in the specified mode.
OpenRead()	Creates a read-only FileStream.

OpenText()	Creates a StreamReader with UTF8 encoding that reads from an existing text file.
OpenWrite()	Creates a write-only FileStream.
Replace()	Replaces the contents of a specified file with the file described by the current FileInfo object, deleting the original file, and creating a backup of the replaced file.
SetAccessControl( )	Applies access control list (ACL) entries described by a FileSecurity object to the file described by the current FileInfo object.
ToString()	Returns the path as a string.(Overrides Object.ToString().)

Collections

- Collection allows you to store different types of data as well as add, remove or modify the individual elements of these data types.
- It also provides automatic memory management and capacity expansion. In programming, storing data is very important part of a program and we mostly use arrays to store and manage a set of data.
- However, arrays can only store a particular type of data, such as Integer and char.
- To resolve this issue, the .NET Framework introduces the concept of collections for data storage and retrieval; where collection means a group of different types of data.
- The System.Collections namespace provides various classes, such as ArrayList, Stacks, Queues, SortedList, BitArray and Hashtables.
- Some collections classes also provide the search and sort capabilities on the basic of index values or associated key values of the elements.
- Some of the most useful collection classes defined in the System.Collections namespace are as follows:
    1. ArrayList
    2. Hashtable
    3. SortedList
    4. Stack
    5. Queue
    6. BitArray

ArrayList

- It represents ordered collection of an object that can be indexed individually.
- It is basically an alternative to an array. However, unlike array you can add and remove items from a list at a specified position using an index and the array resizes itself automatically. It also allows dynamic memory allocation, adding, searching and sorting items in the list.
- The following table lists some of the commonly used properties of the ArrayList class:

Property	**Description**
Capacity	Gets or sets the number of elements that the ArrayList can contain.
Count	Gets the number of elements actually contained in the ArrayList.
IsFixedSize	Gets a value indicating whether the ArrayList has a fixed size.

IsReadOnly	Gets a value indicating whether the ArrayList is read-only.
Item	Gets or sets the element at the specified index.

- The following table lists some of the commonly used methods of the ArrayList class:

Methods	**Description**
public virtual ArrayList GetRange(int index, int count);	Returns an ArrayList which represents a subset of the elements in the source ArrayList.
public virtual bool Contains(object item);	Determines whether an element is in the ArrayList.
public virtual int Add(object value);	Adds an object to the end of the ArrayList.
public virtual int IndexOf(object);	Returns the zero-based index of the first occurrence of a value in the ArrayList or in a portion of it.
public virtual void AddRange(ICollection c);	Adds the elements of an ICollection to the end of the ArrayList.
public virtual void Clear();	Removes all elements from the ArrayList.
public virtual void Insert(int index, object value);	Inserts an element into the ArrayList at the specified index.
public virtual void InsertRange(int index, ICollection c);	Inserts the elements of a collection into the ArrayList at the specified index.
public virtual void Remove(object obj);	Removes the first occurrence of a specific object from the ArrayList.
public virtual void RemoveAt(int index);	Removes the element at the specified index of the ArrayList.
public virtual void RemoveRange(int index, int count);	Removes a range of elements from the ArrayList.
public virtual void Reverse();	Reverses the order of the elements in the ArrayList.
public virtual void SetRange(int index, ICollection c);	Copies the elements of a collection over a range of elements in the ArrayList.
public virtual void Sort();	Sorts the elements in the ArrayList.
public virtual void TrimToSize();	Sets the capacity to the actual number of elements in the ArrayList.

- The following example demonstrates the concept:

```
using System;
using System.Collections;
namespace CollectionsApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            ArrayList al = new ArrayList();
            Console.WriteLine("Adding some numbers:");
            al.Add(101);
            al.Add(103);
            al.Add(108);
            al.Add(104);
            al.Add(105);
            al.Add(107);
            Console.WriteLine("Capacity: ", al.Capacity);
            Console.WriteLine("Count:", al.Count);
            Console.Write("Content is: ");
            foreach (int i in al)
            {
                Console.Write(i + " ");
            }

            Console.WriteLine();
            Console.Write("Sorted Content: ");
            al.Sort();
            foreach (int i in al)
            {
                Console.Write(i + " ");
            }
            Console.WriteLine();
            Console.ReadKey();
        }
    }
}
```

Output:

Adding some numbers:

Capacity:

Count:

Content is: 101 103 108 104 105 107

Sorted Content: 101 103 104 105 107 108

Hashtable

- It uses a key to access the elements in the collection.
- A hash table is used when you need to access elements by using key, and you can identify a useful key value. Each item in the hash table has a key/value pair. The key is used to access the items in the collection.
- The following table lists some of the commonly used properties of the Hashtable class:

Property	Description
Count	Gets the number of key-and-value pairs contained in the Hashtable.
IsFixedSize	Gets a value indicating whether the Hashtable has a fixed size.
IsReadOnly	Gets a value indicating whether the Hashtable is read-only.
Item	Gets or sets the value associated with the specified key.
Keys	Gets an ICollection containing the keys in the Hashtable.
Values	Gets an ICollection containing the values in the Hashtable.

- The following table lists some of the commonly used methods of the Hashtable class:

Methods	Description
public virtual void Add(object key, object value);	Adds an element with the specified key and value into the Hashtable.
public virtual void Clear();	Removes all elements from the Hashtable.
public virtual bool ContainsKey(object key);	Determines whether the Hashtable contains a specific key.
public virtual bool ContainsValue(object value);	Determines whether the Hashtable contains a specific value.
public virtual void Remove(object key);	Removes the element with the specified key from the Hashtable.

- The following example demonstrates the concept:

```
using System;
using System.Collections;
namespace CollectionsApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            Hashtable ht = new Hashtable();
            ht.Add("001", "AAA");
            ht.Add("002", "BBB");
            ht.Add("003", "CCC");
            ht.Add("004", "DDD");
            ht.Add("005", "EEE");
            ht.Add("006", "FFF");
            if (ht.ContainsValue("GGG"))
            {
                Console.WriteLine("Already in the list");
            }
            else
            {
                ht.Add("007", "GGG");
            }
            // Get a collection of the keys.
            ICollection key = ht.Keys;
            foreach (string k in key)
            {
                Console.WriteLine(k + ": " + ht[k]);
            }
            Console.ReadKey();
        }
    }
}
```

Output:

006: FFF

007: GGG

003: CCC

002: BBB

004: DDD

001: AAA

005: EEE

SortedList

- It uses a key as well as an index to access the items in a list.
- A sorted list is a combination of an array and a hash table. It contains a list of items that can be accessed using a key or an index. If you access items using an index, it is an ArrayList, and if you access items using a key , it is a Hashtable. The collection of items is always sorted by the key value.
- The following table lists some of the commonly used properties of the SortedList class:

Property	Description
Capacity	Gets or sets the capacity of the SortedList.
Count	Gets the number of elements contained in the SortedList.
IsFixedSize	Gets a value indicating whether the SortedList has a fixed size.
IsReadOnly	Gets a value indicating whether the SortedList is read-only.
Item	Gets and sets the value associated with a specific key in the SortedList.
Keys	Gets the keys in the SortedList.
Values	Gets the values in the SortedList.

- The following table lists some of the commonly used methods of the SortedList class:

Methods	Description
public virtual bool ContainsKey(object key);	Determines whether the SortedList contains a specific key.
public virtual bool ContainsValue(object value);	Determines whether the SortedList contains a specific value.
public virtual IList GetKeyList();	Gets the keys in the SortedList.
public virtual IList GetValueList();	Gets the values in the SortedList.
public virtual int IndexOfKey(object key);	Returns the zero-based index of the specified key in the SortedList.
public virtual int IndexOfValue(object value);	Returns the zero-based index of the first occurrence of the specified value in the SortedList.
public virtual object GetByIndex(int index);	Gets the value at the specified index of the SortedList.
public virtual object GetKey(int index);	Gets the key at the specified index of the SortedList.
public virtual void Add(object key, object value);	Adds an element with the specified key and value into the SortedList.
public virtual void Clear();	Removes all elements from the SortedList.
public virtual void Remove(object key);	Removes the element with the specified key from the SortedList.
public virtual void RemoveAt(int index);	Removes the element at the specified index of SortedList.
public virtual void TrimToSize();	Sets the capacity to the actual number of elements in the SortedList.

- The following example demonstrates the concept:

```
using System;
using System.Collections;
namespace CollectionsApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            SortedList sl = new SortedList();
            sl.Add("001", "AAA");
            sl.Add("002", "BBB");
            sl.Add("004", "CCC");
            sl.Add("003", "DDD");
            sl.Add("005", "EEE");
            sl.Add("007", "FFF");
            if (sl.ContainsValue("GGG"))
            {
                Console.WriteLine("already in list");
            }
            else
            {
                sl.Add("006", "GGG");
            }
            ICollection key = sl.Keys;
            foreach (string k in key)
            {
                Console.WriteLine(k + ": " + sl[k]);
            }
            Console.ReadKey();
        }
    }
}
```

Output:

001: AAA

002: BBB

003: DDD

004: CCC

005: EEE

006: GGG

007: FFF

Stack

- It represents a last-in, first out collection of object.
- It is used when you need a last-in, first-out access of items. When you add an item in the list, it is called pushing the item and when you remove it, it is called popping the item.
- The following table lists some commonly used properties of the Stack class:

Property	Description

Count	Gets the number of elements contained in the Stack.

- The following table lists some of the commonly used methods of the Stack class:

Methods	**Description**
public virtual bool Contains(object obj);	Determines whether an element is in the Stack.
public virtual object Peek();	Returns the object at the top of the Stack without removing it.
public virtual object Pop();	Removes and returns the object at the top of the Stack.
public virtual object[] ToArray();	Copies the Stack to a new array.
public virtual void Clear();	Removes all elements from the Stack.
public virtual void Push(object obj);	Inserts an object at the top of the Stack.

- The following example demonstrates the concept:

```
using System;
using System.Collections;
namespace CollectionsApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            Stack st = new Stack();
            st.Push('A');
            st.Push('M');
            st.Push('G');
            st.Push('W');
            Console.WriteLine("Current stack: ");
            foreach (char c in st)
            {
                Console.Write(c + " ");
            }
            Console.WriteLine("Removing values ");
            st.Pop();
            st.Pop();
            st.Pop();

            Console.WriteLine("Current stack: ");
            foreach (char c in st)
            {
                Console.Write(c + " ");
            }
            Console.ReadKey();
        }
    }
}
```

Output:

Current stack:

W G M A Removing values

Current stack:

A

Queue

- It represents a first-in, first out collection of object.
- It is used when you need a first-in, first-out access of items. When you add an item in the list, it is called enqueue and when you remove an item, it is called deque.
- The following table lists some of the commonly used properties of the Queue class:

Property	Description
Count	Gets the number of elements contained in the Queue.

- The following table lists some of the commonly used methods of the Queue class:

Methods	Description
public virtual bool Contains(object obj);	Determines whether an element is in the Queue.
public virtual object Dequeue();	Removes and returns the object at the beginning of the Queue.
public virtual object[] ToArray();	Copies the Queue to a new array.
public virtual void Clear();	Removes all elements from the Queue.
public virtual void Enqueue(object obj);	Adds an object to the end of the Queue.
public virtual void TrimToSize();	Sets the capacity to the actual number of elements in the Queue.

- The following example demonstrates the concept:

```
using System;
using System.Collections;
namespace CollectionsApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            Queue q = new Queue();
            q.Enqueue('A');
            q.Enqueue('M');
            q.Enqueue('G');
            q.Enqueue('W');
            Console.WriteLine("Current queue: ");
            foreach (char c in q)
                Console.Write(c + " ");
            Console.WriteLine("Removing some values ");
            char ch = (char)q.Dequeue();
            Console.WriteLine("Removed ", ch);
            ch = (char)q.Dequeue();
            Console.WriteLine("Removed ", ch);
            Console.WriteLine("Current queue: ");
            foreach (char c in q)
                Console.Write(c + " ");
            Console.ReadKey();
        }
    }
}
```

Output:

Current queue:

A M G W Removing some values

Removed

Removed

Current queue:

G W

BitArray

- It represents an array of the binary representation using the values 1 and 0.
- It is used when you need to store the bits but do not know the number of bits in advance. You can access items from the BitArray collection by using an integer index, which starts from zero.
- The following table lists some of the commonly used properties of the BitArray class:

Property	Description
Count	Gets the number of elements contained in the BitArray.
IsReadOnly	Gets a value indicating whether the BitArray is read-only.
Item	Gets or sets the value of the bit at a specific position in the BitArray.
Length	Gets or sets the number of elements in the BitArray.

- The following table lists some of the commonly used methods of the BitArray class:

Methods	Description
public BitArray And(BitArray value);	Performs the bitwise AND operation on the elements in the current BitArray against the corresponding elements in the specified BitArray.
public BitArray Not();	Inverts all the bit values in the current BitArray, so that elements set to true are changed to false, and elements set to false are changed to true.
public BitArray Or(BitArray value);	Performs the bitwise OR operation on the elements in the current BitArray against the corresponding elements in the specified BitArray.
public BitArray Xor(BitArray value);	Performs the bitwise eXclusive OR operation on the elements in the current BitArray against the corresponding elements in the specified BitArray.
public bool Get(int index);	Gets the value of the bit at a specific position in the BitArray.
public void Set(int index, bool value);	Sets the bit at a specific position in the BitArray to the specified value.
public void SetAll(bool value);	Sets all bits in the BitArray to the specified value.

- The following example demonstrates the concept:

```
using System;
using System.Collections;
namespace CollectionsApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            BitArray ba1 = new BitArray(8);
            byte[] a = { 60 };
            ba1 = new BitArray(a);

            Console.WriteLine("Bit array ba1: 60");
            for (int i = 0; i < ba1.Count; i++)
            {
                Console.WriteLine(ba1[i]);
            }
            Console.ReadKey();
        }
    }
}
```

Output:

Bit array ba1: 60

False

False

True

True

True

True

False

False

Benefits of ADO.NET

- ADO.NET is the data access and manipulation protocol used by C#.NET. It uses a disconnected data architecture. Which means that the data you work with is just a copy of the data in the actual database.
- ADO.NET provides a set of components to create distributed applications. It provides a consistent access to data sources. Such as Microsoft SQL Server, OLE DB and XML.
- You can use ADO.NET to retrieve, manipulate, and update data present in these data source.

1. Interoperability: Uses XML to transfer data across a network. This is not necessary that the destination component should be a .NET application. Any receiving component that is able to read XML can read the data specified in the XML format.
2. Maintainability: Enables you to easily transform the architectural changes in a deployed application. For example, a deployed application is based on the client-server two-tiered architecture. This application is very much popular within users and continuously gaining performance load, which requires some architectural changes. For the increased performance load, the developers of the application decide to divide the server's business=logic processing and user-interface processing on the separate tiers on separate machines. Therefore, the server tier on which the application is running is replaced with two tires.
3. Programmability: Enables you to create programs easily and with fewer mistakes in Visual Studio. ADO.NET allows you to access data through typed programming. Because ADO.NET uses typed DataSets to generate data classes. Typed DataSet provides safe data as .NET performs compile-time checking of types.
4. Performance: Provides fast execution of disconnected applications, over the disconnected RecordSets in classic ADO.
5. Scalability: Provides the conservation of resources, such as database locks and database connections. The applications that are based on ADO.NET support disconnected across to data. So that the databases are not locked by user queries for long duration.

Compare classic ADO and ADO.Net. Also explain advantages of ADO.Net compare to classic ADO.

ADO	**ADO.Net**
It is a COM based Library	It is a CLR based Library
ADO works in the connected mode to access data	ADO.Net works in the disconnected mode to access data
Locking features is available	Locking features is not available
Data is stored in Binary Format	Data is stores in XML
XML integration is not possible	XML integration is possible
It uses RecordSet to store the data from datasource	It uses Dataset to store the data from datasource
Using classic ADO, you can obtain information from one table or set of tables through join. You cannot fetch records from multiple tables independently	Dataset object of ADO.Net includes collection of DataTable wherein each DataTable will contain records

	fetched from a particular table. Hence multiple table records are maintained independently
Firewall might prevent execution of Classic ADO	ADO.Net has firewall proof and its execution will never be interrupted
Classic ADO architecture includes client side cursor and server side cursor	ADO.Net architecture doesn't include such cursors
You cannot send multiple transaction using a single connection instance	You can send multiple transaction using a single connection instance

Explain ADO.Net Architecture with figure. OR

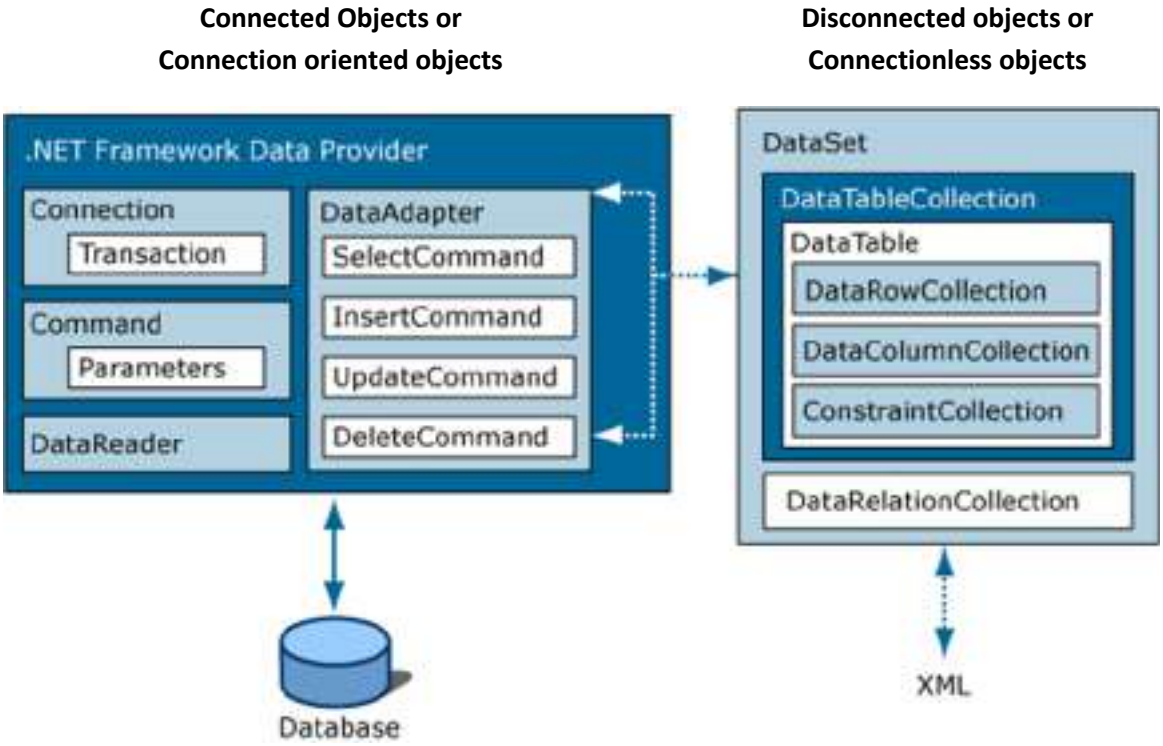
Explain different ADO.Net objects with Example.

What is ADO.NET?

- ADO stands for ActiveX Data Objects
- ADO.NET is a database technology of .NET Framework used to connect application system and database server.
- ADO.NET is a part of the .NET Framework
- ADO.NET consists of a set of classes used to handle data access
- ADO.NET uses XML to store and transfer data among applications, which is not only an industry standard but also provide fast access of data for desktop and distributed applications.
- ADO.NET is scalable and interoperable.

The ADO.NET architecture has two main parts:

1. Data Provider (Connected Objects or Connection oriented objects)
2. Data Set (Disconnected objects or connectionless objects)



Data Provider (Connection Oriented Objects)

- The .NET framework Data Provider is component that has been explicitly designed for data manipulation and fast, forward-only, read-only access to data.
- .NET Framework data provider is used for connecting to a database, executing commands, and retrieving results.
- The Data Provider has four core objects:
    1. Connection
    2. Command
    3. Data Reader
    4. Data Adapter

- **Connection**
    - The Connection object is the first component of ADO.NET.
    - The Connection objects provider connectivity to a data source.
    - It establishes a connection to a specific data source.
    - Connection object helps in accessing and manipulating a database.
    - The base class for all Connection objects is the DbConnection class.
    - Properties of Connection Object
        - **ConnectionString:** Connection String is collection of name/value pairs separated by semicolon which gives information of data source with which connection needs to be established.
        - **Sample connection string:** Data Source=ComputerName\SQLInstance;Initial Catalog=DatabaseName;Integrated Security=False; User ID=username; Password=123;

- **Data Source:** it specifies name of computer and SQL server instance with which connection is required
- **Initial Catalog:** it specifies name of database with which connection is required
- **User ID:** it specifies username to connect with database
- **Password:** it specifies password to connect with database
- **Integrated Security:** it specifies that connection will be established using windows authentication of SQL Server username/password.

- Methods of Connection Object
  - **Open():** this method opens connection using information provided by connection string.
  - **Close():** this method closes already opened connection.

- **Command**
  - The Command object enables access to database commands to return data, modify data, run stored procedures, and send or retrieve parameter information.
  - It executes a command against a data source. Exposes Parameters and can execute in the scope of a Transaction from a Connection.
  - You can execute SQL queries to return data in a DataSet or a DataReader object.
  - Command object performs the standard Select, Insert, Delete and Update T-SQL operations.
  - The base class for all Command objects is the DbCommand class.
  - Properties of Command Object
    - **Connection :** It specifies that on which connection command executes
    - **CommandType:** It specifies type of Command to execute
      - Text – SQL Statement as command type
      - StoredProcedure – Stored Procedure as command type
      - TableDirect – Table Name as command type
    - **CommandText:** Either SQL Statement or name of Stored procedure or name of Database table
  - Methods of Command Object
    - **ExecuteNonQuery()**
      This method executes the command specifies and returns the number of rows affected
    - **ExecuteScalar()**
      This method executes the command specified and returns the first column of first row of the result set. The remaining rows and column are ignored.
    - **ExecuteReader()**
      The ExecuteReader method executes the command specified and returns an instance of SqlDataReader class.
    - **ExecuteXmlReader()**
      This method executes the command specified and returns an instance of XmlReader class. This method can be used to return the result set in the form of an XML document
    - **CreateCommand():** this method created new command object on given connection object.

- **Data Reader**
  - The Data Reader provides a high-performance stream of data from the data source.
  - It reads a forward-only, read-only stream of data from a data source.
  - DataReader object works in connected model.
  - The base class for all DataReader objects is the DbDataReader class.
  - Methods of Data Reader object
    - **Read():** this method reads next row from DataReader object, if row exists it returns true otherwise it returns false.

- **Data Adapter**
  - The Data Adapter provides the bridge between the Data Set object and the data source.
  - The Data Adapter uses command object to execute SQL commands at the data source to both load the Data Set with data and reconcile changes that were made to the data in the dataset back to the data source.
  - It populates a DataSet and resolves updates with the data source.
  - The base class for all DataAdapter objects is the DbDataAdapter class.
  - Methods of Data Adapter object
    - **Fill():** this method takes the results of a database query from a Command object and pushes them into a DataSet
    - **Update():** this method will negotiate any changes to a DataSet back to the original data source.
- The following lists the data providers that are included in the .NET framework.

Data Provider	Description
SQL Server	o Provides data access for Microsoft SQL server. o Uses the System.Data.SqlClient namespace.
OLEDB	o For data sources exposed by using OLEDB. o Uses the System.Data.OleDb namespace.
ODBC	o For data sources exposed by using ODBC. o Uses the System.Data.Odbc namespace.
Oracle	o For Oracle data sources. o Uses the System.Data.OracleClient namespace.

Following Code demonstration of binding GridView control using Connection Oriented Objects of ADO.Net

```
//Step 1: Prepare Connection
SqlConnection objConnection = new SqlConnection();
objConnection.ConnectionString = @"Data Source=ComputerName\SQLInstance;Initial
Catalog=DatabaseName;Integrated Security=False; User ID=username; Password=123;";
objConnection.Open();

//Step 2: Prepare & Execute Command
SqlCommand objCommand = new SqlCommand();
objCommand.Connection = objConnection;
objCommand.CommandType = CommandType.Text;
objCommand.CommandText = "SELECT CountryID, CountryName FROM Country ORDER BY CountryName";

//Step 3: Collect Data to DataReader object which has been received as a result of Command
SqlDataReader objSDR = objCommand.ExecuteReader();

gvCountry.DataSource = objSDR;
gvCountry.DataBind();

objConnection.Close();
```

Data Set (Disconnected objects)

- The dataset object is central to supporting disconnected, distributed data scenarios with ADO.NET.
- The dataset is a memory-resident representation of data that provides consistent relational programming model regardless of the data source.
- The dataset represents a complete set of data, including related tables, constraints, and relationship among the table.
- The dataset has two major objects:
  1. **The Data Table Collection:**
     - The Data table collection contains all the data table objects in a dataset.
     - A Data table is defined in the System.Data namespace and represents a single table of memory-resident data.
     - It contains a collection of columns represented by a data column collection, and constraints represented by a constraint collection, which together define the schema of the table.
  2. **The Data Relation Collection:**
     - A relationship represented by the Data relation object, associated rows in one Data table with rows in another Data table.
     - A relationship is analogous to a join path that might exist between primary and foreign key columns in a relational database.
     - A data relation identifies matching columns in two tables of a dataset.
     - The essential element of a data relation are:
       - The name of the relationship
       - The name of the tables being related
       - The related column in each table
     - Relationship can be built with more than one column per table by specifying an array of Data Column objects as the key columns.

- When you add a relationship to the data relation collection, you can optionally add a Unique key constraint to enforce integrity constraints when changes are made to related column values.

Following code demonstrate code of disconnected objects

```
using System;
using System.Data;
using System.Data.SqlClient;
namespace DataTable_Datarow_DataColumn_Example
{
    class Program
       {
        static void Main(string[] args)
        {
            //Step 1: Prepare Connection
              SqlConnection objConnection = new SqlConnection();
              objConnection.ConnectionString = @"Data Source=ComputerName\SQLInstance;Initial
              Catalog=DatabaseName;Integrated Security=False; User ID=username;
              Password=123;";
              objConnection.Open();

              //Step 2: Prepare & Execute Command
              SqlCommand objCommand = new SqlCommand();
              objCommand.Connection = objConnection;
              objCommand.CommandType = CommandType.Text;
              objCommand.CommandText = "SELECT CountryID, CountryName FROM Country ORDER BY
                                        CountryName";

              SqlDataAdapter sda = new SqlDataAdapter(objCommand);
              DataTable dt = new DataTable();
              sda.Fill(dt);

              objConnection.Close();

        }
    }
}
```

Explain Typed Dataset.

- Along with late bound access to values through weakly typed variables, the DataSet provides access to data through a strongly typed metaphor.
- Tables and columns that are part of the DataSet can be accessed using user-friendly names and strongly typed variables.
- A typed DataSet is a class that derives from a DataSet. As such, it inherits all the methods, events, and properties of a DataSet. Additionally, a typed DataSet provides strongly typed methods, events, and properties.
- This means you can access tables and columns by name, instead of using collection-based methods.

- Aside from the improved readability of the code, a typed DataSet also allows the Visual Studio .NET code editor to automatically complete lines as you type.
- Additionally, the strongly typed DataSet provides access to values as the correct type at compile time.
- With a strongly typed DataSet, type mismatch errors are caught when the code is compiled rather than at run time.

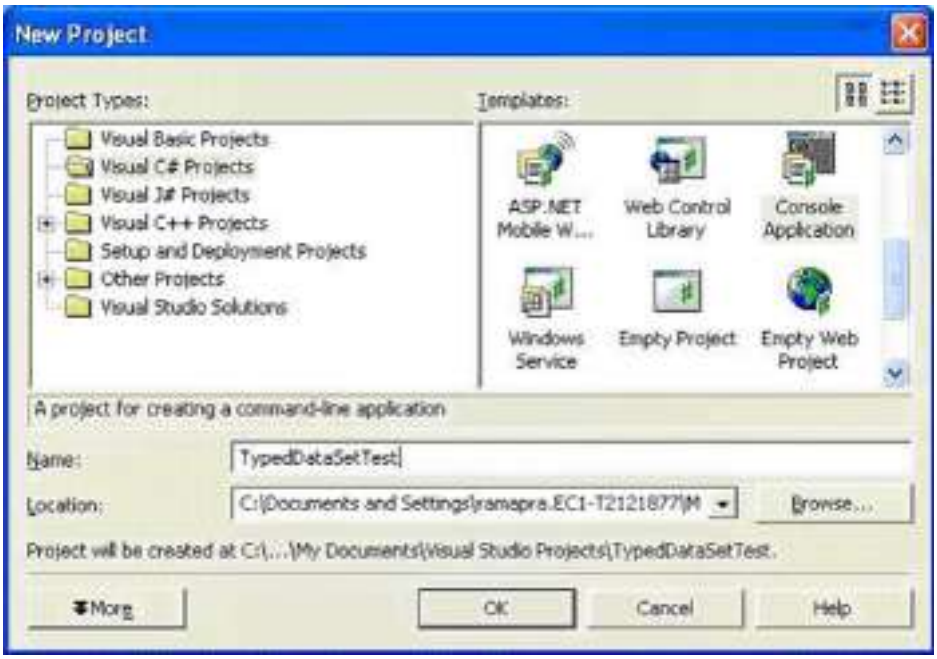
How to Generate Typed DataSet?

- A Typed DataSet can be generated in two ways,
  1. Using Visual Studio .NET IDE.
  2. Using XSD.exe (Using VS.Net command prompt). Open VS.Net command prompt and Type XSD /? For the help on this exe.

Creating a Typed DataSet using Visual Studio .NET IDE

Step by step procedure to create a Typed DataSet,

1. Open VS .Net IDE and Click on File -> New -> Project and Select Console Application.
2. Enter name for the project. Say TypedDataSetTest.



3. Right click on the solution and click on Add-> Add New Item will show a dialog box

Select DataSet from templates pane, give the name (Say TypedDs.xsd) and click on Open. This will add file by name TypedDs.xsd to the solution.

4. Click on the Server Explorer browse to the database and drop the table on the TypedDs.xsd file. if we check the xml file for the same then we can see the schema for the table.

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema id="AnalystListDataSet" xmlns="urn:AnalystList" xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:msdata="urn:schemas-microsoft-com:xml-msdata" targetNamespace="urn:AnalystList" attributeFormDefault=
    elementFormDefault="qualified">
    <xs:element name="AnalystListDataSet" msdata:IsDataSet="true">
        <xs:complexType>
            <xs:choice maxOccurs="unbounded">
                <xs:element name="Table">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="empid" type="xs:int" minOccurs="0" />
                            <xs:element name="empname" type="xs:string" minOccurs="0" />
                            <xs:element name="empaddress" type="xs:string" minOccurs="0" />
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
            </xs:choice>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

The Windows Forms Model

- In .NET, the Windows Forms model provides forms, controls, and their events, to create Windows applications.
- A Windows Form is a graphical user interface on which you display information, either textual or graphical to a user.
- You can also allow the user to enter input in the form of text or images by using variety of controls. Windows applications are developed by adding controls to forms and by handling events, such as mouse clicks, button clicks and so on.
- You can create Windows Forms by using the Form class. This class belongs to the System.Windows.Forms namespace.
- The Form class itself is based on the Control class, which means that the forms share many properties and methods of the Control class. The inheritance hierarchy of the Form class is:

System.Object
System.MarshalByRefObject
System.ComponentModel.Component
System.Windows.Forms.Control
System.Windows.Forms.ScrollableControl
System.Windows.Forms.ContainerControl
System.Windows.Forms.Form

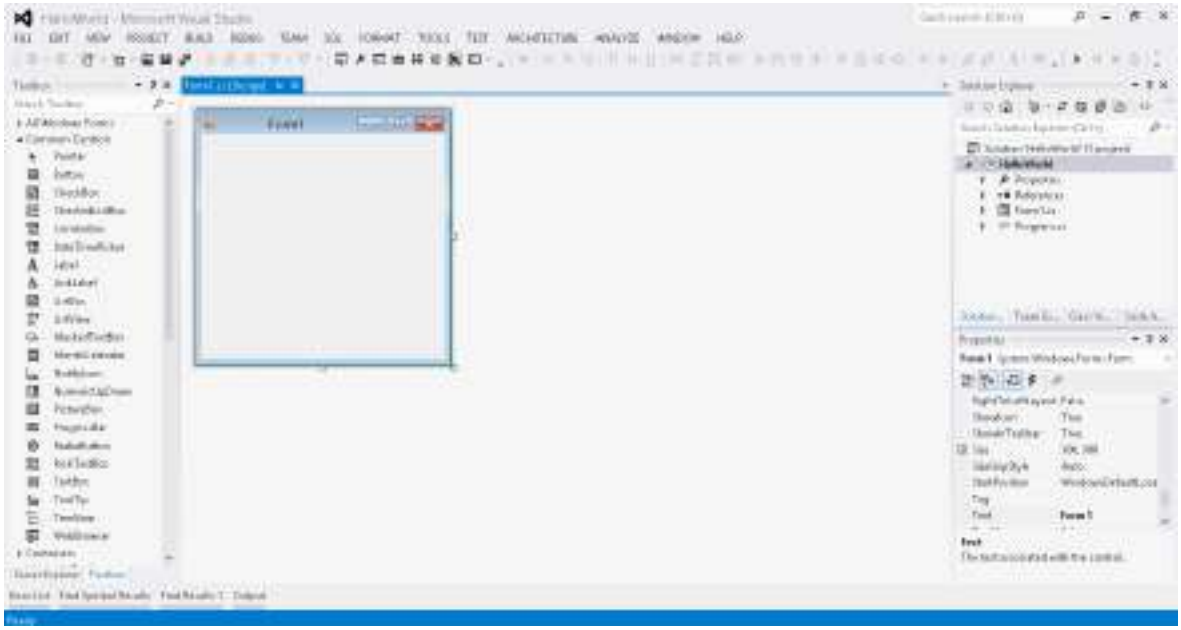


Figure 1: Displaying a Form Designer in Visual Studio

- In Figure 1, the title of the form, Form1, it displays on the Form title bar. At the top-right corner of the form is the control box containing the Minimize, Maximize and Close buttons. Where you place the controls such as buttons, text boxes and labels is called the main area of a Form.

- Each form in this namespace is an instance (that is, an object) of the Form class. Objects are instances of classes, much as an integer variable is an instance of the integer type.
- There are two kinds of class members. The first are members inherent to the class itself (accessed through the class), such as Form.ActiveForm or just ActiveForm.
- These members called static or class members, do not need objects to access the members of the class. The second kind of class members are called instance or object members built into Objects, such as MyForm.BackColor.
- The MyForm1 is an instance of the Form class, where you do need an object to access the class members.
- The basic difference between a class member and an object member is that to use a class member, you do not need an object of that class, but with object members, you need the following members:
  1. Static members: Accesses directly using the class, such as classname.membername
  2. Instance members: Accesses by using an instance of a class (an object), such as objectname.membername
- We prefer the terms such as class members and object members because these clarify what type of members they are, but Microsoft Developer Network (MSDN) often uses the terms Static members and instance members.

Creating Windows Forms

- Windows Forms are one of the important forms that are available in C#.NET. You can create variety of application using these forms. You can add various controls on it to make the application more functional and user-friendly. Let's create a simple Windows Forms application, named FirstProject, by performing the following steps:

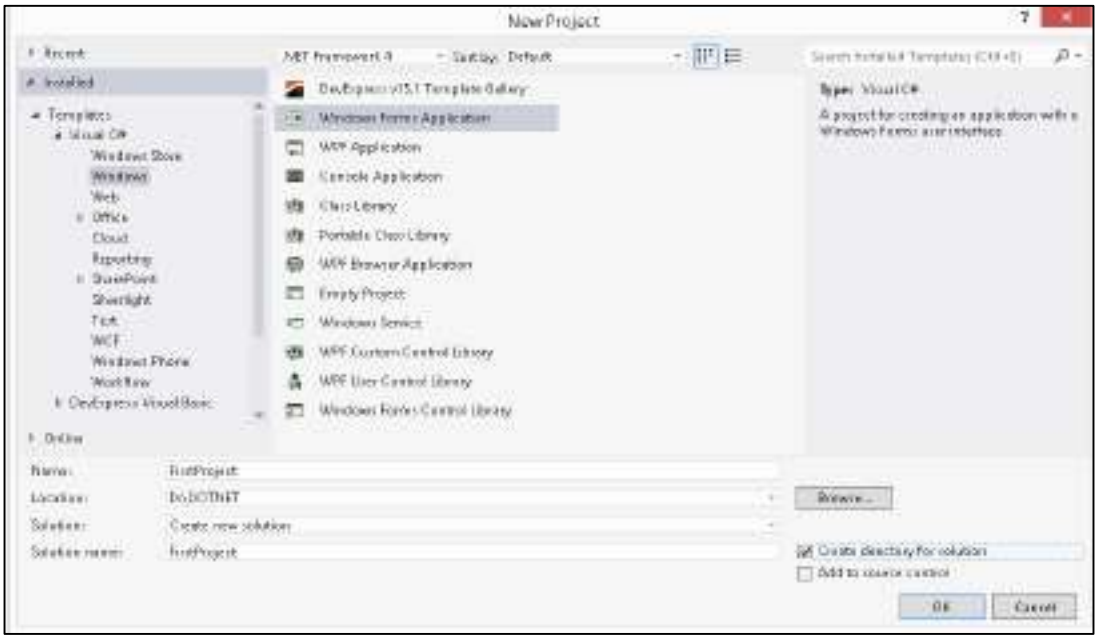


Figure 2: Displaying the New Project Dialog Box

1. Click File->New->Project. The New Project dialog box appears (Figure 2).

2. Select Visual C#-> Windows option from the Installed Templates pane and then select Windows Forms Application option from the middle pane.
3. Enter a name for your application in Name text box. In our case, we have entered the FirstProject.
4. Enter a location for the application, for example D:\DOTNET, in the Location drop down box by using the Browse button.
5. Click the OK button, as shown in Figure 2:

- The FirstProject application is created, as shown in Figure 3:

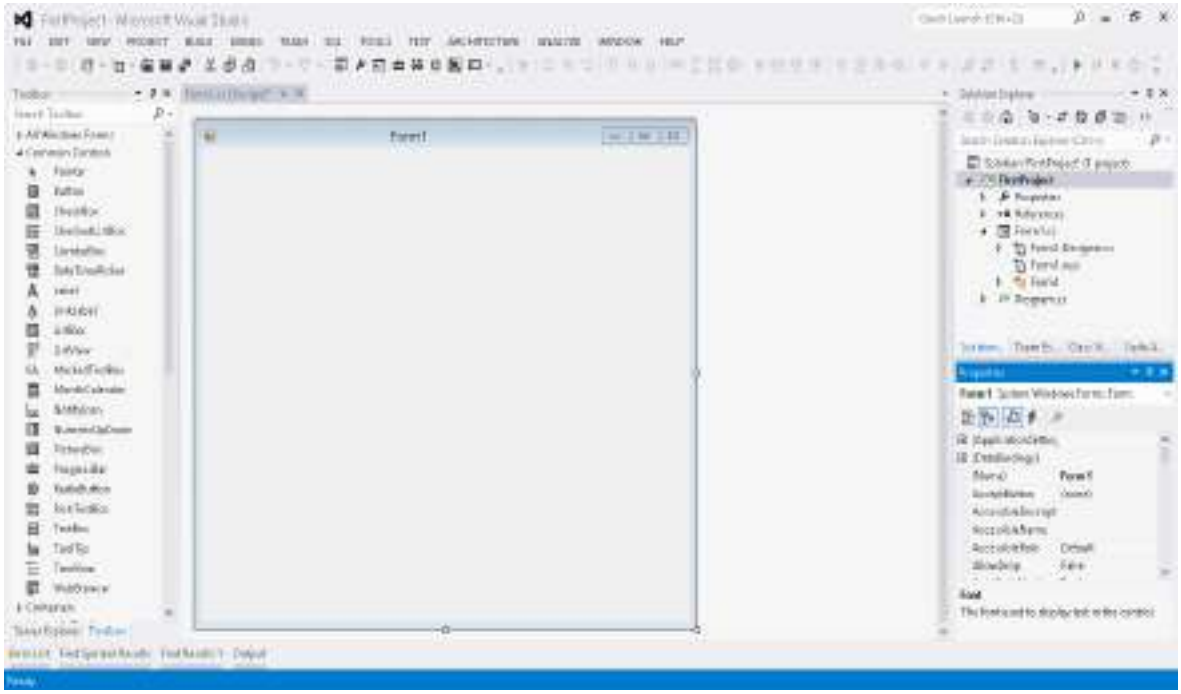


Figure 3: Designing a New Windows Application

- Whenever you create a new Windows Form application or project, certain files and directories are created as well. In our case, the following files and directories are created with the FirstProject application:
  1. My Project.csproj: Opens the Project Designer to access the project properties, settings, and resources
  2. Assemblyinfo.cs: Contains general information such as the assembly and version of the application
  3. Form1.cs: Specifies a file containing the code of a form
  4. Resources.resx: Refers an XML-based resource template
  5. FirstProject.sin: Refers the solution file of the application, storing the solution's configuration
  6. bin: Specifies the directory for binary executable
  7. Obj: Specifies the directory for debugging binaries
- C#.NET creates all these files and directories automatically.
- Let's learn about Windows Forms in the following sections:
  A. Adding controls to Windows Forms
  B. Disabling and enabling Windows Forms
  C. Changing the title of Windows Forms
  D. Setting a border of Windows Forms
  E. Displaying and hiding the Maximize, Minimize, and Close buttons of Windows Forms

F. Specifying the initial position of Windows Forms
G. Creating multiform Windows Application
H. Setting the Startup form

Adding controls to Windows Forms

- You can add many useful controls in a Windows application, such as ScrollBars, Buttons, TextBoxes and Menus, which users generally use to interact with the application. In C#.NET, you use the Toolbox to add controls to a form.
- Let's now add a Button control and a TextBox control by just dragging the controls from the Toolbox to Form1 of the FirstProject application, as shown in Figure 4:

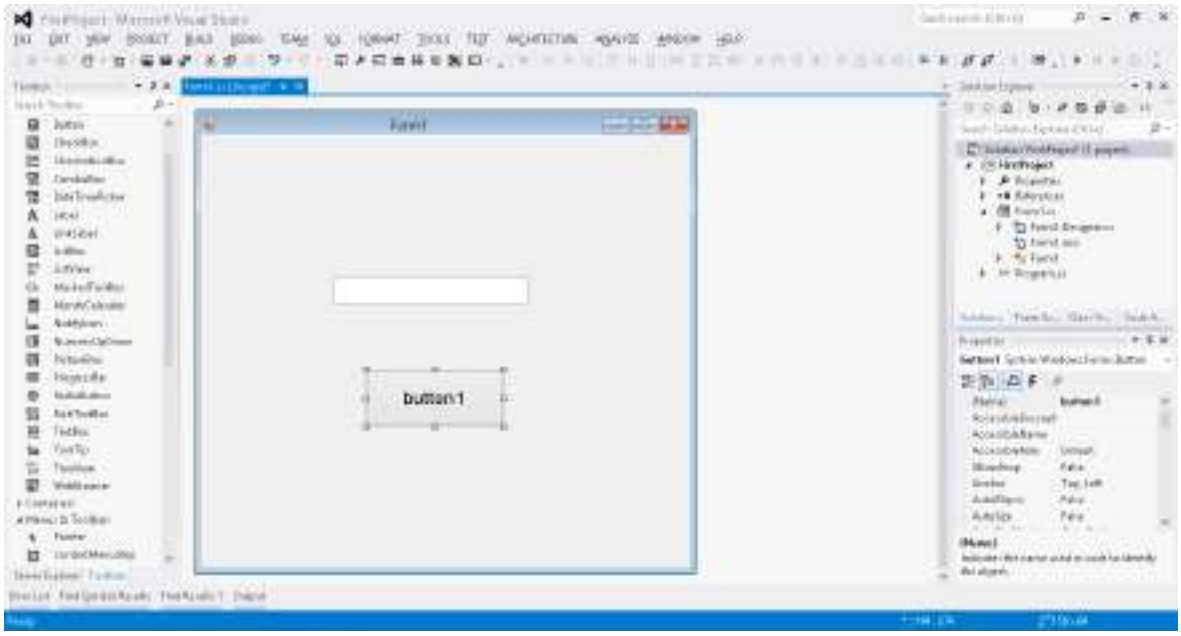


Figure 4: Adding Controls to a Form

- You can also add controls to Form1 by just double-clicking the required control in the Toolbox. The names for these controls, such as Button1 and Textbox1 are provided by C#.NET automatically.
- In this application, we want to display text in the TextBox when a user clicks the Button1 button. Therefore, change the Text property of Button1 to Click Me using the Properties window, as shown in Figure 5:

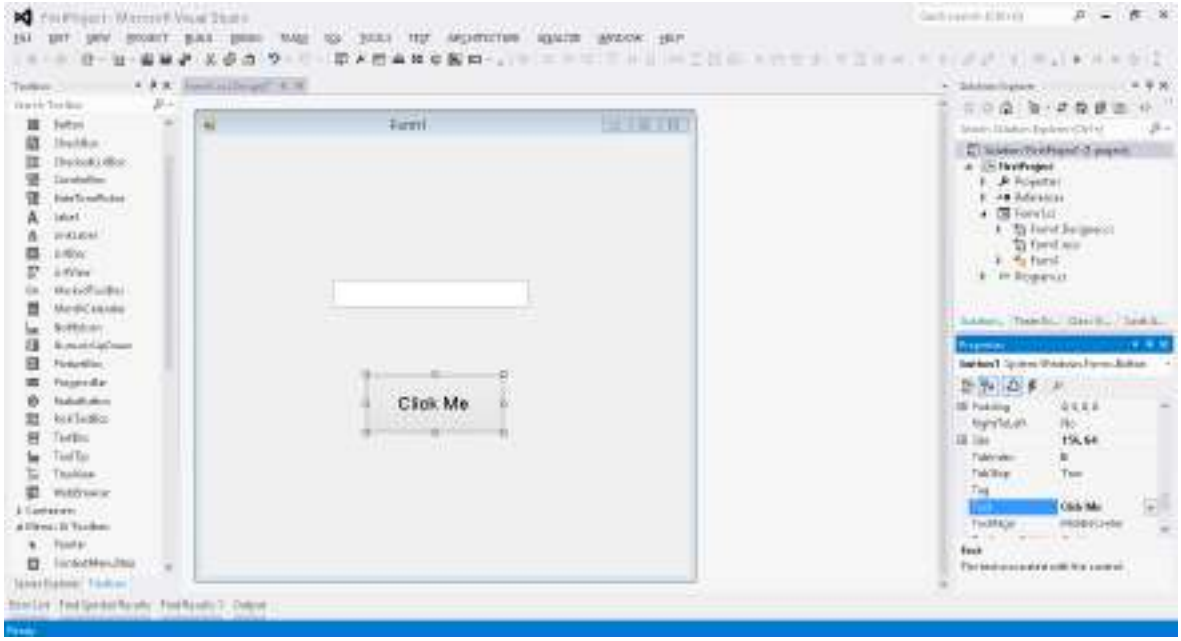


Figure 5: Changing the Caption of Button1 Control

Disabling and enabling Windows Forms

- You can set the various properties of a Windows Form to perform manipulations in your applications.
- For example, you can set the properties to enable or disable the Windows Form. You can do this by using the Properties window of the form or by writing the code for the same. Figure 6 shows how you can use the Properties window to enable and disable a Windows Form:

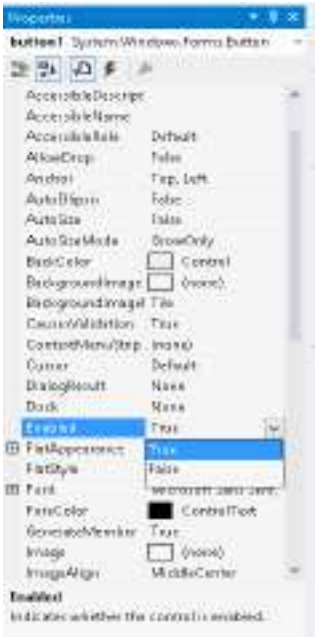


Figure 6: Displaying Properties Window for Windows Forms

- To disable a Windows Form, simply set the Enabled property to False in the Properties window of the form. Similarly, you can enable a disabled Windows Form by setting the Enabled property to True. By default, the Enabled properly of a Windows Form is True.
- Let's create a Windows Forms application, named DisableEnableWindowsForm. Add a Button control to Form1 and change its Text property to the Click Me to Disable this Form, as shown in Figure 7:

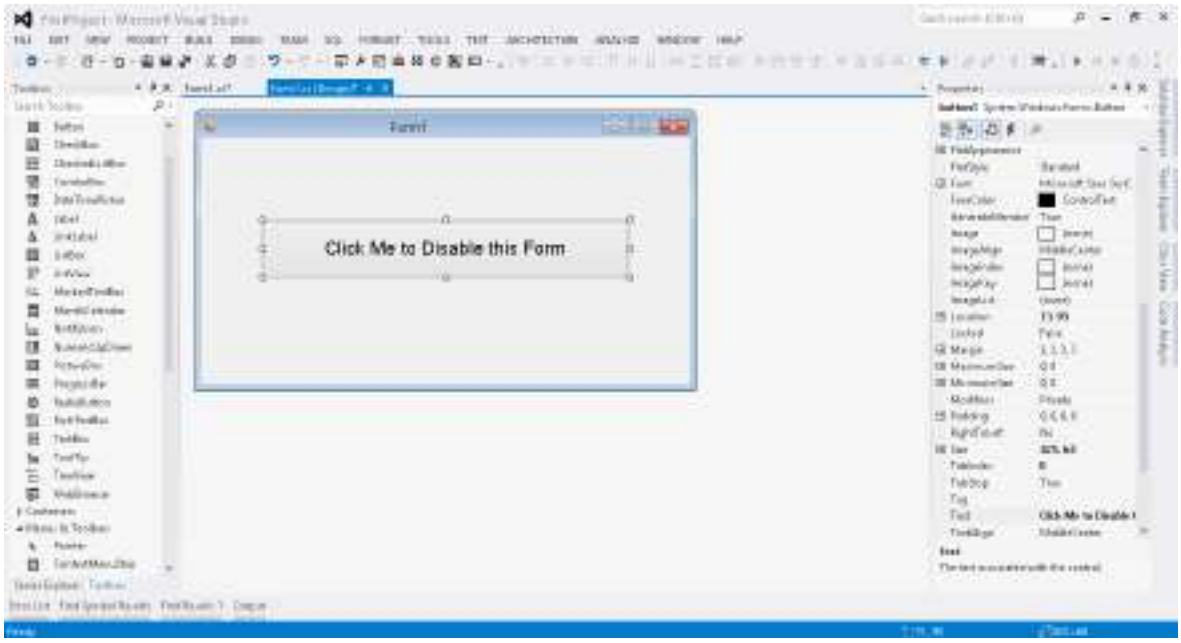


Figure 7: Displaying the Design View of the DisableEnableWindowsForm Application

1. Add the following code to the Click event of the Button1 control to disable Form1 as follows:

```
private void button1_Click(object sender, EventArgs e)
{
    this.Enabled = false;
}
```

2. Run the application by pressing the F5 key and click the Click Me to Disable this Form button.

- When you click the Click me to Disable this Form button, the form gets disabled, as shown in the Figure 8:

Figure 8: Displaying a Disabled Windows Form

Changing the title of Windows Forms

- The title of a Windows Form is an important aspect of the application of which it is a part. It depicts the nature of the work performed by the application. You can easily change the title of a Windows Form either at design time or at runtime.
- If you want to change the title of the form during the designing stage, you can do this by just changing the property of Form1 to Welcome, as shown in Figure 9:

Figure 9: Displaying Properties Window for Windows Form1

- You also can set the Text property at runtime by writing the following code :

```
private void Form1_Load(object sender, EventArgs e)
{
    this.Text = "Title Change";
}
```

- Now press the F5 key to view the output. Figure 10 shows the form with a changed title:

Figure 10: Showing the Title of a Windows Forms

- In Figure 10, the title is changed from Welcome to Title Change. You can find the code to change the title of a Windows Form in the ChangeTitle application

Setting a border of Windows Forms

- You can set a form's border style with its FormBorderStyle property by using the Properties window of Form1 or by writing the code in the code-behind file. The possible values for the FormBorderStyle property are given in Table 1:

Style	Description
Flxed3D	Specifies a fixed, three-dimensional border
FixedDialog	Specifies a thick, fixed dialog-styie border
FixedSingle	Specifies a fixed, single-line border
FixedToolWindow	Specifies a tool window border that is not resizable
None	Specifies no border
Sizable	Specifies a resizable border
SizabieToolWindow	Specifies a resizable tool window border

Table 1: Form Border Styles

- Let's create a Windows Forms application named FormBorder to display a form with a border. Now, set the Text and FormBorderStyle property of Form1 by adding the following code to the Load event of Form1:

```
private void Form1_Load(object sender, EventArgs e)
{
    this.Text = "Form Border";
    this.FormBorderStyle = System.Windows.Forms.FormBorderStyle.Fixed3D;
}
```

- This code helps to Set the Form title to Form Border and change the border style of Form1 to the Fixed3D style.
- Press the F5 key to view the output of Form1 as shown in Figure 11:

Figure 11: Changing a Form's Border Style

- You can also set the Text property of Form1 to Form Border and FormBorderStyle property to Fixed3D at the design time of Form1 using the Properties window.

Displaying and hiding the Maximize, Minimize, and Close buttons of Windows Forms

- Windows Forms usually come with the Minimize and Maximize buttons, as well as a Close button at the upper right corner of the form. To remove these buttons, set the form's ControlBox property to False by using the Properties window of the concerned form, as shown in Figure 12:

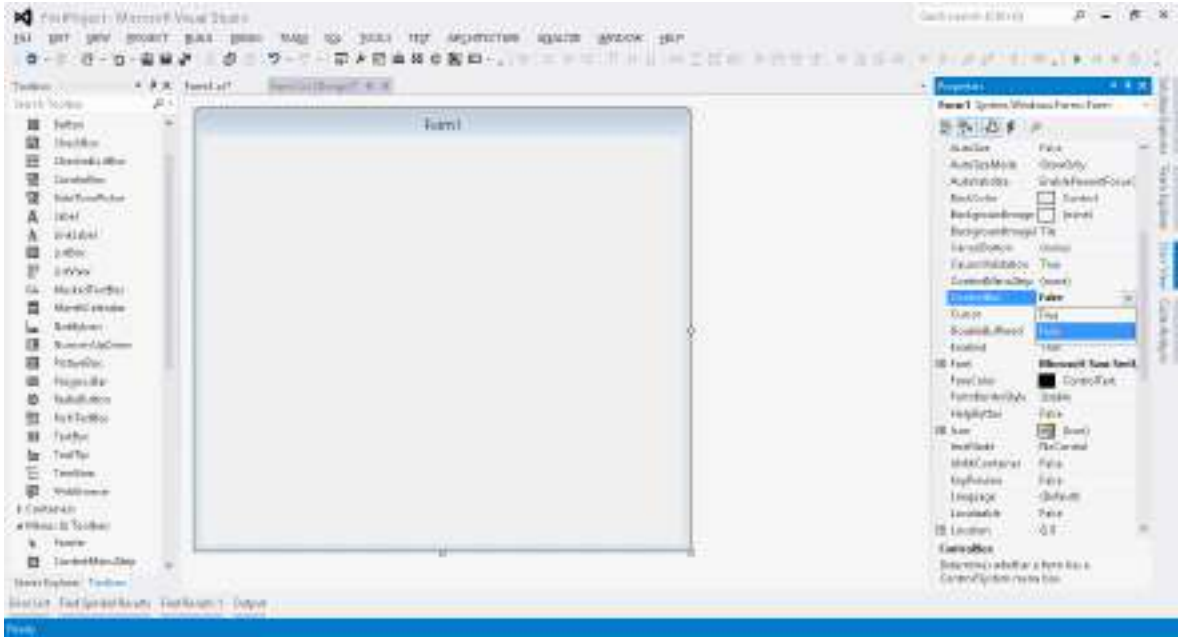


Figure 12: Setting the ControlBox Property form the Properties Window

- You can also display or hide the Minimize and Maximize buttons independently, by setting their values to True or False using the MaximizeBox and MinimizeBox properties in the Properties window.

Specifying the Initial position of Windows Forms

- You can use a form's StartPosition property to specify its initial position on the screen.
- Assign the value for this property from the StartPosition enumeration. The possible values for the position of windows Forms are given in Table 2:

Value	Description
CenterParent	Centers the form within the bounds of its parent form
CenterScreen	Centers the form with the specified dimensions of the form using its Size property
Manual	Determines the starting position of the form using its Location and Size properties
WindowsDefaultBounds	Positions the form at the default location of the Windows interface and has the bounds determined by Windows default
WindowsDefaultLocation	Positions the form at the default location of the Windows interface and has the dimensions specified in the form's Size property

Table 2: Positions of Windows Forms

- Let's create a Windows Form application named InitialPosition
- You can set all these positions by using the Properties window of Form1, as shown in Figure 13:

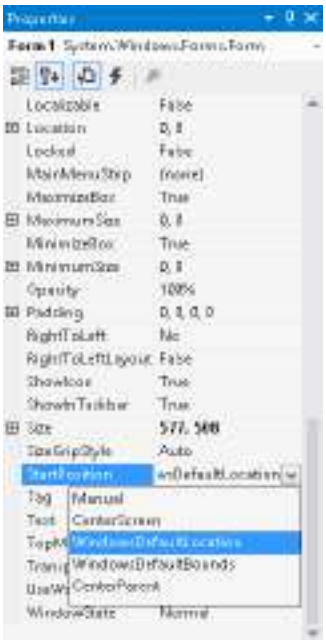


Figure 13: Displaying the Properties of a Windows Form

- You can also set the initial position of a Windows Form by writing the following code in the Form1_Load event:

```
private void Form1_Load(object sender, EventArgs e)
{
    this.StartPosition = FormStartPosition.CenterParent;
}
```

- In the preceding code, we have set the position of the form as CenterParent. Let's now learn how to create multiform Windows applications in C#.NET.

Creating multiform Windows Application

- Suppose that you have designed your application with an introductory form to welcome the user, a data entry form to get data from the user, a summary form to display the data analysis results. You have to now use or send the data of one form to another, for instance the data entered in a data entry form is required in data analysis summary form. In such cases, you can create Windows Forms applications that contain multiple forms.
- To understand multiform applications and how to communicate between them, let's create a new Windows application called MultiForm and change the Text property of Form1 to FirstForm. Perform the following steps:
    1. Select Project->Add Windows Form to insert an additional form, as shown in Figure 14:

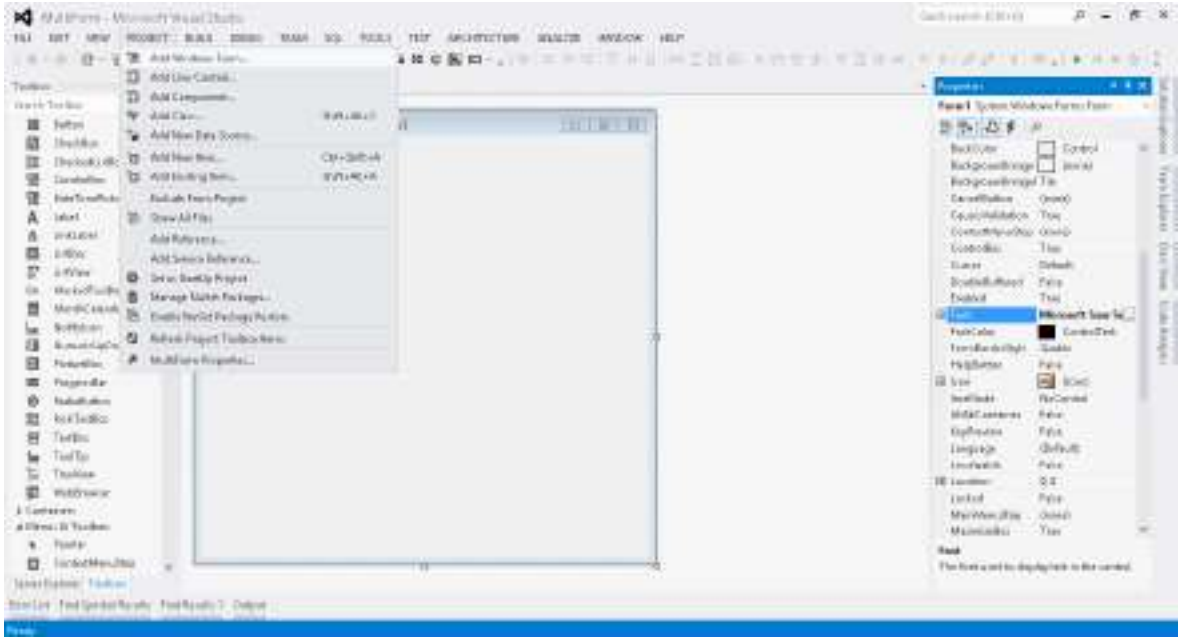


Figure 14: Selecting the Add Windows Form Option from Project Menu

The Add New Item dialog box appears, as shown in Figure 15:

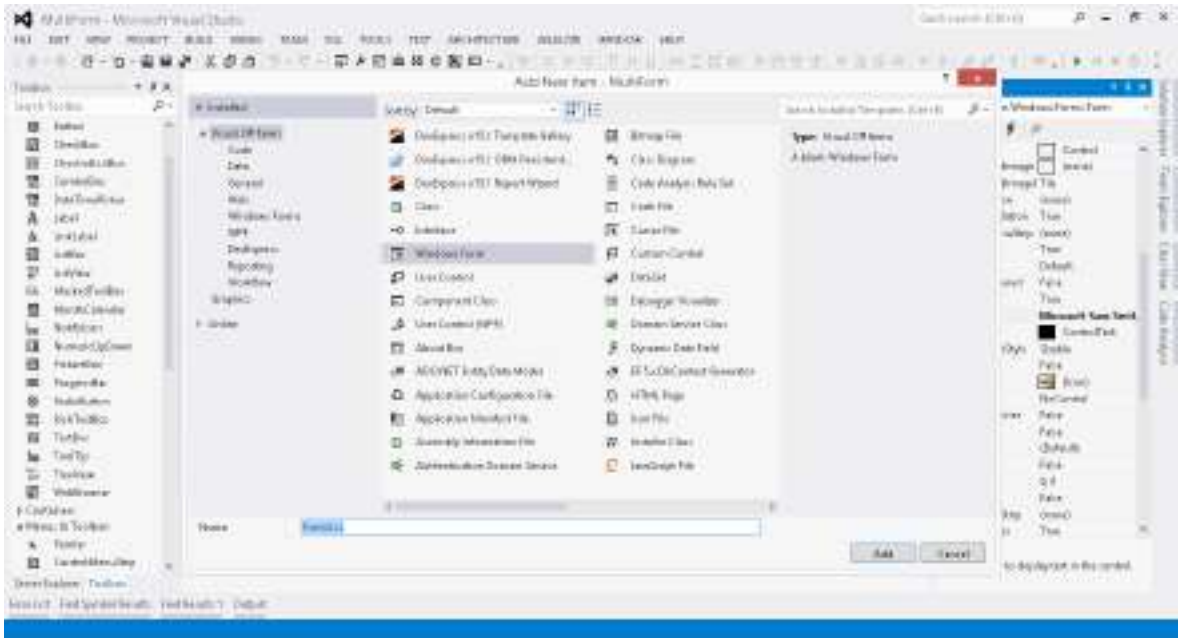


Figure 15: Add New Item Dialog Box

2. Select the Windows Form template in the middle pane and click the Add button. A new form. Form2. Is added to the application, as shown In Figure 16:

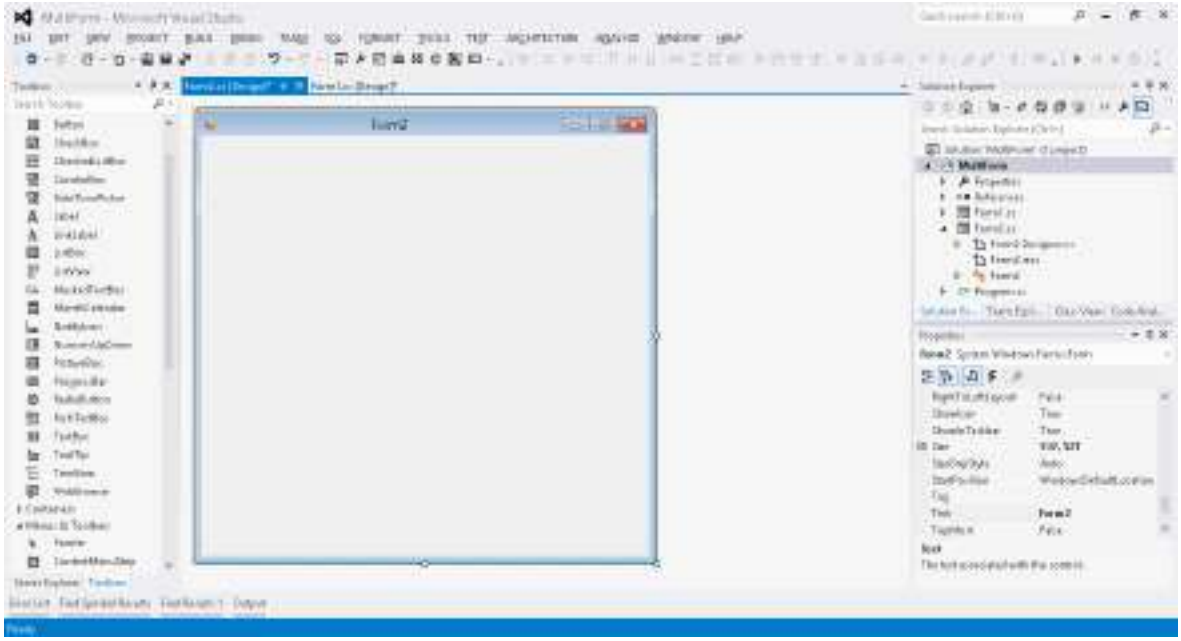


Figure 16: Displaying a New Windows Form Added to the Project

3. Change the Text properly of Form2 to Second Form.
4. Add a TextBox and a Button control to Form1.
5. Set the Text property of button1 control to Read Text from First Form.
6. Add the following code to the code-behind file to handle the Click event of the Button control of Form1:

```
private void button1_Click(object sender, EventArgs e)
{
    Form2 secForm = new Form2();
    secForm.Message = textBox1.Text;
    secForm.Show();
}
```

Inside the button 1_Click event handler in the preceding code, first we are creating an object of Form2, secondForm. After that, we are setting the Message property that we have created on Form2, with the text entered by the user in the TextBox present on Form2 at runtime. At last, we are showing Form2 by calling its Show() method.

7. Add a TextBox control to Form2.
8. Add the Code in the Form2.cs file:

```
namespace MultiForm
{
    public partial class Form2 : Form
    {
        private string strMessage;
        public string Message
        {
            get
            {
                return strMessage;
            }
            set
            {
                strMessage = value;
            }
        }
        private void Form2_Load(object sender, EventArgs e)
        {
            textBox1.Text = strMessage;
        }
    }
}
```

9. Press the F5 key from the keyboard to debug and execute the application.
10. Enter the text, Welcome to .NET Programming, in the TextBox1 placed on Form1 and then click the Read Text from First Form button, as shown in Figure 17:

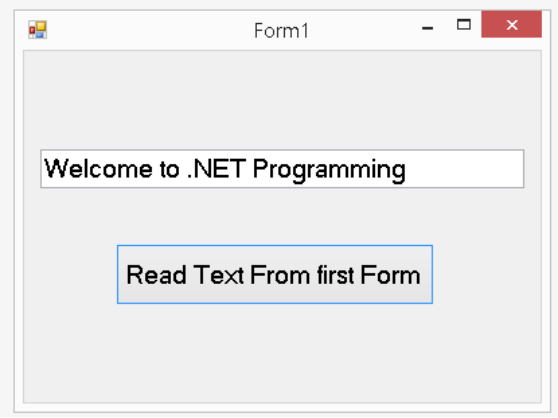


Figure 17: Displaying the First Form

- The control is transferred to Form2 and the text of TextBox1 of Form1 is displayed in a TextBox1 in Second Form, as shown in Figure 18:

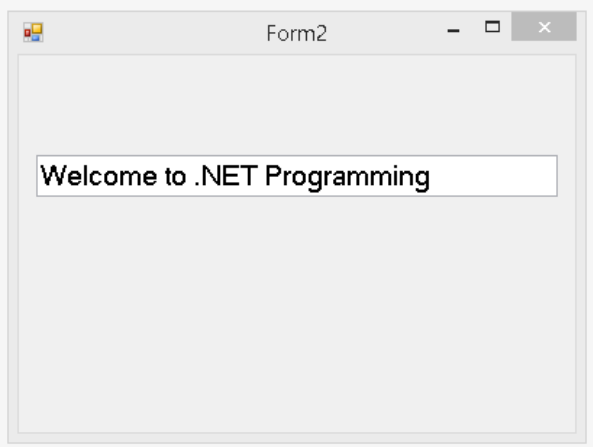


Figure 18: Displaying the Multiform Application

Setting the Startup form

- To change which form will execute first you have to perform the following steps:
  1. Right-click your project in the Solution Explorer, a context menu appears (Figure 19).
  2. Select Properties from the context menu, as shown in Figure 19:

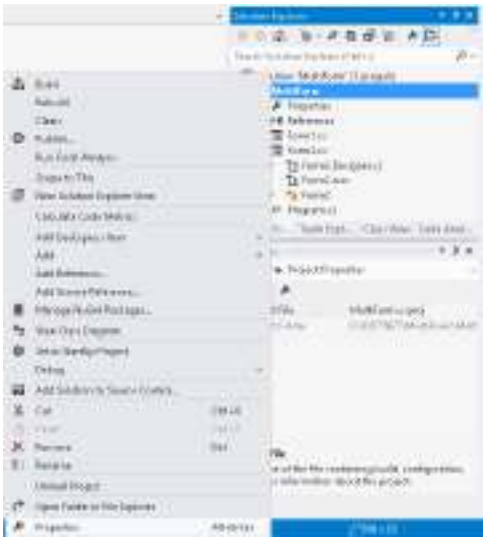


Figure 19: Setting the Properties Option

The project Designer appears (Figure 20).

3. Select the application tab in the left pane. Next select Form2 from the drop-down list of the Startup form option on the right, as shown in Figure 20.

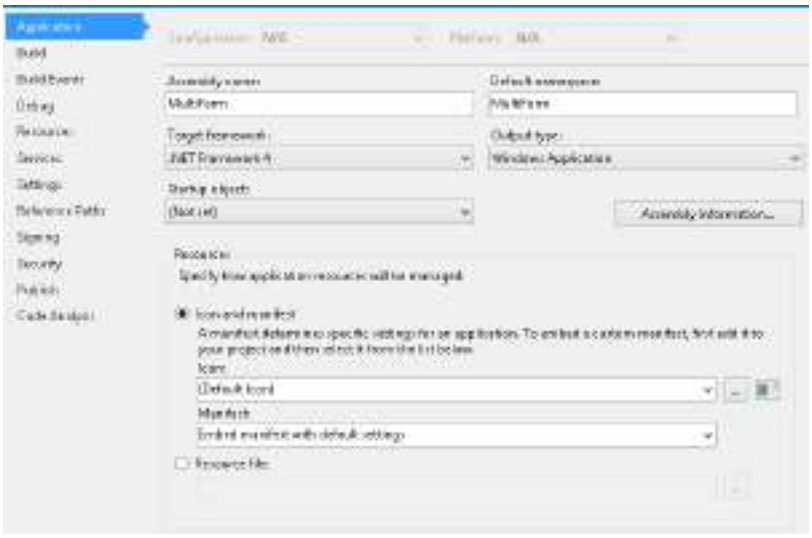


Figure 20: Setting Startup Form

- Now the program displays the form you have selected first when the program runs.
- Let's learn about the properties and events of Windows Forms.

Windows Forms Properties and Events

- The Form, class has only one class property ActiveForm, which holds the currently active form for the entire application. If you want to determine which window is active, use the ActiveForm property. However, the Form class does have many object properties.
- Some of these object properties are public some private to the object, and some protected (that is, only accessible to objects of the Form class or objects of classes derived from Form). When working with forms, one usually uses the public object members (that is public properties, methods, and events of Form objects).
- Note that, as is usual with properties and methods in C#.NET, not all these properties and methods will be available at the time you are designing your code. Only some of them will be available at runtime.

Property	Description
AcceptButton	Obtains or sets the button on the form when the user presses the Enter key
ActiveForm	Retrieves the presently active form of the application
ActiveMdiChild	Retrieves the presently active multiple document interface (MDI) child window
AllowTransparency	Obtains or sets the value to adjust the opacity of the form
AutoScaleBaseSize	Obtains or sets the base size used for autoscaling the form
AutoScroll	Indicates whether the form implements autoscrolling
AutoSize	Resizes the form as per the settings of the AutoSizeMode property
AutoSizeMode	Gets or sets the mode to make changes in the size of the form automatically
AutoValidate	Obtains or sets the value that points whether the controls will be validated automatically when their focus is changed
BackColor	Obtains or sets the form background color

CancelButton	Obtains or sets the Button control that is clicked when the user presses the ESC key
ClientSize	Obtains or sets the size of the main area of Form
ControlBox	Retrieves or sets a value points whether a control box is displayed on the form's caption bar
DialogResult	Retrieves or sets the value of the dialog result property for the form when it is displayed as modal dialog box
FormBorderStyle	Gets or sets the form's border style
HelpButton	Retrieves or sets the value points whether the Help button should appear in the form 's caption box
Icon	Gets or sets an icon for the form
IsMdiChild	Retrieves or sets a value that points if the form is an MDI child form
IsMdiContainer	Obtains or sets a value that points if the form is a container for MDI child forms
IsRestrictedWindow	Obtains a value that determines whether the form will use all the windows and user input events without any restriction
KeyPreview	Retrieves or sets a value that determines whether the form will receive all the key events before an event is passed to an active control
Location	Gets or sets the location of the form
MainMenuStrip	Retrieves or sets the primary menu container for the form
MaximizeBox	Obtains or sets a value that determines if the Maximize button is displayed in the caption bar of the form
MaximumSize	Obtains the value to determine the maximum size of the form
MdiChildren	Retrieves an array of MDI child forms parented to a form
MdiParent	Obtains or sets the current MDI parent form of the form
MinimizeBox	Obtains or sets the value that determines whether the Minimize button is displayed in the caption bar of the form
MinimumSize	Retrieves the minimum size the form can be resized to
Opacity	Obtains or sets the opacity level of the form
RightToLeftLayout	Obtains or sets the value that determines whether the right-to-left placement is turned on
ShowInTaskbar	Obtains or sets a value that determines whether the form is displayed in the Windows Taskbar
ShowIcon	Retrieves or sets the value that determines whether an icon (picture representing the form on the taskbar) is displayed in the caption bar of the form
Size	Retrieves or sets the size of the form
StartPositlon	Retrieves or sets the starting position of the form at run time
Text	Retrieves or sets the text associated with the form
TopMost	Obtains or sets a value that determines whether the form should be displayed as the topmost form of the application
TransparencyKey	Retrieves or sets the color representing the transparent areas of the form
WindowState	Retrieves or sets a value that determine whether a window is normal, minimized, or maximized

Table 3: Properties of the Form Class

Method	**Associated Function**

Activate()	Activates the form by giving it focus
AddOwnedForm()	Adds an owned form to the parent form
Close()	Closes the form
LayoutMdi()	Arranges MDI child forms within the MDI parent form
RemoveOwnedForm()	Removes the owned form from the parent form
SetDesktopBounds()	Sets the bounds of the form in desktop coordinates
SetDesktopLocation()	Sets the location of the form in desktop coordinates
Show()	Displays the form as an object
ShowDialog()	Displays the form as a modal dialog box
ToString()	Gets a string representing the current instance of the form
ValidateChildren()	Validates all selected child controls in the form

Table 4: Methods of the Form Class

Event	**Description**
Load	Raises before a form is displayed for the first time
MaximumSizeChanged	Raises when you have changed the value for the MaximumSize property
MaxmizedBoundsChanged	Raises when you have changed the value for the MaximixedBounds property
MdiChildActivate	Raises when an MDI child form is activated or closed within an MDI application
MenuComplete	Raises when the menu of the form loses focus
MenuStart	Raises when the menu of the form receives focus
MinimumSizeChanged	Raises when you have changed the value of the MaximumSize property
ResizeBegin	Raises when you trying to resize the form
ResizeEnd	Raises when you exits from resizing the form
RightToLeftLayoutChanged	Raises when the RightToLef tLayout property changes
Shown	Raises when the form is displayed for the first time
TablndexChanged	Raises when the value of the Tab index property changes

Table 5: Event of the Form Class

Dialogs

- There are several built-in dialog boxes in Windows Forms. The built-in dialog boxes reduce the time and work required for developing commonly used dialog boxes such as file open, file save and other dialog boxes. Some of the dialog box controls are OpenFileDialog, SaveFileDialog and FontDialog.
- The ShowDialog() method is used to display the dialog box at run time. You can check the return value of the ShowDialog () method (such as DialogResult.OK or DialogResult.Cancel() to retrieve the button clicked by a user. The possible dialog box returns values from the method to the DialogResult enumeration as follows:
    - Abort: Returns an Abort value when the user clicks a button labeled Abort
    - Cancel: Returns a Cancel value when the user clicks a button labeled Cancel
    - Ignore: Returns an ignore value when the user clicks a button labeled Ignore

- No: Returns a No value when the user clicks a button labeled No
- None: Returns nothing. This means that the modal dialog box continues running
- OK: Returns an OK value when the user clicks a button labeled OK
- Retry: Returns a value when the user clicks a button labeled Retry
- Yes: Returns a value when the user clicks a button labeled Yes

- Let's learn more about dialog boxes in the following sections
  - Using the FolderBrowserDialog control
  - Using the OpenFileDialog control
  - Using the SaveFileDialog control
  - Using the FontDialog control
  - Using the ColorDialog control
  - Creating a user-defined dialog Box

Using the FolderBrowserDialog control

- As the name indicates, the FolderBrowserDialog control opens the Browse for Folder dialog box which lets the user select a folder. The dialog box is seen in Figure 21:

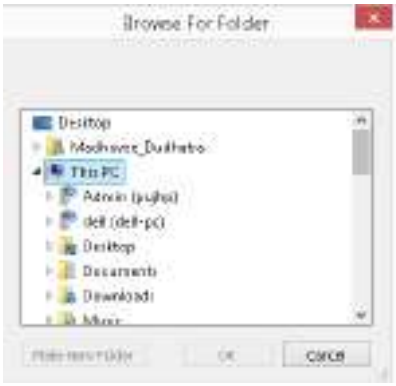


Figure 21: Showing the browser for FolderDialogBox

- The FolderBrowserDialog control is based on the FolderBrowserDialog class, which has the following class hierarchy:

```
System.Object
        System.MarshalByRefObject
                System.ComponentModel.Component
                        System.Windows.Forms.CommonDialog
                                System.Windows.Forms.FolderBrowserDialog
```

Property	Description
Description	Retrieves or sets the descriptive text displayed above the TreeView control in the dialog box
RootFolder	Retrieves or sets the root folder where the browsing starts
SelectedPath	Retrieves or sets the path selected by the user
ShowNewFolderButton	Retrieves or sets a value indicating whether or not the New Folder button appears in the folder dialog box

Table 6: Properties of the FolderBrowserDialog class

- The Browse for Folder dialog box is supported by the FolderBrowserDiaiog class The dialog box also shows the Make New Folder button if the ShowNewFolderButton property is set to True (by default).
- The Browse for Folder dialog box prompts the user to browse and select an existing folder or create a new one and select it. Note that the FolderBrowserDiaiog class is used only to allow the user to select folders, not files and virtual folders.

Event	Description
HelpRequest	Occurs when the Help button on the dialog box is clicked by the user

Table 7: Events of the FolderBrowserDiaiog class

Using the OpenFileDialog control

- The OpenFileDialog control allows you to select a file to open from the Open dialog box. It enables the user to check if a file exists and then opens it. You can also determine whether a read-only check box appears in the dialog box by enabling the ShowReadOnly property to True.
- If you enable the ReadOnlyChecked property to True, the read-only check box appears checked. You can see the Open dialog box in Figure 22:
- The Open dialog box is based on the OpenFileDialog class, which has the following class hierarchy:

System.Object
System.MarshalByRefObject
System.ComponentModel.Component
System.Windows.Forms.CommonDialog
System.Windows.Forms.FileDialog
System.Windows.Forms.OpenFileDialog

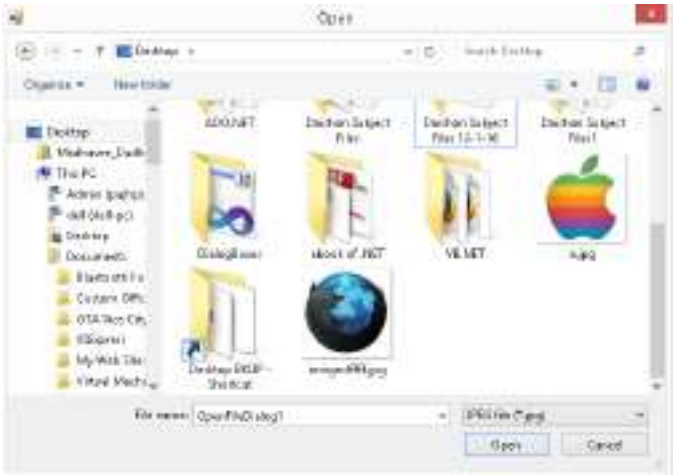


Figure 22: Displaying the OpenDialogBox

- Table 9 lists of the properties of OpenFileDialog class:

Property	Description

CheckFileExists	Retrieves or sets a value indicating if the dialog box displays a warning if the user specifies a non-existent file.
Multiselect	Retrieves or sets a value specifying whether or not the dialog box allows multiple file selections.
ReadOnlyChecked	Retrieves or sets a value signifying whether or not the read-only checkbox is selected on the dialog box.
SafeFileName	Retrieves the file name and extension for the selected file in the dialog box. The file name does not include the path.
SafeFileNames	Retrieves an array of file names and extensions for all the selected files in the dialog box. The file names do not include the path.
ShowReadOnly	Retrieves or sets a value signifying whether or not the dialog box displays a read-only check box.

Table 9: Properties of the OpenFllePialog Class.

Table 10 lists the Methods of the OpenFileDialog class:

Method	Description
OpenFile()	Opens the file selected by the user, with read-only permission. The file is specified by the FileName property
Reset()	Resets all the options to their default values.

Table 10: Methods of the OpenFllePialog Class.

- The OpenFileDialog class supports the Open dialog box, which allows you to retrieve the names of files to open. If the Multiselect property is set to True, the users can also select multiple files.
- The ShowReadOnly property is used to determine if a read-only checkbox appears in the dialog box. The ReadOnlyChecked property indicates whether the read-only checkbox is selected.
- The Filter property sets the current file name filter string, which determines the file extension choices that appear in the dialog box. The name and path selected by the user is stored in the FileName property of the OpenFileDialog object.
- You can use the OpenFile() method to open the selected file directly.

Using the SaveFileDialog control

- The SaveFileDialog control supports the Save As dialog box that allows the user to specify the name of a file to save data to. You can use the ShowDialog() method to display the dialog box at run time. A Save As dialog box is shown in Figure 23:

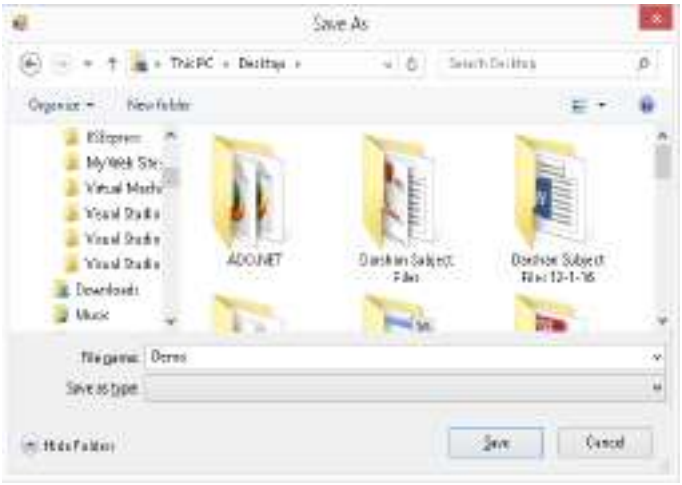


Figure 23: Displaying the SaveFileDialogBox

- The class hierarchy for the SaveFileDialog class is as follows:
  System.Object
    System.MarshalByRefObject
      System.ComponentModel.Component
        System.Windows.Forms.CommonDialog
          System.Windows.Forms.FileDialog
            System.Windows.Forms.SaveFileDialog
- Table 11 lists the Properties of the SaveFileDialog class:

Property	Description
CreatePrompt	Retrieves or sets a value specifying whether or not the dialog box asks the user if it should create a file if the user specifies a non-existent file
OverwritePrompt	Retrieves or sets a value specifying whether or not the dialog box displays a warning if the user specifies a name that already exists

Table 11 Properties of the SaveFileDialog class

- Table 12 Methods of the SaveFileDialog class:

Method	Description
OpenFile()	Opens the file with read/write permission
Reset()	Resets all dialog box options to their default values.

Table 12 Methods of the SaveFileDialog class:

Using the FontDialog control

- The Font dialog box lets the user select the font, font size, and color.
- It returns Font and Color objects directly (using the properties of the same name), which can be used in controls such as rich text boxes.
- This saves the trouble of creating and configuring these objects from scratch. A Font dialog box is displayed, as shown in Figure 24:

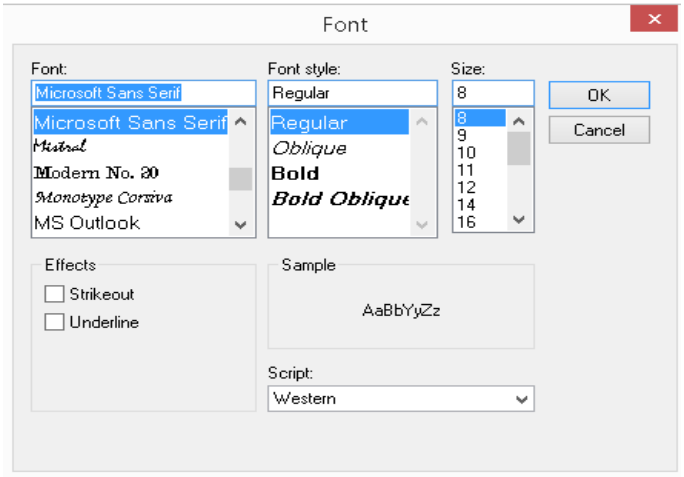


Figure 24: Displaying the FontDialogBox

- The ShowDialog() method is called to display the Font dialog box. This dialog box shows the list boxes for Font, Font style, and Size, check boxes for effects such as strikeout and Underline, a drop-down list for Script (Script refers to different character scripts that are available for a given font for example, Hebrew), and a sample of how the font appears. You can recover these settings using properties of the Font object returned by the Font property.
- The FontDialog class displays a dialog box that lets the user select a font, font style, and size. It returns a Font object in the Font property, and a Color object in the Color property.
- The class hierarchy for the FontDialog class is as follows:

System.Object
System.MarshalByRefObject
System.ComponentModel.Component
System.Windows.Forms.CommonDialog
System.Wi ndows.Forms.FontDialog

- Table 13 Properties of the FontDialog class:

Property	**Description**
AllowSimulations	Retrieves or sets a value specifying whether or not the dialog box allows Graphics Device Interface (GDI) font simulations. GDI is responsible for the drawing lines, curves, and rendering of fonts.
AllowVectorFonts	Retrieves or sets a value specifying whether or not the dialog box allows vector font selections. Vector fonts refer to the connection of lines between a series of dots that can be scaled to different sizes.
AllowVerticalFonts	Retrieves or sets a value specifying whether the dialog box displays either both vertical and horizontal fonts or only horizontal fonts.
Color	Retrieves or sets the selected font color.
FixedPitchOnly	Retrieves or sets a value specifying whether or not the dialog box allows the selection of fixed-pitch fonts, which refers to the fonts having every character with the same width.
Font	Retrieves or sets the selected font.

FontMustExist	Retrieves or sets a value specifying whether or not the dialog box specifies an error condition if the user attempts to select a font or style that does not exist.
MaxSize	Retrieves or sets the maximum font size a user can select. The default value for font size is 0.
MinSize	Retrieves or sets the minimum font size a user can select.
ShowApply	Retrieves or sets a value specifying whether or not the dialog contains an Apply button.
ShowColor	Retrieves or sets a value specifying whether or not the displays the color choice.

Table 13 Properties of the FontDialog class

- Table 14 Methods of the FontDialog class

Property	Description
ShowEffects	Retrieves or sets a value specifying whether or not the dialog box contains controls that allow the user to specify strikethrough, underline and text color options.
ShowHelp	Retrieves or sets a value specifying whether or not the dialog box displays a Help button.

Table 14 Methods of the FontDialog class

Using the ColorDialog control

- The Color dialog box lets the user select a color. The principal property you can use of this dialog box is the Color property, which returns a Color object. A Color dialog box is shown in Figure 25:

Figure 25: Displaying the ColorDialogBox

- In Figure 25. you can use the Color dialog box (by clicking the Define Custom Colors button) to define your own colors with color values and hue, saturation, and luminosity.
- On the other hand, if you set the AllowFullOpen property to False, the Define Custom Colors button is disabled and the user can select colors only from the predefined colors in the palette.
- The class hierarchy for the ColorDialog class is as follows:

System.Object
    System.MarshalByRefObject

System.ComponentModel.Component
System.Windows.Forms.CommonDialog
System.Windows.Forms.ColorDialog

- Table 15 Properties of the ColorDialog class:

Property	Description
AllowFullOpen	Retrieves or sets a value specifying whether or not the user can use the dialog box to define custom colors
AnyColor	Retrieves or sets a value specifying whether or not the dialog box displays all available colors in the set of basic colors
Color	Retrieves or sets the color selected by the user for the text
CustomColors	Retrieves or sets the set of custom colors shown in the dialog box
FullOpen	Retrieves or sets a value specifying whether or not the controls used to create custom colors are visible when the dialog box is opened
ShowHelp	Retrieves or sets a value specifying whether or not a Help button appears in the color dialog box
SolidColorOnly	Retrieves or sets a value specifying whether or not the dialog box restrict users to selecting solid colors only and not dithered colors

Table 15: Properties of the ColorDialog class

Creating a User-Defined Dialog Box

- Sometimes, dialog boxes are so necessary that nothing can substitute them. C#.NET supports message boxes and Input boxes, but they are very basic. In real applications, you need to create custom dialog boxes.
- To create a dialog box, first create a Windows form, Form1, and then add one button and one text box in this form. In addition, add a new Windows form. Form2, two buttons and a Label with the captions, OK, cancel, and Enter Your Text, respectively and a text box to this form. The design of Dialogbox Application view is as shown in Figure 26:
- In Figure 26, we have set the FormBorderStyle property Of Form2 to FixedDialog, giving it a dialog box border, and set the ControlBox property to False to remove the control box. Moreover, set the ShowInTaskbar property of Form2 to False.
- Finally, set the DialogResult property of the OK button to ok, and set the same property of the Cancel button to Cancel. This property returns a value of the DialogResult enumeration when the dialog box is closed, so you can determine which button the user has clicked. The possible settings for the DialogResult property are None, OK, Cancel, Abort, Retry, Ignore, Yes, and No.

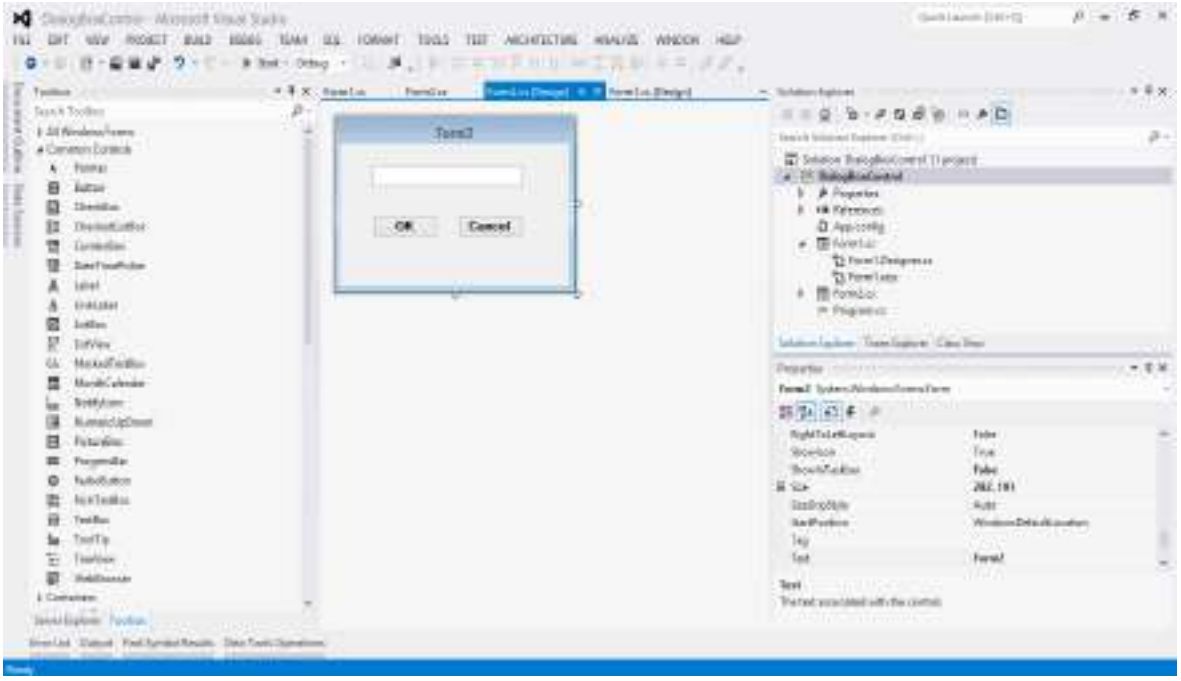


Figure 26: Creating a DialogBox

Displaying and Reading Data from Dialog Boxes

- We need some way of displaying our new dialog box from, Form1, the form that appears when the application starts. Therefore, add a button Button1, to Form1, giving it the text Click Me. In addition, we need a way of displaying the text in the dialog. Therefore, add a text box TextBox1, to Form1.
- To display the dialog box when a user clicks the Click Me button, create a new object of the Form2 form on the Click event of Click Me button of Form1.
- Now, display this form as a dialog box, by using the ShowDialog () method, but not the Show () method. This is because, the ShowDialog() method returns a DialogResult value indicating the button the user has clicked.
- When the user clicks the OK button, the text from the text box is displayed in the dialog box in the main form. The code for displaying the text is shown following:

```
private void Button1_Click(object sender, EventArgs e)
{
    Form2 frm = new Form2();
    if (frm.ShowDialog() == DialogResult.OK)
    {
        textBox1.Text = frm.textBox1.Text;
    }
}
```

- You now know how to display data using a dialog box, the next step is to create the Accept and Cancel buttons so that the user is able to accept or cancel the actions according to the requirement of the application to be executed.

Creating the Accept and Cancel Buttons

- You can easily create the OK and Cancel buttons in an application using the AcceptButton and CancelButton properties of Form2. To do so perform the following steps in the DialogBoxControl application:
    1. Add the following code to create buttons as follows:

```
private void Form2_Load(object sender, EventArgs e)
{
    this.AcceptButton = this.btnOk;
    this.CancelButton = this.btnCancel;
}
```

2. Press the F5 key from the keyboard to run the application, as shown in Figure 27

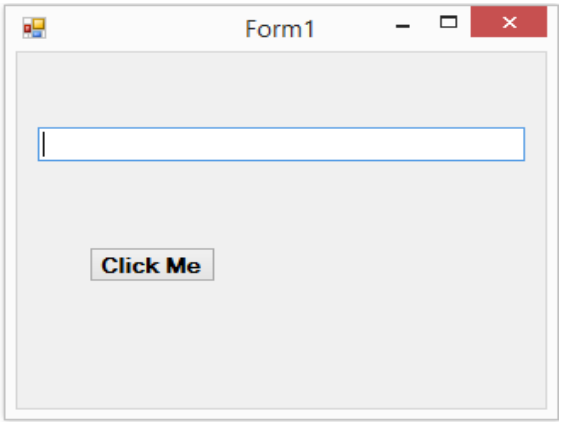


Figure 27: Displaying Form1

3. When you click the Click Me button in Form1, a dialog box appears (Figure 28).
4. Enter the text in the Enter Your Text text box and click the OK button, as shown in Figure 28.

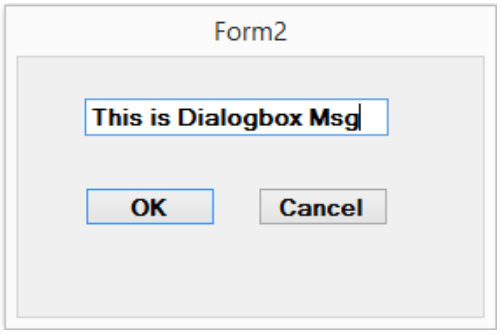


Figure 28: Displaying Form2 as Dialogbox

The dialog box disappears and your text appears in the text box of Form1, as shown in Figure 29:

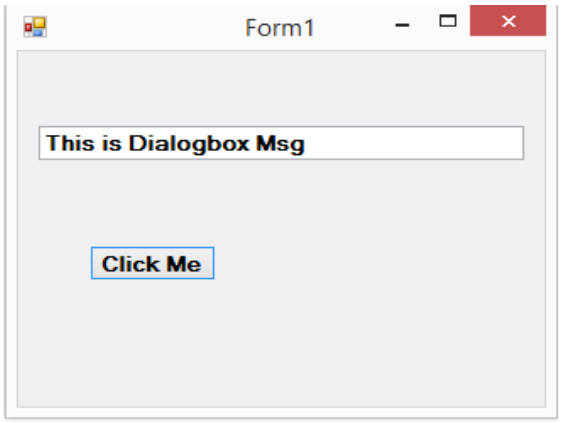


Figure 29: Receiving Text from a DialogBox

Example of DialogBox:

- Let's take example of all types of Dialog boxes, in which on the form control 5-Buttons, 5 Labels, all the types of 5 Dialog Boxes & one PictureBox arrange according to display in Figure 30 (Give name of controls according to use).

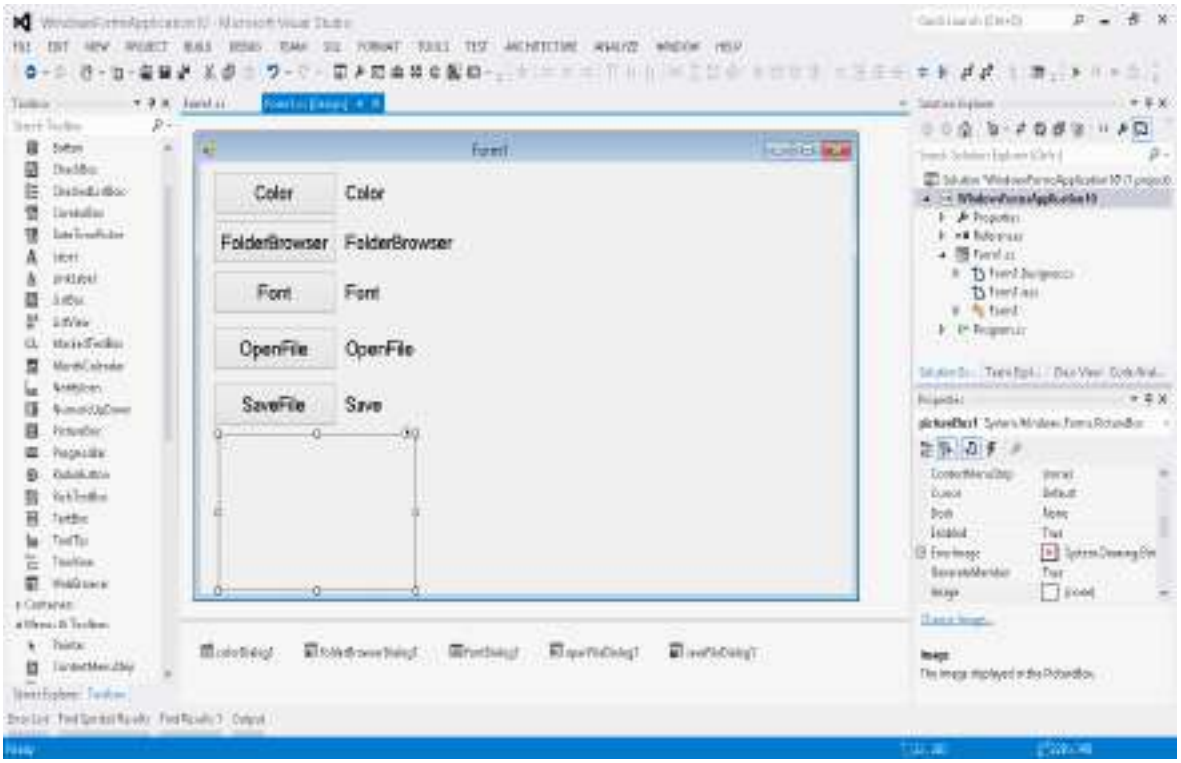


Figure 30: Example of DialogBox

- Write code of C# file as following.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Windows.Forms;
namespace DialogBoxExample
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void btColor_Click(object sender, EventArgs e)
        {
            if (colorDialog1.ShowDialog() == DialogResult.OK)
            {
                lblColor.BackColor = colorDialog1.Color;
            }
        }
        private void btFolderBrowser_Click(object sender, EventArgs e)
        {
            if (folderBrowserDialog1.ShowDialog() == DialogResult.OK)
            {
                lblFolderBrowser.Text =
folderBrowserDialog1.SelectedPath.ToString();
            }
        }
        private void btFont_Click(object sender, EventArgs e)
        {
            if (fontDialog1.ShowDialog() == DialogResult.OK)
            {
                lblFont.Font = fontDialog1.Font;
            }
        }
        private void btOpenFile_Click(object sender, EventArgs e)
        {
            if (openFileDialog1.ShowDialog() == DialogResult.OK)
            {
                lblOpenFile.Text = openFileDialog1.FileName;
                pictureBox1.Image = Image.FromFile(openFileDialog1.FileName);
            }
        }
        private void btSaveFile_Click(object sender, EventArgs e)
        {
            if (saveFileDialog1.ShowDialog() == DialogResult.OK)
            {
                lblSaveFile.Text = saveFileDialog1.FileName;
pictureBox1.Image.Save(saveFileDialog1.FileName,System.Drawing.Imaging.ImageF
ormat.Jpeg);
            }
        }
    }
}
```

- The output is as following Figure

Figure 31: Example output

ToolTips

- The ToolTip control is used to display a small window with explanatory text for an element on the Interface. The tool tip for a control or window appears when you move the mouse over the control or window.
- Following is the class hierarchy for the ToolTips class

  System.Object
  System.MarshalByRefObject
  System.ComponentModel.Component
  Systew.Windows.Forms.ToolTip

- You can associate a tool tip with any other control. To connect a tool tip with a control, you use its SetToolTip() method. For example, to connect the tool tip to Button1. you can use the following code:

  ToolTip.setToolTip(Button1, “This is a button")

- Table 17 lists properties of the ToolTip class:

Property	Description
Active	Specifies whether or not the ToolTip control is active
AutoPopDelay	Specifies the period of time the ToolTip text remains visible if the pointer is kept over a control
AutomaticDelay	Obtains or sets the time (in milliseconds) before the tool tip appears
ReshowDelay	Specifies the length of time in which the ToolTip text of two controls appear, as the pointer moves from one control to another
InitialDelay	Obtains or sets the starting delay for the tool tip
ShowAlways	Specifies whether the tool tip should appear when its parent control is not active

Table 17: Properties of the ToolTip class:

- Table 18 Methods of the ToolTip class:

Method	Description
GetToolTip()	Gets the ToolTip text specified for a control
Hide()	Hides the ToolTip window
SetToolTip()	Sets the ToolTip text for a specified control

Table 18: Methods of the ToolTip class:

- Table 19: Events of the ToolTip Class

Events	Description
Draw	Occurs in ownerDraw mode when the tool tip needs to be drawn
Popup	Occurs whenever a ToolTip is about to be shown

Table 19: Events of the ToolTip Class

- Now, let's create a new Windows Forms application, named ToolTipDemo, to learn how to work with the ToolTip control. Then, add a ToolTip control and a Button control from the Toolbox to the Form designer, as shown in Figure 32:

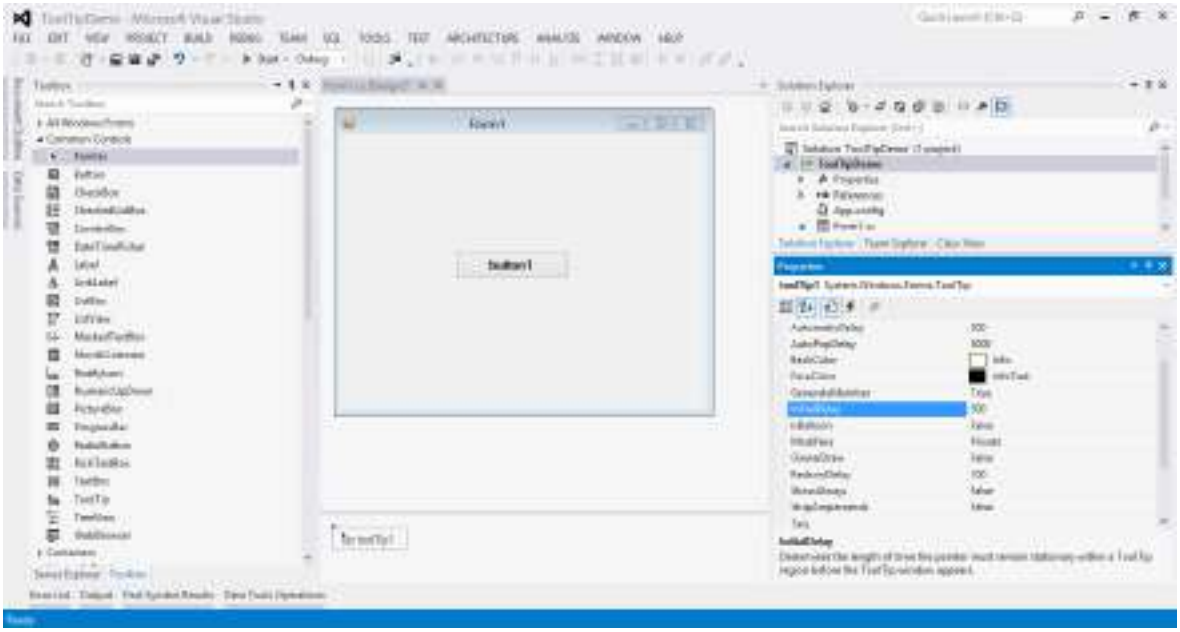


Figure 32: Designing View of ToolTipDemo Application & Setting the delay in displaying the ToolTip control

- Now, you need to perform the following two tasks:
  1. Setting the delay in displaying the ToolTip control (See in Figure 32)
  2. Setting the tool tips for controls (See in Figure 33)
- Press F5 to run program. When you take mouse on the button, you will see tool tip "This is ToolTip Text"

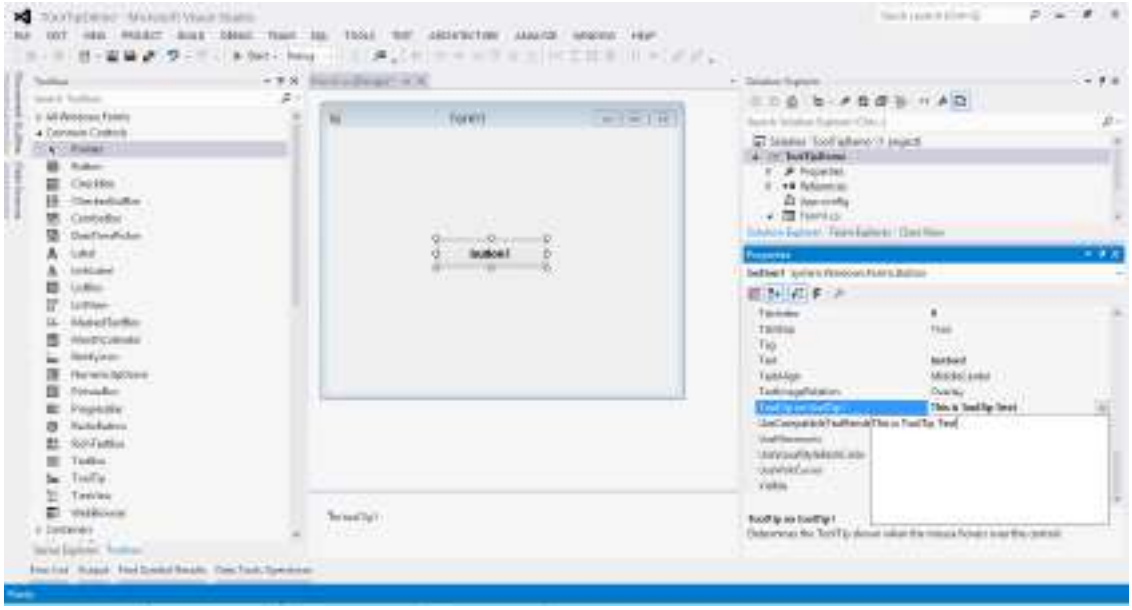


Figure 33: Setting the tool tips for controls

The Windows Forms Controls

- The Windows Form controls are used to design an interface that enhances the user instructiveness in Windows application.
- Some of the Windows Form Controls are Button, TextBox, Label and CheckBox. All of the Windows Form controls are inherited from the Control class, which is the base class for all Windows Form controls.
- The control class is placed inside the System.Windows.Forms namespace. The inheritance hierarchy for the Control class is given follow:

  System.Object
  System.MarshalByRefObject
  System.ComponentModel.Component
  System.Windows.Forms.Control
- In .NET so many controls are available which is used for create the windows application. Using that controls we can create the User Interface. We can add the controls in windows form from the ToolBox.

Common Properties:-

- All controls have a number of properties that are used to manipulate the behavior of the control. The base class of most controls, Control has a number of properties that other controls either inherit directly or override to provide some kind of custom behavior.
- The table below shows some of the most common properties of the Control class. These properties will be present in most of the controls.

Properties	Description
Anchor	Using this property, you can specify how the control behaves when its container is resized. See below for a detailed explanation of this property.
BackColor	The background color of a control.

Bottom	By setting this property, you specify the distance from the top of the window to the bottom of the control. This is not the same as specifying the height of the control.
Dock	Allows you to make a control dock to the edges of a window. See below for a more detailed explanation of this property.
Enabled	Setting Enabled to true usually means that the control can receive input from the user. Setting Enabled to false usually means that it cannot.
ForeColor	The foreground color of the control.
Height	The distance from the top to the bottom of the control.
Left	The left edge of the control relative to the left edge of the window.
Name	The name of the control. This name can be used to reference the control in code.
Parent	The parent of the control.
Right	The right edge of the control relative to the left edge of the window.
TabIndex	The number the control has in the tab order of its container.
TabStop	Specifies whether the control can be accessed by the Tab key.
Tag	This value is usually not used by the control itself, and is there for you to store information about the control on the control itself. When this property is assigned a value through the Windows Form designer, you can only assign a string to it.
Top	The top edge of the control relative to the top of the window.
Visible	Specifies whether or not the control is visible at runtime.
Width	The width of the control.

Common Events:-

- When a user clicks a button or presses a button, you as the programmer of the application, want to be told that this has happened. To do so, controls use events.
- The Control class defines a number of events that are common to the controls. The table below describes a number of events.

Events	Description
Click	Occurs when a control is clicked. In some cases, this event will also occur when a user presses Enter.
DoubleClick	Occurs when a control is double-clicked. Handling the Click event on some controls, such as the Button control will mean that the DoubleClick event can never be called.
DragDrop	Occurs when a drag-and-drop operation is completed, in other words, when an object has been dragged over the control, and the user releases the mouse button.
DragEnter	Occurs when an object being dragged enters the bounds of the control.
DragLeave	Occurs when an object being dragged leaves the bounds of the control.
DragOver	Occurs when an object has been dragged over the control.
KeyDown	Occurs when a key becomes pressed while the control has focus. This event always occurs before KeyPress and KeyUp.
KeyPress	Occurs when a key becomes pressed, while a control has focus. This event always occurs after KeyDown and before KeyUp. The difference between KeyDown and KeyPress is that KeyDown passes the keyboard code of the key that has been pressed, while KeyPress passes the corresponding char value for the key.

KeyUp	Occurs when a key is released while a control has focus. This event always occurs after KeyDown and KeyPress.
GotFocus	Occurs when a control receives focus.
LostFocus	Occurs when a control loses focus.
MouseDown	Occurs when the mouse pointer is over a control and a mouse button is pressed.
MouseMove	Occurs continually as the mouse travels over the control.
MouseUp	Occurs when the mouse pointer is over a control and a mouse button is released.
Paint	Occurs when the control is drawn.
Validated	This event is fired when a control with the CausesValidation property set to true is about to receive focus. It fires after the Validating event finishes and indicates that validation is complete.
Validating	Fires when a control with the CausesValidation property set to true is about to receive focus. Note that the control which is to be validated is the control which is losing focus, not the one that is receiving it.

The Label Control

- It represents a standard Windows label.

Properties:

- Autosize : Gets or sets a value specifying if the control should be automatically resized to display all its contents.
- BorderStyle : Gets or sets the border style for the control.
- Text : Used to Display the text.
- TextAlign : Gets or sets the alignment of text in the label.

Events:

- Leave : Occurs when the input focus leaves the control.
- LostFocus : Occurs when the control loses focus.
- TextChanged : Occurs when the Text property value changes.
- Click : Occurs when user clicks the Button.

Example :

```
private void btnShowMessage_Click(System.Object sender, System.EventArgs e)
{
    lblMessage.Text = txtMessage.Text;
}
```

The Button Control

- It represents a Windows button control.

Properties:

- BackColor : Gets or sets the background color of the control.
- DialogResult : Gets or sets a value that is returned to the parent form when the button is clicked. This is used while creating dialog boxes.
- Image : Gets or sets the image that is displayed on a button control.
- ImageAlign : Gets or sets the alignment of the image on the button control
- Font : Sets the font.
- FlatStyle : Gets or sets the flat style appearance of the button control.
- Size : Sets the size.
- Enable : By default, this property is True and you can set it to False to disable the working of button.

Events:

- GotFocus : Occurs when the control receives focus.
- TextChanged : Occurs when the Text property value changes.
- Validated : Occurs when the control is finished validating.
- Click : Occurs when user clicks the Button.
- FontChange : Occurs when font is changed.

Example :

```
private void btnShowMessage_Click(System.Object sender, System.EventArgs e)
{
    lblMessage.Text = txtMessage.Text;
}
```

The TextBox Control

- It represents a Windows text box control.

Properties:

- CharacterCasing : Gets or sets whether the TextBox control modifies the case of characters as they are typed.
- Multiline : Gets or sets a value indicating whether this is a multiline TextBox control.
- PasswordChar : Gets or sets the character used to mask characters of a password in a single-line TextBox control.

- AutoSize : Gets or sets a value indicating whether the height of the control automatically adjusts when the font assigned to the control is changed
- MaxLength : Gets or sets the maximum number of characters the user can type or paste into the text box control.
- WordWrap : Indicates whether a multiline text box control automatically wraps words to the beginning of the next line when necessary.
- Readonly : By default, this property is True and you can set it to False to make the text Readonly.
- Visible : By default, this property is True and you can set it to False to make the Textbox invisible.

Events:

- TextAlignChanged : Occurs when the TextAlign property value changes.
- TextChanged : Occurs when the text in the textbox changes.

Example :

```
private void btnShowMessage_Click(System.Object sender, System.EventArgs e)
{
    lblMessage.Text = txtMessage.Text;
}
```

The NumericUpDown Control

Properties:

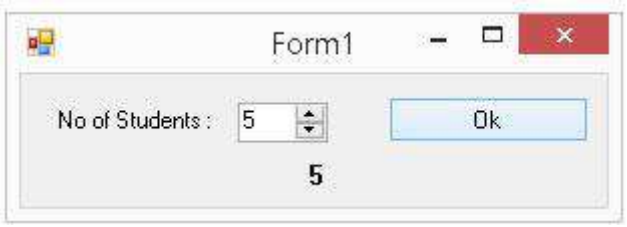
- AutoSize : Gets or sets a value indicating whether the control should automatically resize based on its contents.
- TextAlign : Gets or sets the alignment of the text in the spin box
- UpDownAlign : Gets or sets the alignment of the up and down buttons on the spin box
- Value : Gets or sets the value assigned to the spin box
- InterceptArrowKeys : Gets or sets a value indicating whether the user can use the UP ARROW and DOWN ARROW keys to select values.
- DecimalPlaces : Gets or sets the number of decimal places to display in the spin box
- Maximum : Gets or sets the maximum value for the spin box
- Minimum : Gets or sets the minimum allowed value for the spin box
- Increment : Gets or sets the value to increment or decrement the spin box

Events:

- MouseClick : Occurs when the control is clicked by the mouse
- Scroll : Occurs when the user or code scrolls through the client area

Example:

```
private void btnShow_Click(System.Object sender, System.EventArgs e)
{
    lblAnswer.Text = NumericUpDown1.Value.ToString();
}
```



The CheckBox Control

- It represents a Windows CheckBox.

Properties:

- 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.
- ThreeState : Gets or sets a value indicating whether or not a check box should allow three check states rather than two.

Events:

- AppearanceChanged : Occurs when the value of the Appearance property of the check box is changed.
- CheckedChanged : Occurs when the value of the Checked property of the CheckBox control is changed.
- CheckStateChanged : Occurs when the value of the CheckState property of the CheckBox control is changed.

Example:

```
private void btnOK_Click(System.Object sender, System.EventArgs e)
{
    lblCity.Text = "You have selected ";
    if (chkRajkot.Checked == true)
    {
        lblCity.Text += "Rajkot ";
    }
    if (chkJunagadh.Checked == true)
    {
        lblCity.Text += "Junagadh ";
    }
    if (chkBaroda.Checked == true)
    {
        lblCity.Text += "Baroda ";
    }
    if (chkAhmedabad.Checked == true)
    {
        lblCity.Text += "Ahmedabad ";
    }
}
```

The RadioButton Control

- It enables the user to select a single option from a group of choices when paired with other RadioButton controls.

Properties:

- CheckAlign : Gets or sets the location of the check box portion of the radio button.
- Checked : Gets or sets a value indicating whether the control is checked.

Events:

- AppearanceChanged : Occurs when the value of the Appearance property of the RadioButton control is changed.
- CheckedChanged : Occurs when the value of the Checked property of the RadioButton control is changed.

Example:

```
private void btnOK_Click(System.Object sender, System.EventArgs e)
{
    lblCity.Text = "Your favorite city is ";
    if (rdRajkot.Checked == true)
    {
        lblCity.Text += "Rajkot ";
    }
    if (rdJunagadh.Checked == true)
    {
        lblCity.Text += "Junagadh ";
    }
    if (rdBaroda.Checked == true)
    {
        lblCity.Text += "Baroda ";
    }
    if (rdAhmedabad.Checked == true)
    {
        lblCity.Text += "Ahmedabad ";
    }
}
```

The DateTimePicker :

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

Properties :

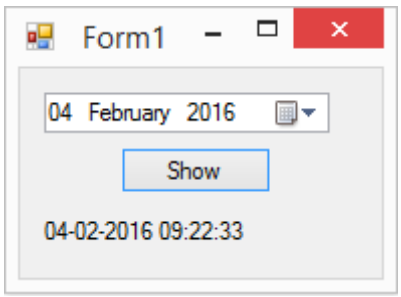
- DropDownAlign : Gets or sets the alignment of the drop-down calendar on the DateTimePicker control.
- Format : Gets or sets the format of the date and time displayed in the control.
- MaxDate : Gets or sets the maximum date and time that can be selected in the control.
- MinDate : Gets or sets the minimum date and time that can be selected in the control.
- ShowCheckBox : Gets or sets a value indicating whether a check box is displayed to the left of the selected date.
- ShowUpDown : Gets or sets a value indicating whether a spin button control is used to adjust the date/time value.
- Value : Gets or sets the date/time value assigned to the control.

Events :

- ValueChanged : Occurs when the Value property changes.
- CloseUp : Occurs when the drop-down calendar is dismissed and disappears.

Example :

```
private void btnShow_Click(object sender, EventArgs e)
{
    Label1.Text = DateTimePicker1.Value;
}
```



The ListBox :

- The ListBox represents a Windows control to display a list of items to a user.
- A user can select an item from the list. It allows the programmer to add items at design time by using the properties window or at the runtime.

Properties :

- Items : Gets the items of the list box.
- MultiColumn : Gets or sets a value indicating whether the list box supports multiple columns.
- SelectedIndex : Gets or sets the zero-based index of the currently selected item in a list box.

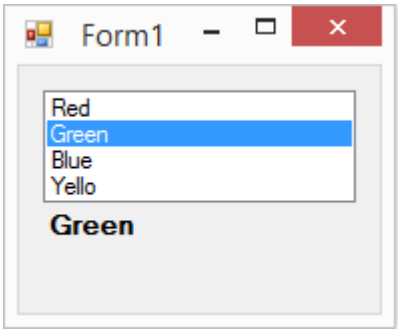
- SelectedItem : Gets or sets the currently selected item in the list box.
- SelectedValue : Gets or sets the value of the member property specified by the ValueMember property.
- Sorted : Gets or sets a value indicating whether the items in the list box are sorted alphabetically.
- TopIndex : Gets or sets the index of the first visible item of a list box.

Events :

- Click : Occurs when a list box is selected.
- SelectedIndexChanged : Occurs when the SelectedIndex property of a list box is changed.

Example :

```
private void ListBox1_SelectedIndexChanged(object sender, EventArgs e)
{
    Label1.Text = ListBox1.SelectedItem.ToString();
}
```



The ComboBox :

- The ComboBox control is used to display a drop-down list of various items. It is a combination of a text box in which the user enters an item and a drop-down list from which the user selects an item.

Properties :

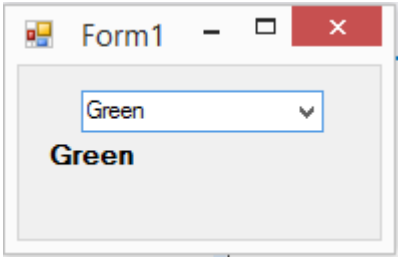
- Items : Gets the items of the list box.
- SelectedIndex : Gets or sets the zero-based index of the currently selected item in a list box.
- SelectedItem : Gets or sets the currently selected item in the list box.
- SelectedValue : Gets or sets the value of the member property specified by the ValueMember property.
- Sorted : Gets or sets a value indicating whether the items in the list box are sorted alphabetically.

Events :

- DropDownClosed : Occurs when the drop-down portion of a combo box is no longer visible.
- SelectedIndexChanged : Occurs when the SelectedIndex property of a list box is changed.

Example :

```
private void ComboBox1_SelectedIndexChanged(object sender, EventArgs e)
{
    Label1.Text = ComboBox1.SelectedItem.ToString();
}
```



The PictureBox :

- The PictureBox control is used for displaying images on the form. The Image property of the control allows you to set an image either at design time or at run time.

Properties :

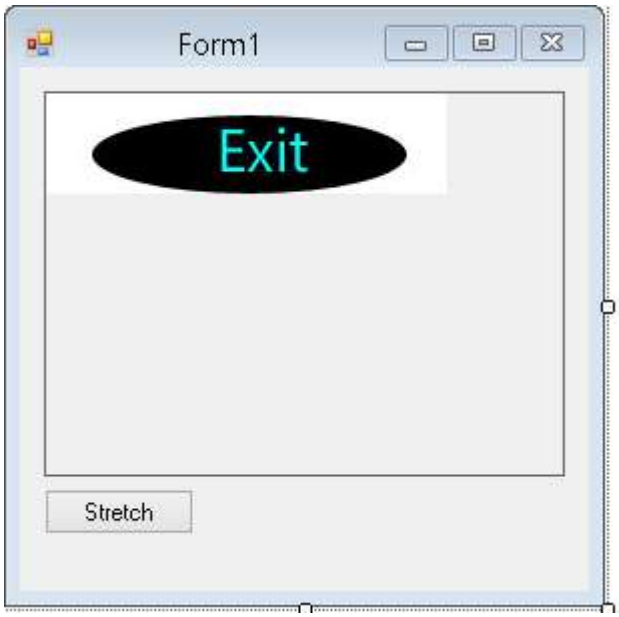
- Image : Gets or sets the image that is displayed in the control.
- ImageLocation : Gets or sets the path or the URL for the image displayed in the control.
- SizeMode : Determines the size of the image to be displayed in the control. This property takes its value from the PictureBoxSizeMode enumeration, which has values:
- Normal - the upper left corner of the image is placed at upper left part of the picture box
- StrechImage - allows stretching of the image
- AutoSize - allows resizing the picture box to the size of the image
- CenterImage - allows centering the image in the picture box
- Zoom - allows increasing or decreasing the image size to maintain the size ratio.

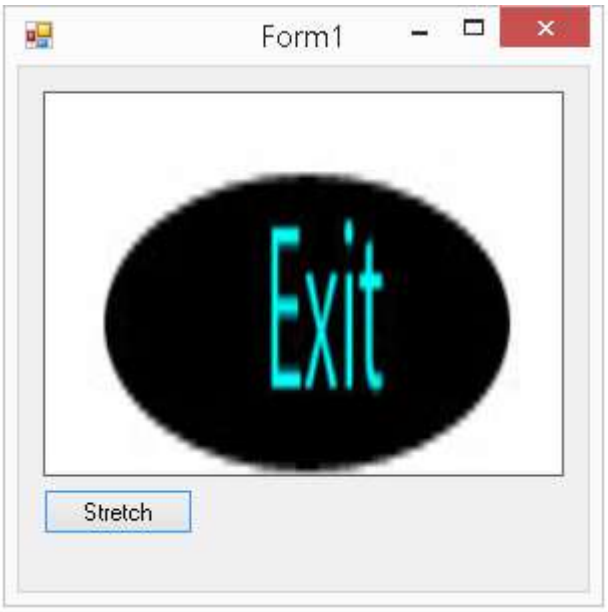
Events :

- Click : Occurs when the control is clicked.
- KeyPress : Occurs when a key is pressed when the control has focus.
- SizeModeChanged : Occurs when SizeMode changes.

Example :

```
private void btnStretch_Click(System.Object sender, System.EventArgs e)
{
    PictureBox1.SizeMode = PictureBoxSizeMode.StretchImage;
}
```





The RichTextBox :

Properties :

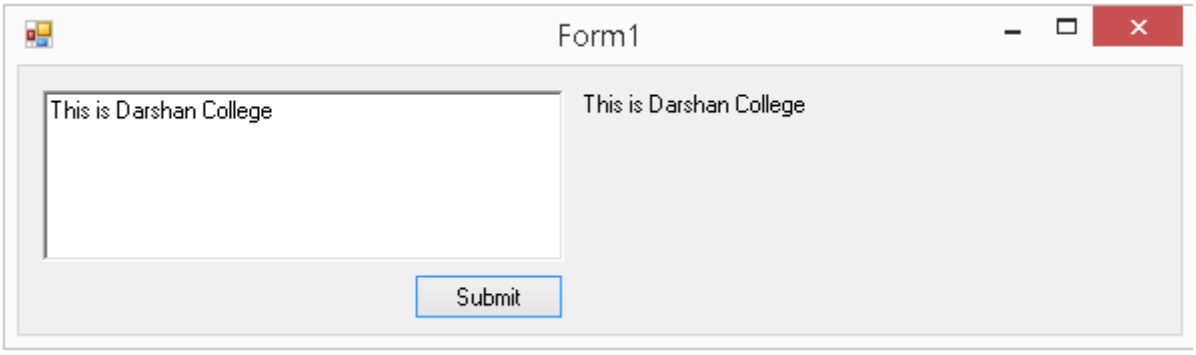
- AcceptsTab : Gets or sets a value that indicates how the text editing control responds when the user presses the TAB key
- MaxLength : Gets or sets the maximum Length of String constraint of the element
- Multiline : Gets or sets a value indicating whether this is a multiline RichTextBox control.
- ReadOnly : Gets or sets a value indicating whether text in the text box is read-only.
- ScrollBars : Gets or sets which scroll bars should appear in a multiline TextBox control. This property has values: None, Horizontal, Vertical, Both

Events :

- TextChanged : Occurs when content changes in the text element.
- VScroll : Occurs when content scroll vertically by mouse in the text element.
- HScroll : Occurs when content scroll horizontally by mouse in the text element.

Example :

```
private void btnSubmit_Click(System.Object sender, System.EventArgs e)
{
    Label1.Text = RichTextBox1.Text;
}
```



The Progressbar :

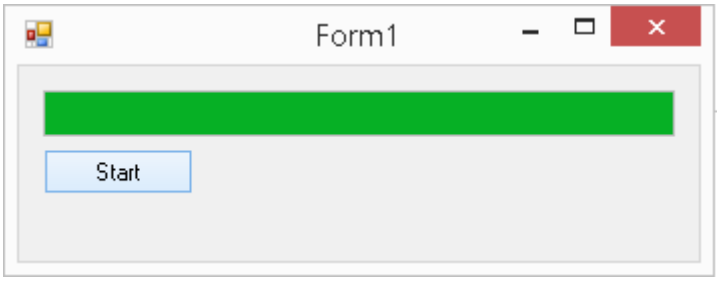
- It represents a Windows progress bar control. It is used to provide visual feedback to your users about the status of some task. It shows a bar that fills in from left to right as the operation progresses.

Properties :

- Maximum : Gets or sets the maximum value of the range of the control.
- Minimum : Gets or sets the minimum value of the range of the control.
- Step : Gets or sets the amount by which a call to the perform Step method increases the current position of the progress bar.
- Value : Gets or sets the current position of the progress bar.

Example :

```
private void btnStart_Click(System.Object sender, System.EventArgs e)
{
    int i = 0;
    for (i = 0; i <= 100; i += 1)
    {
        ProgressBar1.Value = i;
        Threading.Thread.Sleep(100);
    }
}
```



The MaskedTextBox :

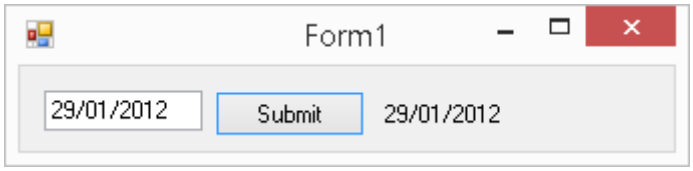
- This control is a TextBox that provides a mask that helps the user in entering a value in a particular format.
- The mask determines which characters are allowed at different positions in the text, displaying placeholder characters to help prompt the user and underscores where the user can enter characters.

Properties :

- Mask : Gets or sets the input mask to use at run time.
- PromptChar : Gets or sets the character used to represent the absence of user input in MaskedTextBox.

Example :

```
private void btnSubmit_Click(System.Object sender, System.EventArgs e)
{
    lblValue.Text = MaskedTextBox1.Text;
}
```



The LinkLabel :

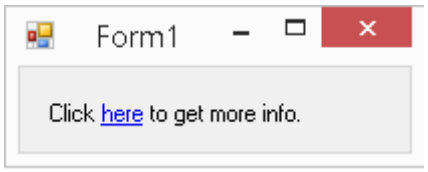
- The LinkLabel control displays a label that is associated with a hyperlink, and by default it's blue and underlined.
- In addition when you mouse over this control it displays a pointing hand cursor, which makes it easy to be recognized as a link on a web page.

Properties :

- LinkColor : this property defines the color of a normal link.
- LinkVisited : this property is a Boolean that indicates whether the link will be displayed as visited.
- VisitedLinkColor : this property defines the color of a visited link.
- LinkBehavior : this property determines when the link is underlined, and can take the values AlwaysUnderline, HoverUnderline, NeverUnderline, and SystemDefault.
- Image : The Image property of a LinkLabel control is used to set a LinkLabel background as an image.
- LinkArea : LinkArea property represents the range of text that is treated as a part of the link. It takes a starting position and length of the text.

Example :

```
private void LinkLabel1_LinkClicked(System.Object sender,
System.Windows.Forms.LinkLabelLinkClickedEventArgs e)
{
    System.Diagnostics.Process.Start(e.Link.LinkData.ToString());
}
```



The CheckedListBox :

Properties :

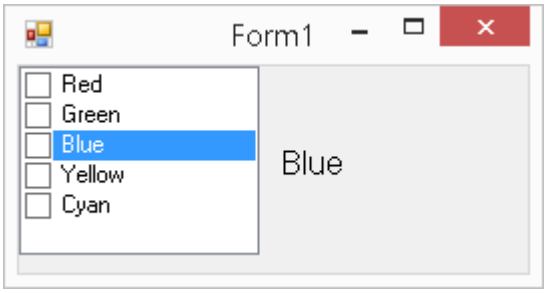
- CheckOnClick : Gets or sets a value indicating whether the check box should be toggled when an item is selected.
- ColumnWidth : Gets or sets the width of columns in a multicolumn ListBox.
- Items : Gets the collection of items in this CheckedListBox.
- MultiColumn : Gets or sets a value indicating whether the ListBox supports multiple columns.
- ScrollAlwaysVisible : Gets or sets a value indicating whether the vertical scroll bar is shown at all times.
- SelectedIndex : Gets or sets the zero-based index of the currently selected item in a ListBox.
- SelectedItem : Gets or sets the currently selected item in the ListBox.

Events :

- SelectedIndexChanged : Occurs when the SelectedIndex property or the SelectedIndices collection has changed.

Example :

```
private void CheckedListBox1_SelectedIndexChanged(System.Object sender,
System.EventArgs e)
{
    lblColor.Text = CheckedListBox1.SelectedItem.ToString();
}
```



The ScrollBars :

- The ScrollBar controls display vertical and horizontal scroll bars on the form. This is used for navigating through large amount of information.
- There are two types of scroll bar controls: HScrollBar for horizontal scroll bars and VScrollBar for vertical scroll bars. These are used independently from each other.

Properties :

- LargeChange : Gets or sets a value to be added to or subtracted from the Value property when the scroll box is moved a large distance.
- Maximum : Gets or sets the upper limit of values of the scrollable range.
- Minimum : Gets or sets the lower limit of values of the scrollable range.
- SmallChange : Gets or sets the value to be added to or subtracted from the Value property when the scroll box is moved a small distance.

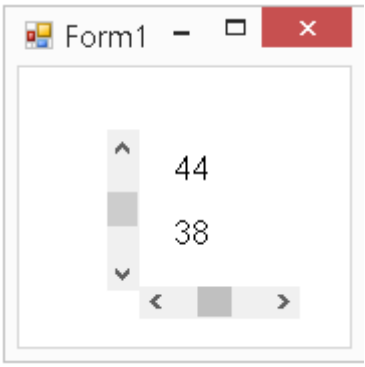
- Value : Gets or sets a numeric value that represents the current position of the scroll box on the scroll bar control.

Events :

- Scroll : Occurs when the control is moved.
- ValueChanged : Occurs when the Value property changes, either by handling the Scroll event or programmatically.

Example :

```
private void HScrollBar1_Scroll(System.Object sender,
System.Windows.Forms.ScrollEventArgs e)
{
    lblHscrollbarValue.Text = HScrollBar1.Value;
}
private void VScrollBar1_Scroll(System.Object sender,
System.Windows.Forms.ScrollEventArgs e)
{
    lblVscrollBarValue.Text = VScrollBar1.Value;
}
```



The Timer :

Properties :

- Enabled : Property used to Get or set whether the timer is running.
- Intrevale : Property used to set or get the time in millisecond between the timer clicks.

Methods :

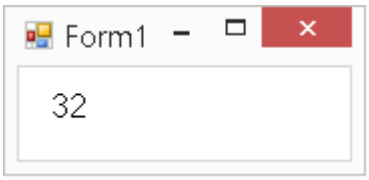
- Start : Method used to start timer.
- Stop : Method used to stop timer.

Events :

- Tick : Triggered when the time intreval has elapsed.

Example :

```
int i = 0;
private void Timer1_Tick(System.Object sender, System.EventArgs e)
{
    lblTimerValue.Text = i.ToString();
    i = i + 1;
}
```



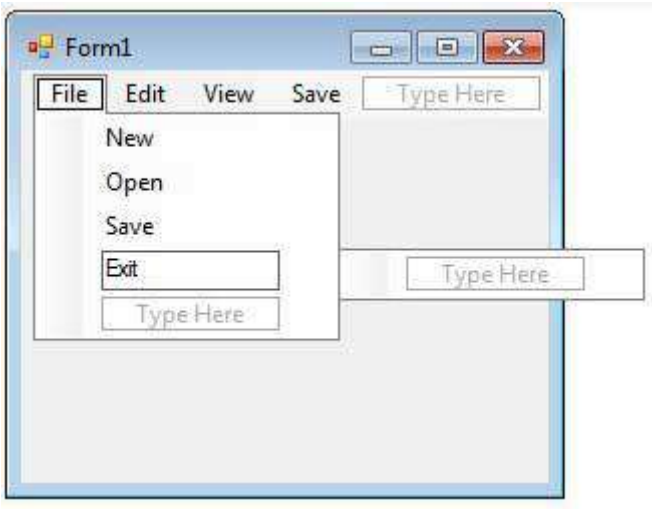
The Menu Control

The MenuStrip:

- The MenuStrip control works as the top-level container for the menu structure.

Add menu and sub-menu items :

- Drag and drop or double click on a MenuStrip control, to add it to the form.
- Click the Type Here text to open a text box and enter the names of the menu items or sub-menu items you want. When you add a sub-menu, another text box with 'Type Here' text opens below it.
- Complete the menu structure shown in the diagram above.
- Add a sub menu Exit under the File menu.



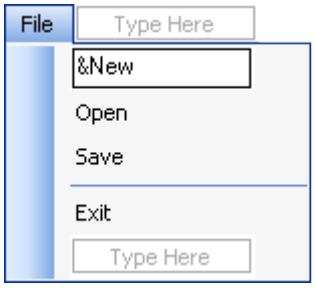
- Double-Click the Exit menu created and add the following code to the Click event of ExitToolStripMenuItem:

Example :

```
private void ExitToolStripMenuItem_Click(object sender, EventArgs e)
{
    System.Environment.Exit(0);
}
```

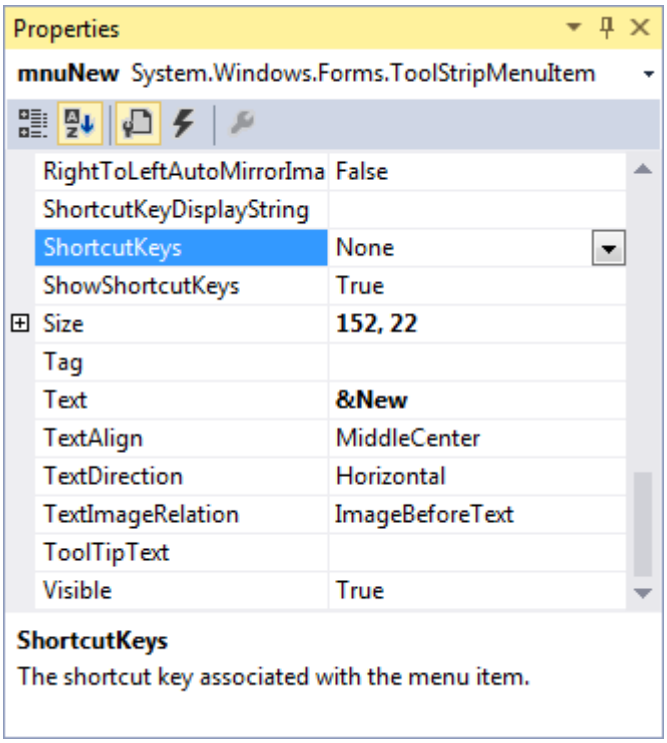
Shortcut (By using Underline) :

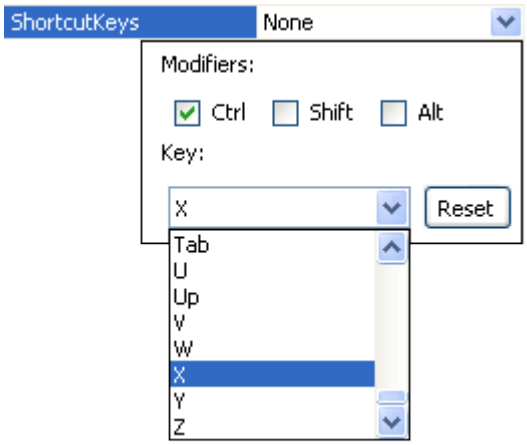
- Click on your New menu item once. This will select it
- Position your cursor before the "N" of New
- Type an ampersand symbol (&)



Shortcuts (By Key combination) :

- A key combination shortcut is one that appears at the end of a menu item (Ctrl + X, for example). You can easily add this option to your own programs. So try this:
- In Design time, select the Exit item on your menu
- Look at the properties box on the right
- Locate the ShortcutKeys item
- The Modifier is the key you press with your shortcut. For example, the CTRL key then the "X" key on your keyboard. Place a check inside the Ctrl box. Then select the letter "X" from the Key dropdown list



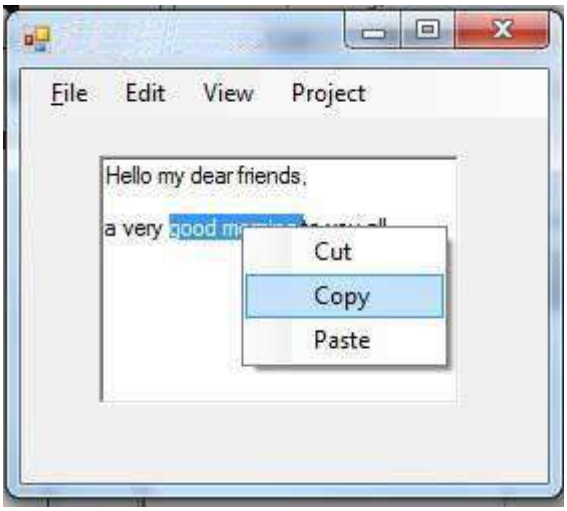


The Popup/ContextMenuStrip Menu :

- The ContextMenuStrip control represents a shortcut menu that pops up over controls, usually when you right click them. They appear in context of some specific controls, so are called context menus. For example, Cut, Copy or Paste options.
- Content/Popup menu with the menu items Cut, Copy and Paste.
- Drag and drop or double click on a ControlMenuStrip control to add it to the form.
- Add the menu items, Cut, Copy and Paste to it.
- Add a RichTextBox control on the form.
- Set the ContextMenuStrip property of the rich text box to ContextMenuStrip1 using the properties window.
- Double the menu items and add following codes in the Click event of these menus:

Example:

```
private void CutToolStripMenuItem_Click(object sender, EventArgs e)
{
    RichTextBox1.Cut();
}
private void CopyToolStripMenuItem_Click(object sender, EventArgs e)
{
    RichTextBox1.Copy();
}
private void PasteToolStripMenuItem_Click(object sender, EventArgs e)
{
    RichTextBox1.Paste();
}
```


File
Edit
View
Project
Hello my dear friends,
a very
Cut
Copy
Paste

Creating Base Forms

- Base form, also known as the parent form, is a normal Windows Form of C#.NET application.
- A from which inherits the components or controls of the base form is known as the inherited or child form.
- Let's create an application, named Inheritance, to understand the concept of base and inherited form. Now, add a Button control on the Form1 form of the Inheritance application and change the Text property of the Button control to Click, as shown in Figure:

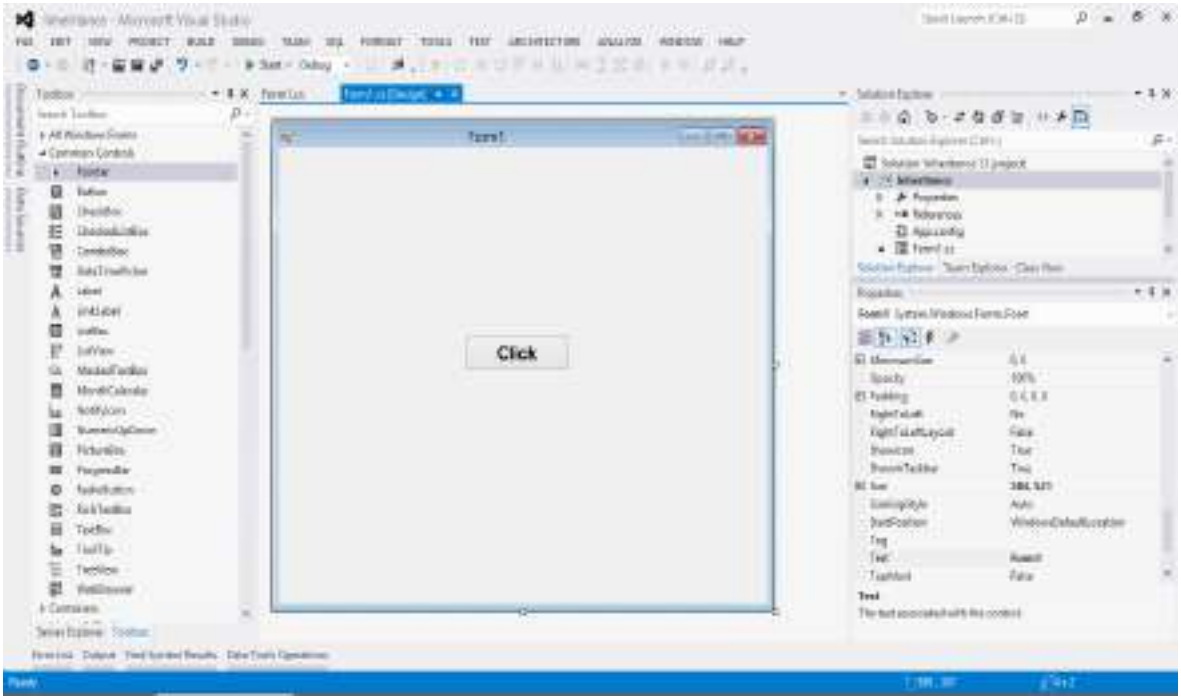


Figure: Showing the Button Control

- Now, double-click the Button control and add the code, shown as following, on the Click event of the Button control:

```
private void button1_Click(object sender, EventArgs e)
{
    MessageBox.Show("Hello Wrold");
}
```

- In code, when you click the Click button a message box appears showing a message Hello World, as shown in Figure:

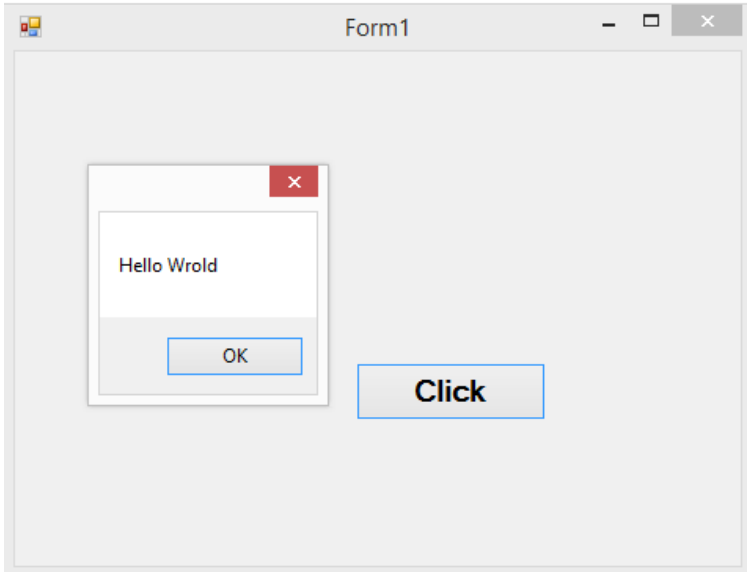


Figure: Showing the Hello World Message

- Now, right click the name of the Inheritance application in the Solution Explorer and select Properties to open the Properties page. In the Application tab, change the value of the Application type combo box to Class Library, as shown in Figure:

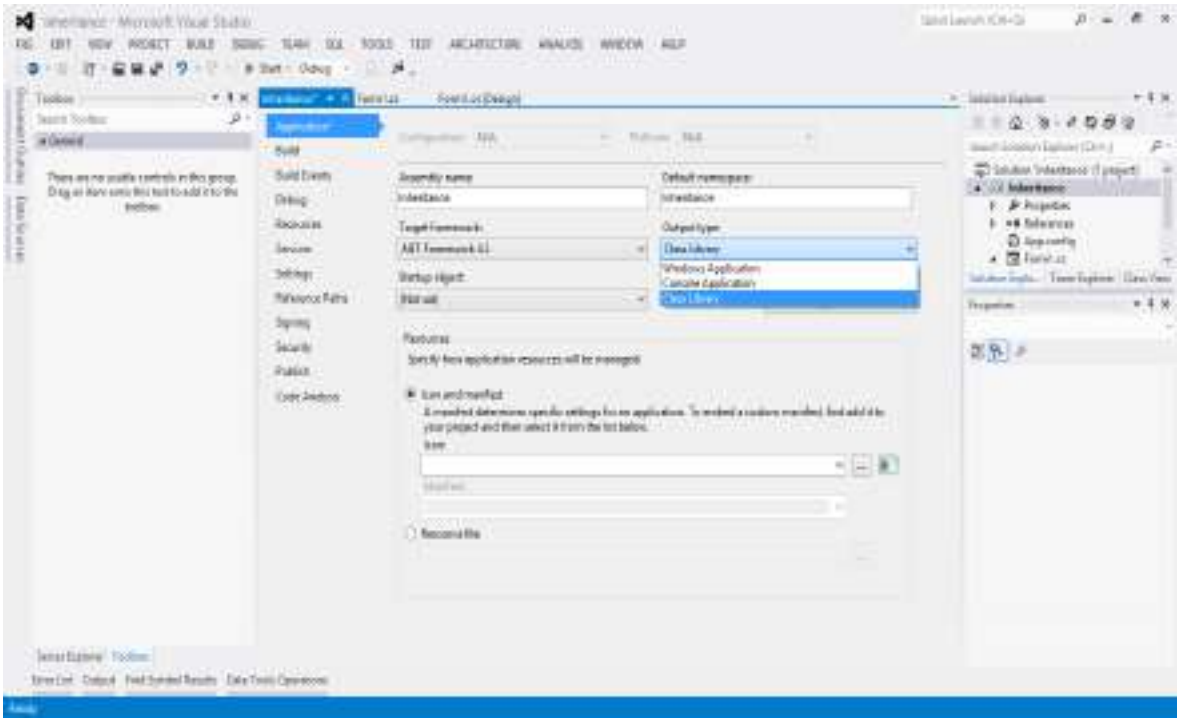


Figure: Showing the Properties Page of the Inheritance Application

- Now, rebuild the application again.
- The dll file of your application will be generated in …bin\Degug\ folder of your application as showing following figure which is used as base form.

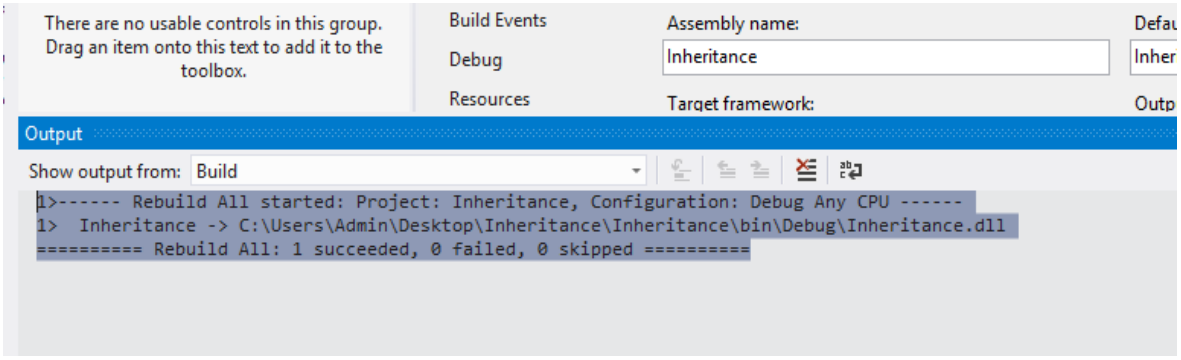


Figure: Showing the Output window with creating "Inheritance.dll" file of Inheritance Application

Apply Inheritance techniques to Forms

- Inheritance is a process of reusing the code of the base class into one or more inherited classes.
- As already discussed, that C#.NET is an object-oriented language and supports inheritance, which allows the reusability of code of a class by inheriting it into other classes.
- Let's create an application, named InheritedForm, to learn how to use inheritance and perform the following steps:
  1. Right-click the InheritedForm application in the Solution Explorer and select Add -> New Item from the context menu. This opens the Add New Item dialog box.
  2. Select the Windows Forms from the list of Common Items, and then select the Inherited Form template, as shown in Figure:

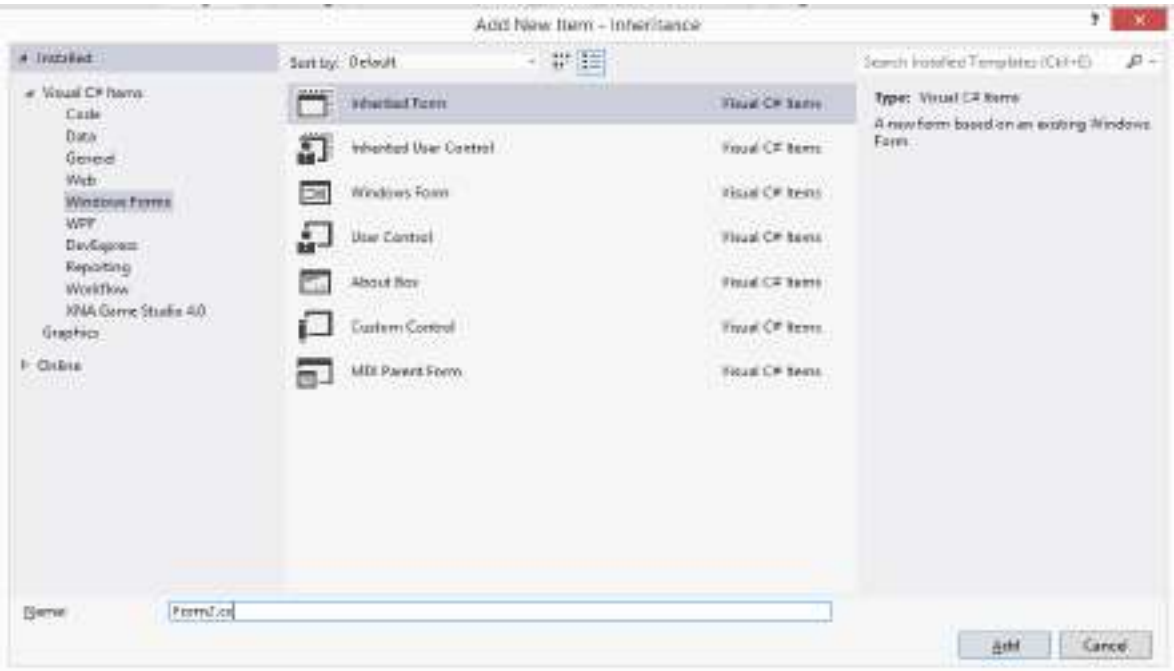


Figure: Showing the Add New Item Dialogbox

3. Click the Add button to add the inherited form. This opens the Inheritance Picker dialog box, as shown in Figure:

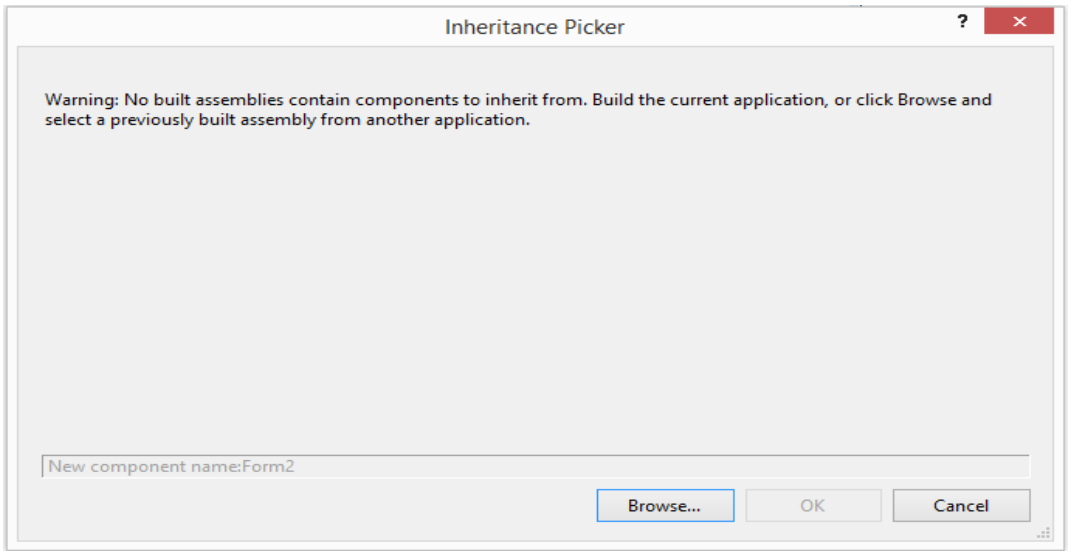


Figure: Showing the Add New Item Dialogbox

4. Click the Browse button on the Inheritance Picker dialog box and select the .dll file of the application, whose form you want to inherit in this application. In our case, we have selected the Inhertiacne.dll file, as shown in Figure:

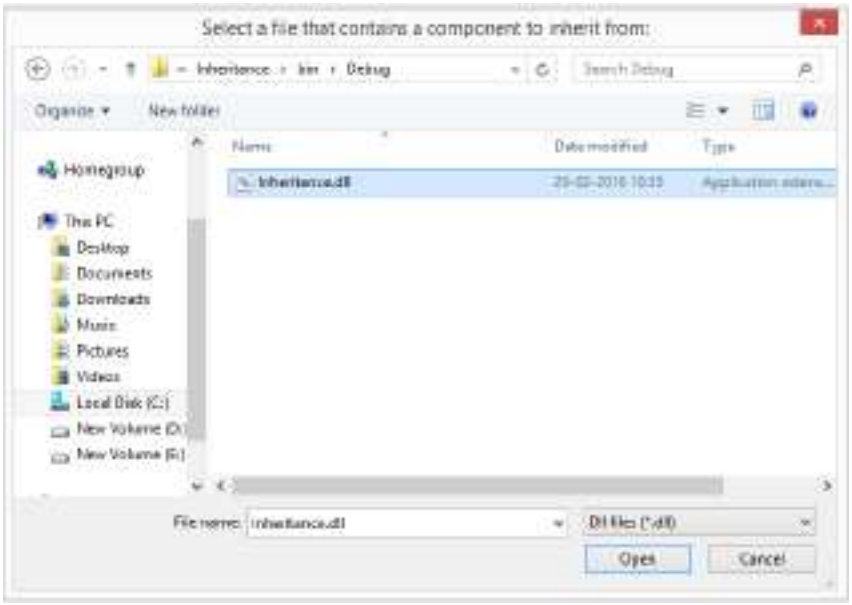


Figure: Selecting the .dll File

When you select the .dll file, it appears in the Inheritance Picker dialog box, as shown in Figure:

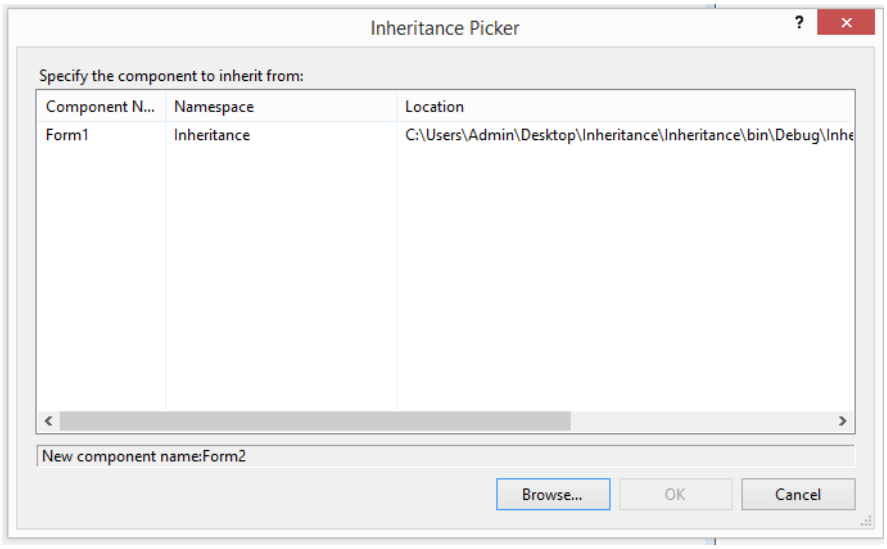


Figure: Showing the Selected .dll File in the Inheritance Picker Dialog Box

5. Click the OK button to add the inherited form in the InheritedForm application. The design view of the inherited form appears as shown in Figure:

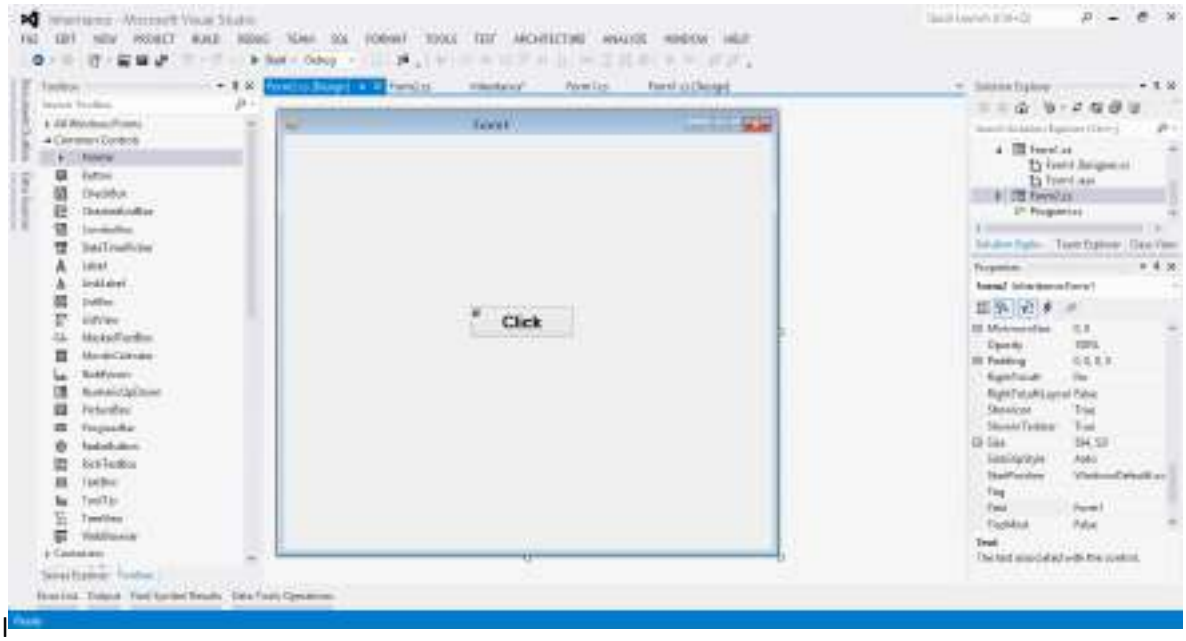


Figure: Showing the Design View of the Inherited Form

6. Right-click the name of the InheritedForm application in the Solution Explorer and select Properties option form the context menu. In the Application tab of the Properties page, change the Output Type Windows Application, as shown in Figure:

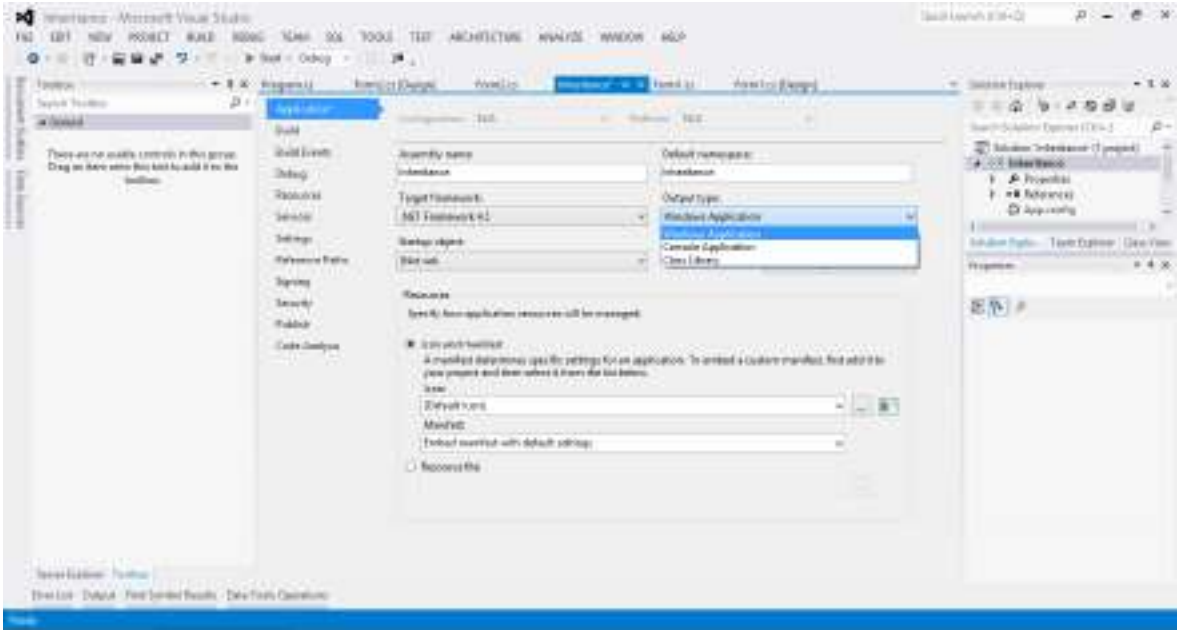


Figure: Changing the Startup Form

And change code Form1 to Form2 in Program.cs file as showing following.

```
static void Main()
{
    Application.EnableVisualStyles();
    Application.SetCompatibleTextRenderingDefault(false);
    Application.Run(new Form2());
}
```

7. Press the F5 key to run the application. A form appears Inheriting the Button control from the Inheritance application, as shown in Figure:

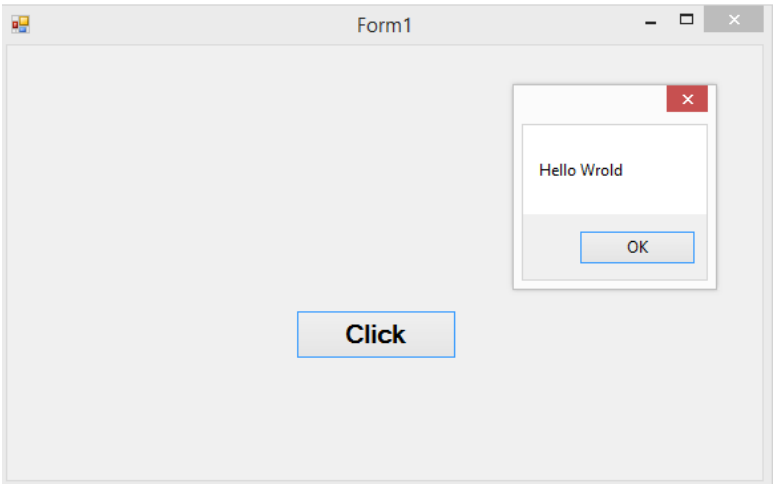


Figure: Showing the Inherited Button

When you click the Click button it also displays the message box.

Programming Derived Forms

- In the previous sections, you have learned about applying the inheritance techniques on Windows Forms to inherit the controls and other components of a base into another form.
- This process of inheritance is done with the help of the Inherited Form template of Visual Studio. In this section, you learn to derive a form using code.
- Now, let's create an application, named DerivedForm, to derive a form using code and perform the following steps:
  1. Add three Button controls and a Label control on the form and change the Text property of the Button controls to Next, Previous, and Child Form, respectively, shown in Figure

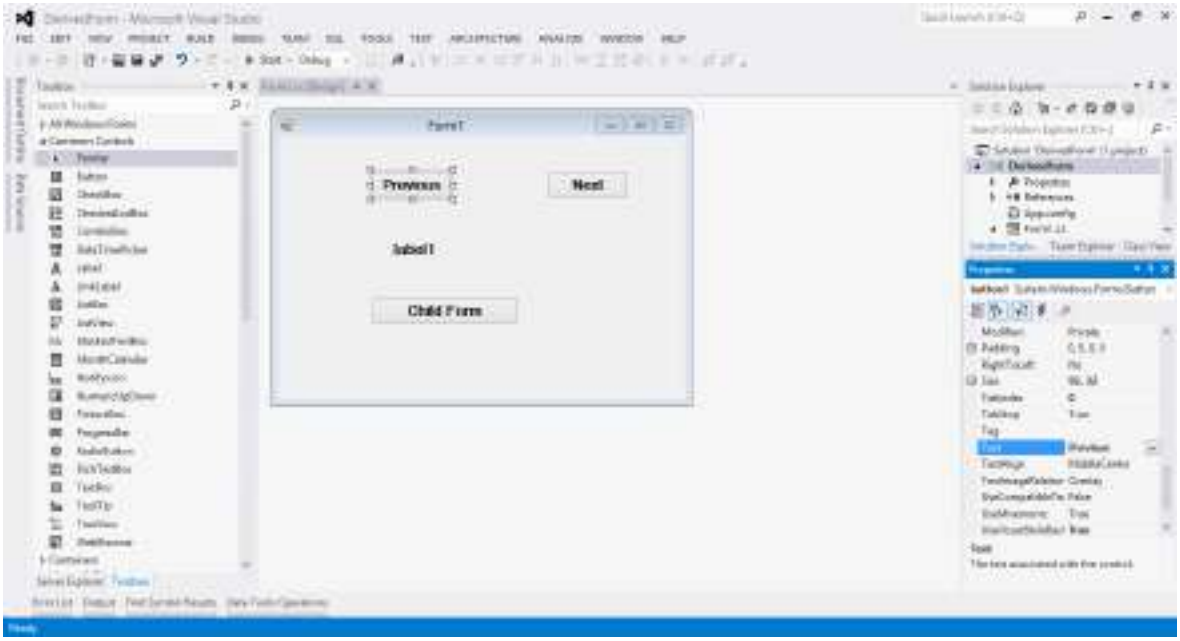


Figure: Showing the Design View of the DerivedForm Application

  2. Add the code, shown in code, in the code-behind file of the Form1 form:

```
using System;
using System.Windows.Forms;
namespace DerivedForm
{
    public partial class Form1 : Form
    {
        long LPosition;
        protected long Position
        {
            get
            {
                return LPosition;
            }
            set
            {
                LPosition = value;
            }
        }
        public Form1()
        {
            InitializeComponent();
        }
        private void button1_Click(object sender, EventArgs e)
        {
            try
            {
                Position = Position - 1;
                label1.Text = "Move Previous " + Position.ToString();
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Source + " : " + ex.Message);
            }
        }
        private void button2_Click(object sender, EventArgs e)
        {
            try
            {
                Position = Position + 1;
                label1.Text = "Move Next " + Position.ToString();
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Source + " : " + ex.Message);
            }
        }
        private void button3_Click(object sender, EventArgs e)
        {
            DerivedForm frm = new DerivedForm();
            frm.Show();
        }
    }
}
```

In coding, we have added the code on the click event of the Button controls. The Next button displays the next position in the Label control and the Previous button displays the previous position. The Child Form button opens the chid form, which is the derived form of Form1 form.

3. Add another Windows Form in the DerivedForm application, named DerivedForm.
4. Add the code to inherit the Form1 in the DerivedForm application, as shown in following code

```
public partial class DerivedForm : Form1
{
    public DerivedForm()
    {
        InitializeComponent();
    }
}
```

We have used the Inherits keyword to inherit the From1 form in the DerivedForm application.

5. Press the F5 key to run the application. The output of the DerivedForm application in shown in Figure

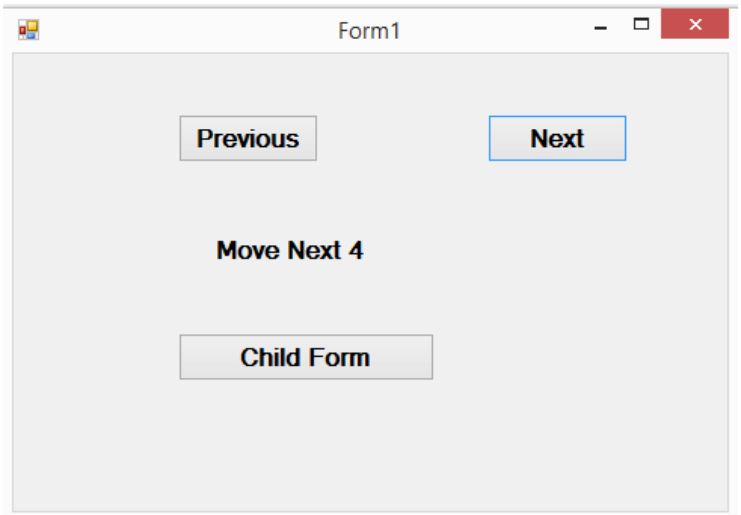


Figure: Showing the Output of the DerivedForm Application

You can click the Next and Previous button to display the position in the Label control

6. Click the Child Form button to open the derived form, as shown in Figure:

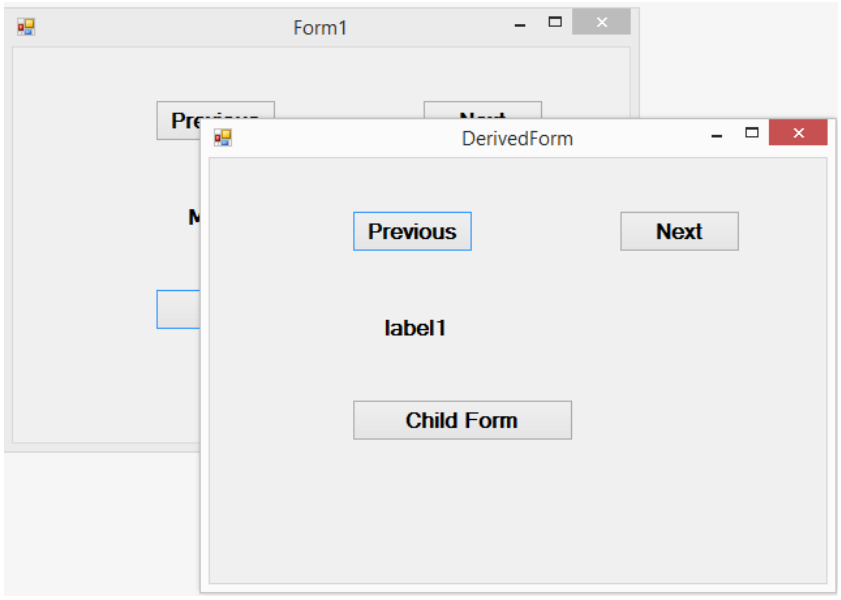


Figure: Showing the the DerivedForm

Note that the derived form contains all the controls that is added on the Form1 form.

Printing

- In C#.NET, the PrintDocument class allows you to print documents. You can add an object of this class to a project, and then handle events, such as PrintPage, which is called every time a new page is ready to print. When it is added to a form, the PrintDocument object component appears in the Component Tray.
- Besides the PrintDocument object, there are a number of controls in C#.NET that are used to print a document. Some of them are listed as follows.
  1. The PrintDocument control
  2. The PageSetupDialog control
  3. The PrintPreviewControl control
  4. The PrintPreviewDialog control
  5. The PrintDialog Control

Let's discuss about these controls one by one.

The PrintDocument control

- The PrintDocument control supports the actual events and operations of printing in Visual Basic and sets the properties that describe how to print.
- The Document property of the PrintDialog class needs to be set before calling the Print dialog box.
- This property accepts an object of the PrintDocument class, which obtains the printer settings and sends the output to the printer.
- Following is table of Properties of the PrintDocument Class

Property	Description
DefaultPageSettings	Retrieves or sets default page settings for all pages to be printed
DocumentName	Retrieves or sets the document name to display while printing the document
OriginAtMargins	Retrieves or sets a value specifying whether or not the position of a graphics object associated with a page is located just inside the user-specified margins or at the top-left area of the page corner of the printable
PrintController	Retrieves or sets the print controller for guiding the printing process
PrinterSettings	Retrieves or sets the printer that prints the document

- Following is table of Methods of the PrintDocument Class

Method	Description
OnBeginPrint()	Generates the BeginPrint event
OnEndPrint()	Generates the EndPrint event
OnPrintPage()	Generates the PrintPage event
Print()	Starts the printing operation

- Following is table of Events of the PrintDocument Class

Event	Description
BeginPrint	Occurs when the first page of the document is printed by the Print() method
EndPrint	Occurs when the last page of the document has been printed

PrintPage	Occurs when the output to print for the current page is needed
QueryPageSettings	Occurs immediately before each PrintPage event

The PageSetupDialog control

- The Page Setup dialog box lets the user specify the format for the pages that are to be printed, such as setting page orientation (portrait or landscape) and margin size.
- In other words, the PageSetupDialog control sets the page related settings for printing. The user can set the border and margin adjustments, headers and footers, and portrait or landscape orientation.
- You can use a Page Setup dialog box to modify both the PrinterSettings and PageSettings objects in a PrintDocument object to record the settings the user wants to use for printing. The PrinterSettings class is used to specify the printer related settings for the printing of a document.
- These settings are stored in the PrinterSettings property of the PageSetupDialog class which returns a PrinterSettings object. This object is assigned to the PrinterSettings property of the PrintDocument object. This makes sure that the settings the user wants are the same settings that are assigned to the document for printing.
- The PageSettings class is used to specify the page related settings for the printing of a document. The main properties of the PageSettings class are as follows:
    - Bounds: Retrieves the bounds of the page
    - Color: Retrieves a value indicating whether the page should be printed in color
    - Landscape: Retrieves a value indicating whether the page is printed in landscape or portrait orientation
    - Margins: Retrieves the margins for this page
    - Paper size: Retrieves the paper size for the page
    - Paper Source: Retrieves the page's paper source
    - PrinterResolution: Retrieves the printer resolution for the page
    - PrinterSettings: Retrieves the printer settings associated with the page
- Following is the table properties of the PageSetupDialog class:

Property	**Description**
AllowMargins	Retrieves or sets a value specifying whether or not the margins section of the dialog box is enabled.
AllowOrientation	Retrieves or sets a value specifying whether or not the orientation section of the dialog box (landscape or portrait) is enabled.
AllowPaper	Retrieves or sets a value specifying whether or not the paper section of the dialog box (paper size and paper source) is enabled.
AllowPrinter	Retrieves or sets a value specifying whether or not the Printer button is enabled.
Document	Retrieves or sets a value specifying the PrintDocument object which gets the page settings.
EnableMetric	Retrieves or sets a value specifying whether or not the margin settings, when displayed in millimetres, should be automatically converted to and from hundredths of an inch.
MinMargins	Retrieves or sets a value specifying the minimum margins the user is allowed to select. The value is measured in hundredths of an inch.
PageSettings	Retrieves or sets a value specifying the page settings to be modified.

PrinterSettings	Retrieves or sets the printer settings that are modified when the user clicks the Printer button in the dialog box.
ShowHelp	Retrieves or sets a value indicating whether or not the Help button is visible.
ShowNetwork	Retrieves or sets a value specifying whether or not the Network button is visible.

The PrintPreviewControl control

- PrintPreviewControl control displays a document to be printed- i.e., it displays a Review of the document to be printed. This control has no buttons or any other user interface elements.
- The PrintPreviewControl control is typically used only when there is a need to write custom print preview user interfaces. You can use PrintPreviewControl objects to create your own custom print previews.
- Following is the Properties of the PrintPreviewControl class

Property	Description
AutoZoom	Retrieves or sets a value that indicates whether or not the Zoom property is set automatically
Columns	Retrieves or sets the number of pages displayed horizontally
Document	Retrieves or sets a value indicating the document to be previewed
Rows	Retrieves or sets the number of pages displayed vertically
StartPage	Retrieves or sets the page number of the upper left page
UseAntiAlias	Retrieves or sets a value specifying whether or not printing uses the anti-aliasing features of the operating system
Zoom	Retrieves or sets the zoom level of the print preview

- Following is the Methods of the PrintPreviewControl class

Method	Description
InvalidatePreview()	Refreshes the preview of the document
ResetBackColor()	Resets the background color of the control to the color of the application workspace, which is the default color
ResetForeColor()	Resets the foreground color of the control to White, which is the default color

- Following is the Events of the PrintPreviewControl class

Event	Description
StartPageChanged	Occurs when the value of the StartPage property is changed
TextChanged	Occurs when the value of the Text property is changed

The PrintPreviewDialog control

- You can use the PrintPreviewDialog control that displays a preview of the document that is to be printed. The PrintPreviewDialog class supports the PrintPreviewDialog control.
- This class contains the buttons for printing, zooming in, displaying one or multiple pages, and closing the dialog box.

Property	Description

AcceptButton	Retrieves or sets the button that is automatically clicked when the user presses the ENTER key
ControlBox	Retrieves or sets a value specifying whether or not a control box is displayed in the title bar of the Windows Form
Document	Retrieves or sets the document to preview
FormBorderStyle	Retrieves or sets the border style of the form
HelpButton	Retrieves or sets a value specifying whether or not a Help button should be displayed in the control box of the form
MaximizeBox	Retrieves or sets a value specifying whether or not the Maximize button is displayed in the title bar of the form
MaximumSize	Retrieves or sets the maximum size the form
MinimizeBox	Retrieves or sets a value indicating whether or not the Minimize button is displayed in the caption bar of the form
MinimumSize	Retrieves the minimum size the form
PrintPreviewControl	Retrieves the data contained in this form
ShowlnTaskbar	Retrieves whether or not the form is displayed in the Windows taskbar
StartPosition	Retrieves or sets the starting position of the dialog box at run time
TopMost	Retrieves or sets a value specifying whether or not the form should be displayed as your application's topmost form

The PrintDialog Control

- The Print dialog box lets the user print documents, and this dialog box is supported by the PrintDialog class. The PrintDialog control shows the standard Print dialog box.
- This allows a user to select a printer and set its properties. If you do not show this dialog box, the output is sent automatically to the default printer and it uses the default settings of the printer.

Property	**Description**
AllowCurrentPage	Retrieves or sets a value specifying whether or not the CurrentPage option button is displayed. The Current Page option button prints the current page.
AllowPrintToFile	Retrieves or sets a value specifying whether or not the Print to file check box is enabled.
AllowSelection	Retrieves or sets a value specifying whether or not the Selection option button is enabled. The Selection option prints the selected text on a page.
AllowSomePages	Retrieves or sets a value specifying whether or not the Pages option button is enabled.
Document	Retrieves or sets a value specifying the PrintDocument control used to obtain PrinterSettings.
PrinterSettings	Retrieves or sets the printer settings that the dialog box modifies.
PrintToFile	Retrieves or sets a value specifying whether or not the Print to file check box is selected.
ShowHelp	Retrieves or sets a value specifying whether or not the Help button is displayed.

ShowNetwork	Retrieves or sets a value specifying whether or not the Network button is displayed.

Example of Printing Class

1. Take new C# application. In form1, take two Buttons, one RichTextBox and take controls from printing toolbox, as showing in Figure:

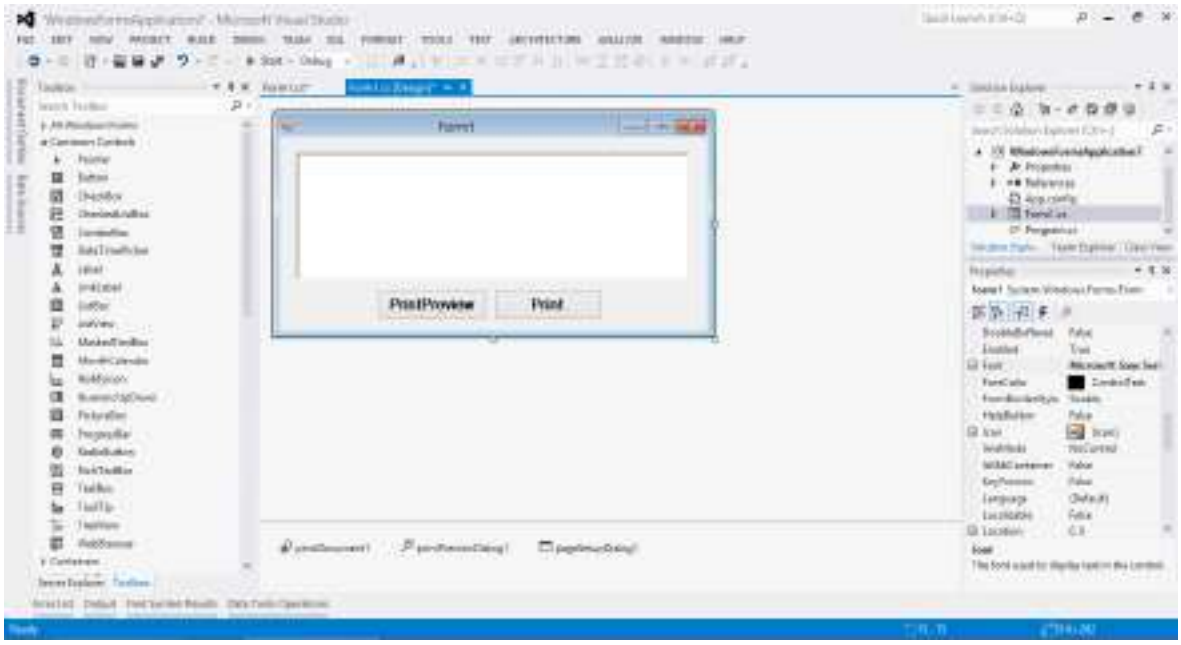


Figure: Showing design of Printing example

2. Add following code in both buttons' click event and PrintPage event of printDocument1 control.

```
namespace WindowsFormsApplication7
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
        }

        private void printDocument1_PrintPage(object sender,
System.Drawing.Printing.PrintPageEventArgs e)
        {
            e.Graphics.DrawString(richTextBox1.Text, new
Font("Arial",Convert.ToInt64(12)), Brushes.Blue, e.MarginBounds.Left  +10,
e.MarginBounds.Top+ 10);
        }

        private void btnPrintPreview_Click(object sender, EventArgs e)
        {
            printPreviewDialog1.Document = printDocument1;
            printPreviewDialog1.ShowDialog();
        }

        private void btPrint_Click(object sender, EventArgs e)
        {
            pageSetupDialog1.PrinterSettings = printDocument1.PrinterSettings;
            pageSetupDialog1.PageSettings = printDocument1.DefaultPageSettings;
            if (pageSetupDialog1.ShowDialog() ==
System.Windows.Forms.DialogResult.OK)
            {
                printDocument1.PrinterSettings =
pageSetupDialog1.PrinterSettings;
                printDocument1.DefaultPageSettings =
pageSetupDialog1.PageSettings;
            }
            printPreviewDialog1.Document = printDocument1;
            printPreviewDialog1.Show();
        }
    }
}
```

3. Press F5 to execute the program.

Figure: Showing the output when click on PrintPreview

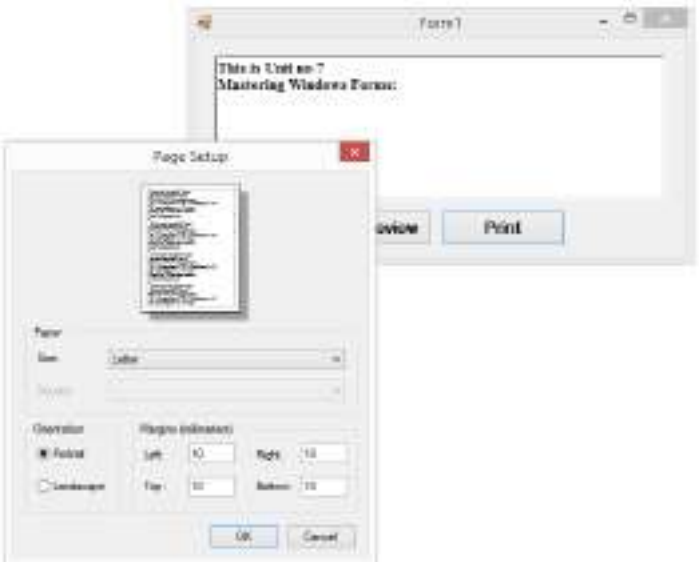


Figure: Showing the output when click on Print

Handling Multiple Events

- C#.NET is an event-driven language, which means the objects of C#.NET reacts when an event is triggered, for example, clicking a button and entering some text into a text box.
- An event is nothing but an action that calls a function or may cause another event, which does something as specified in the code. You can control the flow of an application by handling events.
- There are mainly two types of events: Mouse events (events generated by supplying the input through a mouse) and Keyboard events (events generated by supplying the input through the keyboard).

- You can handle mouse events-such as mouse movements in forms and controls. Following are the possible mouse events for the Control class, which is a base class for controls and forms:
  1. MouseDown: Occurs when the mouse pointer is over the control and a mouse button is pressed
  2. MouseEnter: Occurs when the mouse pointer enters the control
  3. MouseHover: Occurs when the mouse pointer moves to and from over the control
  4. MouseLeave: Occurs when the mouse pointer leaves the control
  5. MouseMove: Occurs when the mouse pointer moves over the control
  6. Mouseup: Occurs when the mouse pointer is over the control and a mouse button is released
  7. MouseWheel: Occurs when the mouse wheel moves, while the control has focus
- The following properties of the MouseEventArgs object are used to handle the mouse events:
  1. Buttons: Indicates which mouse button is pressed.
  2. Clicks: Indicates the number of times the mouse button is pressed and released.
  3. Delta: Indicates a calculated count of the number of detents the mouse wheel has rotated. A detent is the rotation of the mouse wheel by one notch.
  4. X: Indicates the x-coordinate of a mouse click.
  5. Y: Indicates the y-coordinate of a mouse click.
- The Buttons property holds one of these members of the MouseButtons enumeration:
  1. Left: Indicates that the left mouse button is pressed
  2. Middle: Indicates that the middle mouse button is pressed
  3. None: Indicates that no mouse button is pressed
  4. Right: Indicates that the right mouse button is pressed
  5. XButton1: Indicates that the first XButton is pressed
  6. XButton2: Indicates that the second XButton is pressed
- In C#.NET, the following events are available to handle the keyboard events:
  1. KeyDown: Occurs when a key is pressed down while the control has focus
  2. Keypress: Occurs when a key is pressed while the control has focus
  3. KeyUp: Occurs when a key is released while the control has focus
- For the Key Down and KeyUp events, the event handler receives an argument of type KeyEventArgs, containing data related to this event with these properties:
  1. Alt: Holds a value indicating whether the ALT key is pressed
  2. Control: Holds a value indicating whether the CTRL key is pressed
  3. Handled: Holds or sets a value indicating whether the event is handled
  4. KeyCode: Holds the keyboard code for a KeyDown or KeyUp event
  5. KeyData: Holds the keyboard data for a KeyDown or KeyUp event
  6. KeyValue: Holds the keyboard value for a KeyDown or KeyUp event
  7. Modifiers: Holds the modifier flags for a KeyDown or KeyUp event. This indicates which modifier keys (Ctrl, Shift, and/or Alt) are pressed.
  8. Shift: Holds a value indicating whether the shift key is pressed
- To handle the Keypress events, event handlers take an argument of type KeyPressEventArgs, which contains the following properties:
  1. Handled: Gets or sets a value indicating whether the Keypress event is handled. If you set this value to True, Visual Basic will not handle this key.

2. KeyChar: Holds the character corresponding to the key pressed

Example of Handling Multiple Events

1. Take new C# application, in form1 have one button with caption Add and add two textbox controls , as shown in Figure:

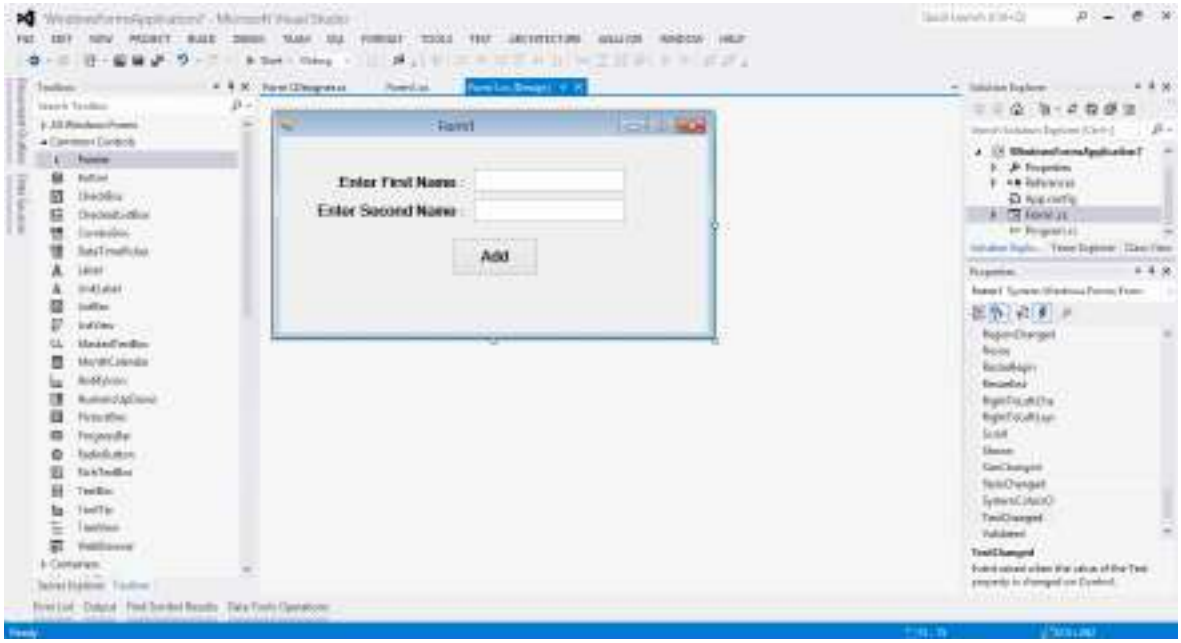


Figure: Showing the designing of example

2. Add following code in for the multiple events of textboxes and button.

```
namespace WindowsFormsApplication7
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
                private void textBox1_MouseEnter(object sender, EventArgs e)
        {
            textBox1.BackColor = Color.Red;
            textBox1.ForeColor = Color.White;
        }

        private void textBox1_MouseLeave(object sender, EventArgs e)
        {
            textBox1.BackColor = Color.White;
            textBox1.ForeColor = Color.Black;
        }
        private void textBox1_KeyPress(object sender, KeyPressEventArgs e)
        {
            if (!char.IsControl(e.KeyChar) && !char.IsLetter(e.KeyChar))
            {
                e.Handled = true;
            }
        }

        private void textBox2_MouseEnter(object sender, EventArgs e)
        {
            textBox2.BackColor = Color.Red;
            textBox2.ForeColor = Color.White;
        }

        private void textBox2_KeyPress(object sender, KeyPressEventArgs e)
        {
            if (!char.IsControl(e.KeyChar) && !char.IsLetter(e.KeyChar))
            {
                e.Handled = true;
            }
        }
        private void textBox2_MouseLeave(object sender, EventArgs e)
        {
            textBox2.BackColor = Color.White;
            textBox2.ForeColor = Color.Black;
        }

        private void button1_Click(object sender, EventArgs e)
        {
            MessageBox.Show("Your name is " + textBox1.Text + " " +
textBox2.Text);
        }
    }
}
```

3. Press F5 to execute the program, click on Button you will see the following output.

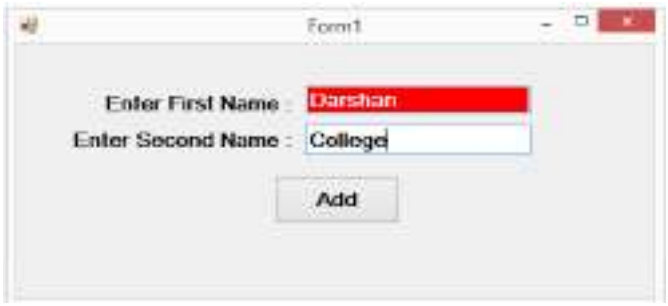


Figure: Showing the output when mouse enter in textBox1 control

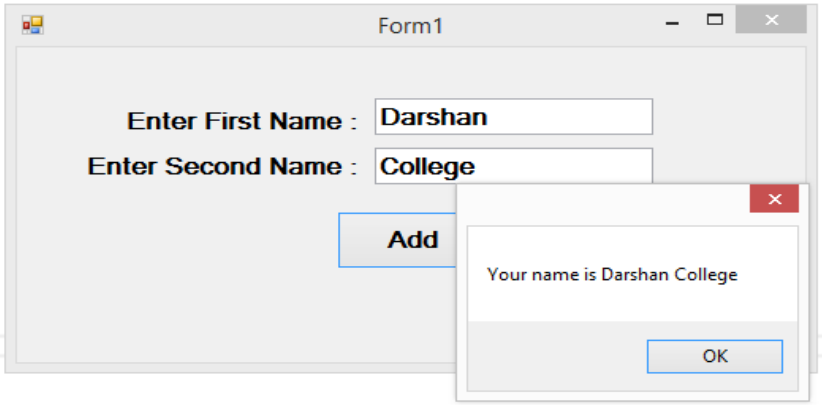


Figure: Showing the output when click on Add button

GDI+

- In Windows Forms applications, you might need to create simple shapes and other graphical objects. In such a case, you can use the graphics elements of Windows Forms which help to create a visually appealing application.
- In Windows Forms applications, GDI+ is used to create graphics, draw text, and manipulate graphical images. GDI is responsible for the integration of graphics in Windows Forms, and GDI+ is an advanced implementation of GDI.
- The basic use of GDI+ is to render graphical images on a form. The GDI+ managed class interface and the Basic GDI+ managed classes are used to display graphics on the Windows Forms.
- Many classes work together with the Graphics class to get the rendered effect. For example, the DrawLine() method receives a Pen object, which contains the attributes of the line to be drawn. The important GDI+ managed classes are the Graphics class, the Pen class, and the Brush class.

- Along with these classes, Windows Forms application also defines the color structure that helps you design graphics.

The Graphics Class

- The GDI+ drawing in Windows Forms is represented by the Graphics class, as it provides methods, such as the CreateGraphics() method, for drawing objects. It is contained in the System.
- You create the Graphics object from an image by using the FromImage() method.
- In addition, you can draw many different shapes and lines by using different methods of the Graphics object, such as DrawLine, DrawArc, DrawClosedCurve, DrawPolygon, and DrawRectangle.
- You can also draw images by using the DrawImage() and DrawIcon() methods of the Graphics class.
- Following is the table of Properties of the Graphics class:

Property	Description
Clip	Gets or sets the region that limits the drawing region of the Graphics object.
ClipBounds	Retrieves a Rectangle structure that bounds the clipping region of Graphics object.
CompositingMode	Retrieves a value that specifies how composited images are drawn to the Graphics object. A composited image refers to the combination of images from different image sources into a single image creating an illusion as if all these images are part of the single image.
DpiX	Retrieves the horizontal resolution of the Graphics object.
DpiY	Retrieves the vertical resolution of the Graphics object.
InterpolationMode	Gets or sets the interpolation mode associated with the Graphics object. The interpolation mode determines how the intermediate values between two endpoints are calculated.
IsClipEmpty	Retrieves a value indicating whether the clipping region of the Graphics object is empty.
IsVisibleClipEmpty	Retrieves a value indicating whether the visible clipping region of the Graphics object is empty.
PageScale	Gets or sets the scaling between the world units and page units for the Graphics object.
PageUnit	Gets or sets the unit of measure used for page coordinates in the Graphics object. The unit of measure is used for measuring the coordinates of a Graphics object in a Windows Forms application.
PixelOffsetMode	Gets or sets a value specifying how pixels are offset during rendering of the Graphics object.
RenderingOrigin	Gets or sets the rendering origin of the Graphics object for dithering and hatch brushes.
SmoothingMode	Gets or sets the rendering quality for the Graphics object.
TextContrast	Gets or sets the gamma correction value for rendering text. The gamma correction value is used for rendering the ClearType text. It must be between 0 and 12 and the default value is 4.
TextRenderingHint	Gets or sets the rendering mode for the text associated with the Graphics object.
Transform	Gets or sets a copy of the transformation for the Graphics object.

VisibleClipBounds	Retrieves the bounding rectangle for the area of clipping region of the Graphics object.

- Following is the table of Methods of the Graphics class:

Method	**Description**
BeginContainer()	Saves the current state of the Graphic object and opens and uses a new graphics container
Clear()	Clears the entire drawing surface and fills it with the specified background color
CopyFromScreen()	Copies the color data from the screen to the drawing surface of the Graphics object
Dispose()	Releases all the resources used by the Graphics object Draws an arc representing a portion of ellipse Draws a Bezier curve defined by four point structure Draws a series of Bezier curves
DrawArc()	Draws an arc representing a portin of ellipse
DrawBezier()	Draws a Bezier curve defined by four point structure
DrawBeziers()	Draws a series of Bezier curves
DrawClosedCurve()	Draws a closed cardinal curve
DrawCurve()	Draws a cardinal curve
DrawEllipse()	Draws an ellipse defined by a bounding rectangle
DrawIcon()	Draws an image represented by the specified icon
DrawIconUnstretched()	Draws an image without scaling the image
DrawImageUnscaled()	Draws an image using its original physical size at the location specified
DrawImageUnscaledAndClipped()	Draws the specified image without scaling and clips to fit the rectangle
DrawLine()	Draws a line connecting the two points specified by the coordinate pairs
DrawLines()	Draws a series of line segments that connect an array of points
DrawPath()	Draws a series of connected lines and curves
DrawPie()	Draws a pie shape defined by an ellipse
DrawPolygon()	Draws a polygon defined by an array of the Point structure
DrawRectangle()	Draws a rectangle specified by a coordinate pair, width and height
DrawRectangles()	Draws a series of rectangles defined by the Rectangle structure
DrawString()	Draws a text string at the specified location with the Brush and Font objects
EndContainer()	Closes the current graphics container and restores the state of the Graphics object
Equals()	Determines whether the specified object is equal to the current object
FillClosedCurve()	Fills the interior of the cardinal curve
FillEllipse()	Fills the interior of the ellipse
FillPath()	Fills the interior of the GraphicsPath class

FillPie()	Fills the interior of the pie section
FillPolygon()	Fills the interior of the polygon
FillRectangle()	Fills the interior of the rectangle
FillRectangles()	Fills the interior of the series of rectangles
FillRegion()	Fills the interior of the region of the graphic shape
Finalize()	Enables an object to free resources and perform cleanup operations before destroying the object
Flush()	Executes pending graphics operations forcefully
FromHdc()	Creates a new Graphics object from the specified point
FromHdcInternal()	Returns a Graphics object for a particular device context
FromHow()	Creates a Graphics object
FromHwnd()	Creates a new Graphics object from the specified handle to a window
FromHwndInternal()	Creates a new Graphics object for a windows handle
FromImage()	Creates a new Graphics object from an image
GetContextInfo()	Retrieves the cumulative graphics context
MeasureCharacterRanges()	Retrieves an array of region object
MeasureString()	Measures the specified string
MultiplyTransform()	Multiplies the transformation of the Graphics object
ResetTransform()	Resets the transformation matrix of the Graphics object to the identity matrix
Restore{)	Restores the state of the graphics to the state represented by the GraphicsState class
RotateTransform()	Applies the specified rotation to the transformation matrix of the Graphics object
Save()	Saves the current state of the Graphics object and identifies the saved state with the GraphicsState class
ScaleTransform()	Applies the specified scaling operation to the transformation matrix of the Graphics object
SetClip()	Sets the clipping region of the Graphics to the Clip property of the specified graphic
ToString()	Returns a String value that represents the Graphics class
TransformPoints()	Transforms an array of points from one coordinate space to another
TranslateClip()	Changes the clipping region of the Graphics object by the specified amounts in horizontal and vertical directions
TranslateTransform()	Changes the origin of the coordinate system

The Pen Class

- The Pen class defines an object to draw lines and curves in Windows Forms. Pens are represented by the System.Drawing.Pen class contained in the System.Drawing namespace.
- The Pen class draws the line of specified width and style. You need to specify how thick the line should be and how to fill the area inside a wide line.

- For example, you can use the DashStyle property to draw several kinds of dashed lines. You can fill the area drawn by the Pen class with variety of fill styles, for instance using a solid color or a texture.
- Following is the properties of the Pen class:

Property	Description
Alignment	Obtains or specifies the alignment of the Pen class
Brush	Obtains or specifies the Brush class that determines attributes of the Pen class
Color	Obtains or specifies the color of the Pen class
CompoundArray	Obtains or specifies an array of real numbers that specifies the compound Pen class
CustomEndCap	Obtains or specifies a custom cap that is to be used at the end of the lines drawn with the Pen class
CustomStartCap	Obtains or specifies a custom cap that is to be used in the beginning of the lines drawn with the Pen class
DashCap	Obtains or specifies the cap style that is to be used at the end of each dash in the dashed lines drawn with the Pen class
DashOffset	Obtains or specifies the distance between the dashes in the dashed lines drawn with the Pen class
DashPattern	Obtains or specifies an array of custom dashes and spaces
DashStyle	Obtains or specifies the style used for dashed lines
EndCap	Obtains or specifies the cap style that is to be used at the end of the lines
LineJoin	Obtains or specifies a style to join the two consecutive lines that are drawn with the Pen class
MiterLimit	Obtains or specifies the thickness of the joined corner
PenType	Retrieves the style of lines drawn with the Pen class
StartCap	Obtains or specifies the cap style that is to be used at the beginning of lines
Transform	Obtains or specifies a copy of the geometric transformation for the Pen class
Width	Obtains or specifies the width of the Pen class

- Following is the methods of the Pen class

Method	Description
Clone()	Creates a copy of the Pen class
CreateObjRef()	Creates an object containing the information to generate a proxy for the communication with a remote object
Dispose()	Releases all the resources used by the Pen class
Equals()	Determines whether or not a particular object is equal to the current object
Finalize()	Enables an object to free resources and perform cleanup operations before destroying the object
GetHashCode()	Indicates a hash function for a specified type
GetLifetimeService()	Retrieves the current lifetime service object that controls the lifetime for the Pen class
GetType()	Retrieves the Type object of the Pen class
InitializeLifetimeService()	Obtains a lifetime service object to control the lifetime of the Pen class
ResetTransform()	Sets the world transformation matrix of the current class to the identity matrix

SetLineCap()	Sets the values of the SetLineCap() method that determines the style of cap used to end lines drawn by the Pen class
ToString()	Returns a String object that represents the Pen class

The Brush Class

- The Brush class is used to fill the interiors of graphical shapes, such as rectangles, ellipse, polygons, and paths. You can draw the outline of the shape using the Pen class and fill the shape using the Brush class.
- The different types of brushes available in Windows Forms are SolidBrush, HatchBrush, TextureBrush, LinearGradientBrush, and PathGradientBrush.
- Following is the properties of the Brush class:

Property	Description
SolidBrush	Fills the shape with one solid color
HatchBrush	Fills the shape with a hatch style
TextureBrush	Fills the shape with a pattern stored in a bitmap
LinearGradientBrush	Fills the shape with color that changes gradually as you move across the shape
PathGradientBrush	Fills the shape with color which changes as you move from the center of a shape towards the edge

- Following is the methods of the Brush class

Method	Description
Clone()	Creates an exact copy of the Brush object
CreateObjRef()	Creates an object that contains all the relevant information required to generate a proxy used to communicate with a remote object
Equals()	Determines whether the specified object is equal to the current object
GetHashCode()	Serves as a hash function for a particular type
GetLifetimeService()	Retrieves the current lifetime service object, which is used to control the current lifetime policy
GetType()	Retrieves the Type object of the current instance
InitializeLifetimeService()	Obtains a lifetime service object to control the lifetime policy for this instance
ToString()	Returns a string that represents the current object

Example of GD+

1. Take new C# application, in form1 have one button with caption Click Here and add one PictureBox controls , as shown in Figure:

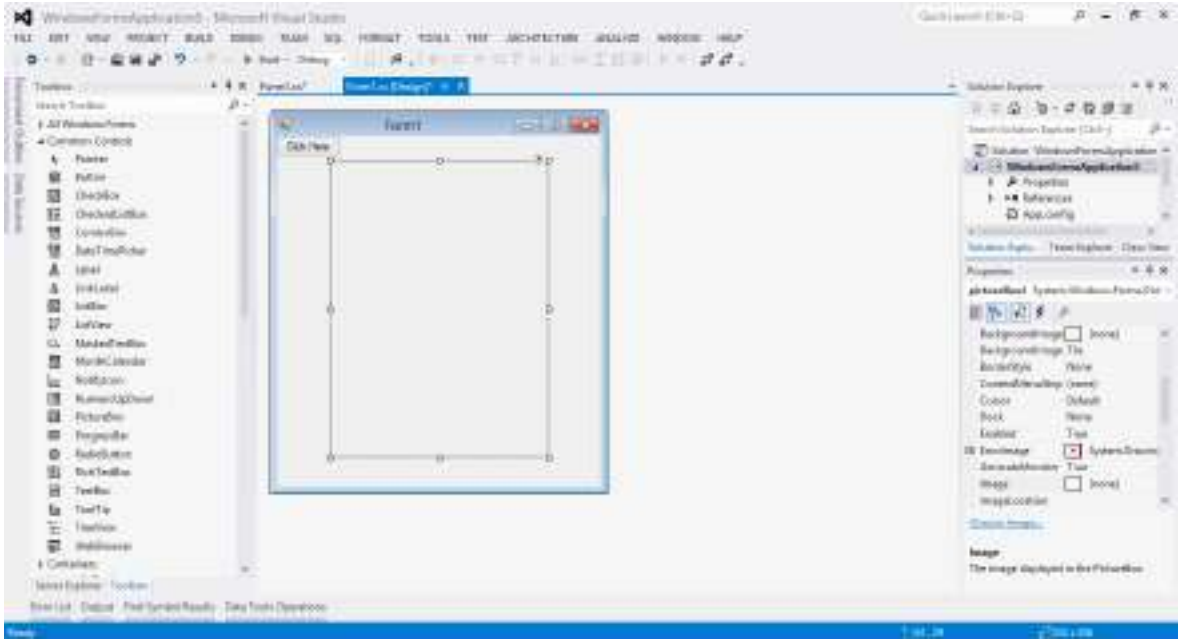


Figure: Showing to add one Button and PictureBox

2. Add following code in button click event.

```
private void button1_Click(object sender, EventArgs e)
{
    Graphics g = pictureBox1.CreateGraphics();
    SolidBrush myBrush = new SolidBrush(Color.Teal);
    g.FillEllipse(myBrush, 0, 0, 50, 50);
    myBrush.Color = Color.Black;
    g.FillEllipse(myBrush, 10, 15, 10, 10);
    g.FillEllipse(myBrush, 30, 15, 10, 10);
    //g.FillPolygon(myBrush, new Point[] { new Point(25, 30), new Point(15, 45),
new Point(35, 45) });
    g.FillPie(myBrush, 0, 15, 50, 30, 45, 90);
    myBrush.Color = Color.Red;
    g.FillRectangle(myBrush, 0, 55, 50, 100);
    g.DrawLine(new Pen(myBrush, 5), 25, 155, 0, 230);
    g.DrawLine(new Pen(myBrush, 5), 25, 155, 50, 230);
}
```

3. Press F5 to execute the program, click on Button you will see the following output.

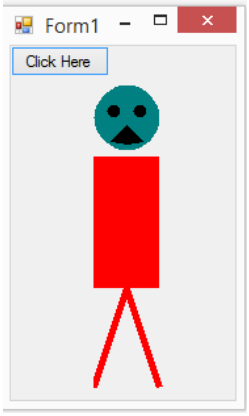


Figure: Showing the output

Creating Windows Forms Controls

- There are situations, when you need some specific functionality in a control, which is not provided by the built-in C#.NET controls. In such situations, you can create user controls and custom controls according to the requirement.
- User controls are basically containers into which you can put markup and C#.NET controls. Later on you can use the user control as a unit in various Windows Forms. Whereas, custom control is a class, which you write, and it derives from Control classes.
- Creating user controls is comparatively easier task to create than custom controls, because user controls are similar to a Windows Form, with both a user interface page and code, while custom control does not have any user interface.
- Let's create an application, named CustomControl to learn how to create a custom control. Now, perform the steps to create a custom control:
  1. Add a Custom Control in the CustomControl application, as CustomControl1.cs and add the following code.

```
public partial class CustomControl1 : Control
{
    public CustomControl1()
    {
        InitializeComponent();
    }

    protected override void OnPaint(PaintEventArgs pe)
    {
        base.OnPaint(pe);
        pe.Graphics.FillEllipse  (Brushes.Blue , 10, 10, 100, 50);
    }
}
```

  2. Build the application. This adds the custom control in the Toolbox, as shown in Figure:

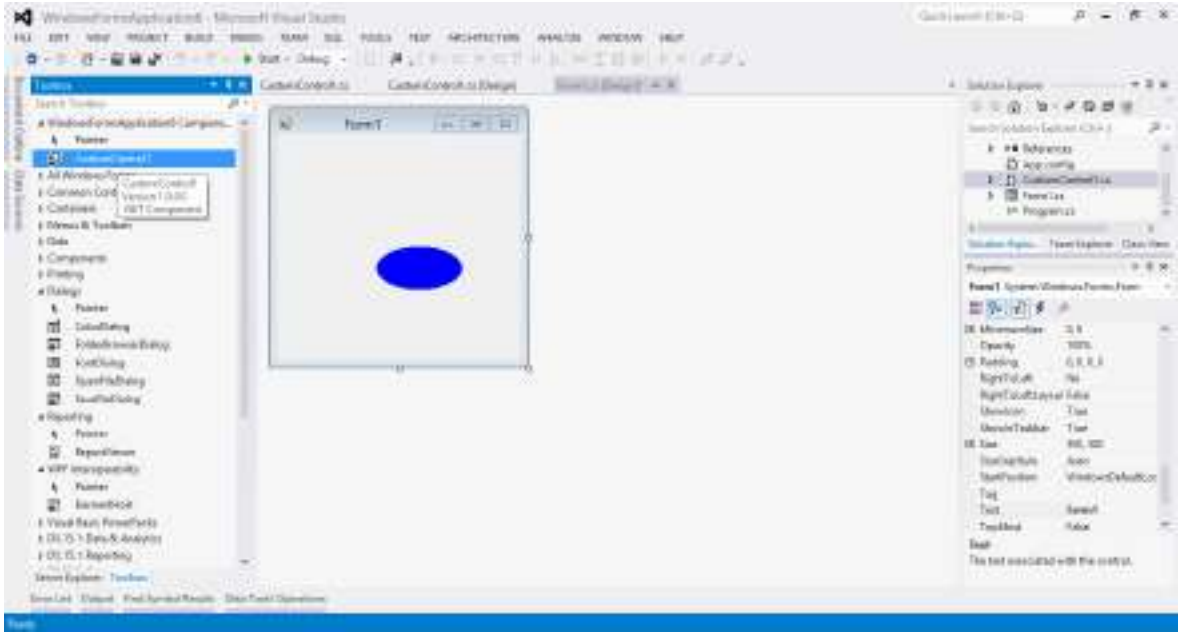


Figure: Showing the Custom Control in Toolbox

3. Drag and drop the CustomControl1 control on the Form1 form.
4. Double click the CustomControl1 control and add the code, shown in following code, in the click event of the CustomControl1 control:

```
private void customControl11_Click(object sender, EventArgs e)
{
    MessageBox.Show("Hello");
}
```

5. Press the F5 key to run the application and click the CustomControl1 control. A message box appears, as shown in Figure:

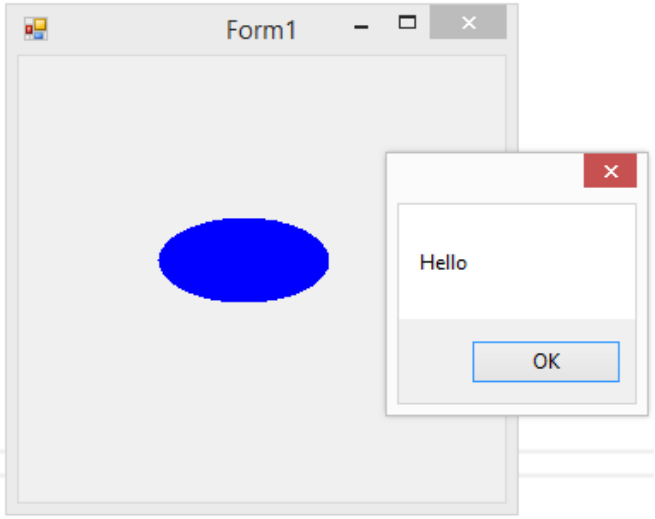


Figure: Displaying themessagebox from Custom Control

Introduction to ASP.Net:

- ASP.Net Refers as Active Server Page .net which is server side object oriented programming language.
- ASP.Net is language of Microsoft.net framework to develop web applications.
- Benefits using Asp.net:

Easier OOP Language	Quick Drag & Drop control	Code Separation
Easier database operations	Version Compatibility	High security
Easy app. Development	Highly Integrated IDE	Etc

- Asp.net uses to develop dynamic web pages, websites, web services and web applications.
- Asp.net is built on the CLR (Common Language Runtime) that allows user to write Asp.net program using any .net framework supported languages like C#, VB, Visual C++, J# etc.
- Asp.net also Supports ADO.Net (ActiveX Data Objects.Net) that helps to connect and work with data stored in database.
- Asp.net also provides Web services through which Same Service can be applicable to more than one website. E.g. weather, cricket live score etc.
- Asp.net provides everything Readymade such as Rich controls, Validation Controls, data bound controls, AJAX controls for developing easier, fast and highly dynamic web applications.

Directory Structure of ASP.Net website:

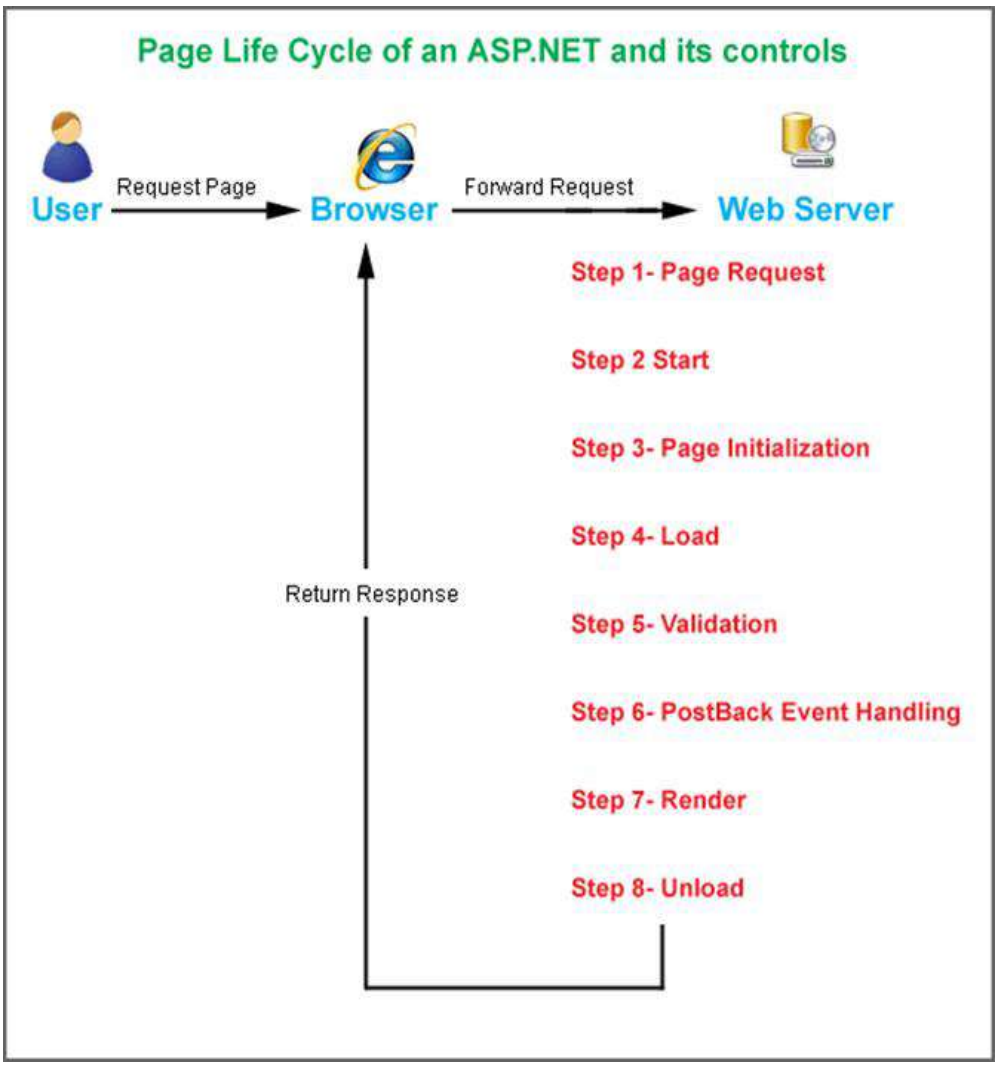
Bin	App_Code	App_GlobalResources	App_LocalResources
App_Data	App_Themes	App_Browsers	App_Webreferences

Differentiate ASP and ASP.Net

	ASP.Net	**ASP**
Language	OOP language	Scripting Language
Inheritance	Supports	Doesn't Support
Code behind files	Separate file for code and data both	Single file for code and data
Configuration files	Machine.config, Web.Config	No Configuration files
Custom Controls	Supports	Doesn't Support
Database Language	ADO.Net with XML integration	Simple ADO with limited functionalities

Explain ASP.Net Webpage life cycle & Events

When a user request for ASP.Net page, the page gone through many stages that is call its life cycle.



Various stages in ASP.Net Web Page

Stages	What is done in Stages?
Page Request	Asp.Net checks that request of the page is new or old, if new then it compiles and executes that page and if request is old, cached copy will be returned.
Start	Request And Response properties are set and using IsPostBack property new or old request can be identified.
Page Initialization	In page, each control is initialized. Themes will be loaded. If requested page is old then post back data is not loaded.
Load	If current request is old one, the property of control is loaded with data from view and control state.
Postback Event Handling	If request for the page is old one, then any event handler can be invoked.
Validation	After loading the page, the validation controls are used to invoke Validate method for error free access.
Rendering	All controls on the page are saved. During rendering, render method for each control on page and writes it to output stream object of page response property.
Unload	Request and response properties of the page are unloaded and cleaning performed if required.

Asp.Net Pages Life Cycle Events

Events	Working Of Events
PreInit	First event of Asp.net Page Life cycle. It checks whether the page is processed for the first time or not. It also creates and recreates dynamic controls, sets master page and themes. It sets and gets profile property.
Init	This event is raised after initialization of all controls and applying skin properties and initialize or read control properties.
Load	This event is called to set control properties and establish database connections. Page class calls OnLoad event of web page after that calls event for each child control, until page is loaded.

Explain various Web Server Controls of ASP.Net

- Web server controls are special ASP.NET tags understood by the server.
- Like HTML server controls, Web server controls are also created on the server and they require a runat="server" attribute to work.
- However, Web server controls do not necessarily map to any existing HTML elements and they may represent more complex elements.
- Mostly all Web Server controls inherit from a common base class, namely the WebControl class defined in the System.Web.UI.WebControls namespace.
- The syntax for creating a Web server control is:
  <asp:control_name id="some_id" runat="server" />

Web Server Controls:

Text Box	Check Box	Radio Button	Label
Button	Hyper Link	Image	Etc.

Common Properties for all controls:

ID	Runat	Text	CssClass
Height	Width	Auto post back	Visible

- **Text Box:**
  - The TextBox control is used to create a text box where the user can input text.

```
<asp:TextBox ID="TextBox1" runat="server" AutoPostBack="true" height="10px" width="10px" visible="true"></asp:TextBox>
```

  - Textbox has one property named Textmode with following values
    - Textmode: SingleLine – this creates a basic TextBox
    - Textmode: MultiLine – this a multiline TextBox
    - Textmode: Password – this creates a password TextBox
- **Button:**
  - The Button control is used to display a push button. The push button may be a submit button or a command button. By default, this control is a submit button.
  - A submit button does not have a command name and it posts the page back to the server when it is clicked. It is possible to write an event handler to control the actions performed when the submit button is clicked.

```
<asp:Button ID="Buttton1" runat="server" Text="Click Me" OnClick="Button1_Click" />
```

- **Check Box:**
  - Checkbox allow web site visitor to select (checked) a true or false condition. So the checkbox control creates a checkbox on web forms page which allow user to switch between a true or false state.
  - Checkbox Text property creates a caption for it. We can left or right align caption by using checkbox **TextAlign** property.
  - If we want to determine whether checkbox is checked (selected) then we need to test the checkbox Checked property.
  - Checkbox **AutoPostBack** property value true enable automatic posting to server.
  - Checkbox **CheckedChanged** event raised when someone change the checkbox true or false state

```
<asp:CheckBox ID="CheckBox1" runat="server" Text="Are you an asp.net user group member?" OnCheckedChanged="CheckBox1_CheckChanged" AutoPostBack="true" Font-Names="Serif" Font-Size="X-Large" />
```

- **Radio Button:**
  - Radiobutton control lets you make a group of radiobuttons with other radiobutton.
  - If you set it's **GroupName** property same for multiple radiobuttons then all radiobuttons with same name act as a single group. Within a group you can only select one radiobutton at a time.
  - Radiobutton have AutoPostBack property and CheckChanged event, by using those feature you can determine immediately which radiobutton is selected from a group.
  - This control is used to select Single value from a list.

```
<asp:RadioButton ID="RadioButton1" runat="server" AutoPostBack="True" Visible="True" Group="one"></asp:RadioButton>

<asp:RadioButton ID="RadioButton1" runat="server" AutoPostBack="True" Visible="True" Group="one"></asp:RadioButton>
```

Male ◯ **Female** ◯

- **Label:**
  - This Control is used to display information on the web Page.

```
<asp:Label ID="Label1" runat="server" Visible="True" Text="Hi This Is GTU !!!"> <asp:Label>
```

  - Example:
    Hi this Is GTU !!!
- **HyperLink:**
  - This control provides easy navigation between various Pages.
  - We can display a hyperlink on a web form page by inserting a HyperLink server control on the page.
  - Hyperlink sever control can render as either text or as graphics.
  - **NavigateUrl** property value stores the link destination.
  - To display hyperlink control as a text we need to set the **Text** property value and to display as an image set the **ImageUrl** property value. if we set both ImageUrl and Text property values, ImageUrl takes precedence
  - Hyperlink server control's another important property is the **Target** property. .net developers can specify the frame or window to display the linked page by this Target property settings.

```
<asp:HyperLink ID="HyperLink1" runat="Server" NavigateUrl="Home.aspx" Text="Home Page"></asp:HyperLink>
```

- **Image:**
  - This Control is used to display images on the web page.
  - ImageUrl = Location of the image in application folder or in computer.
  - AlternateText= alternate text if image is not found

```
<asp:Image ID="Img" runat="server" ImageUrl="~/images/ab.jpg" AlternateText="Jerry"/>
```

- Example:

Explain various Validation Controls of ASP.Net

- Asp.Net provides various validation controls for error free access for the web page. This validation Controls validates the input for controls. So that any mismatching and faulty data cannot be inserted to the data storage.
- Various Validation controls are:
  - Required Field Validator
  - Range Validator
  - Regular Expression Validator
  - Compare Validator
  - Custom Validator
  - Validation Summary

Common Properties of validation controls are

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

Required Field Validator

- 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.
- This Validator is applied to such controls where input is necessary.

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

Range Validator

- 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.

- E.g. Salary must between 10000 and 50000.

```
<asp:TextBox ID="TextBox1" runat="server" AutoPostBack="true" height="10px" width="10px"
visible="true"></asp:TextBox >

<asp:RangeValidator ID="rv1" runat="server" ControlToValidate="TextBox1"
ErrorMessage="Salary between 10000 to 50000" MinValue="10000" MaxValue="50000">
</asp:RangeValidator >
```

Regular Expression Validator

- 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.
- Syntax

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

Compare Validator:

- 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.

- Syntax

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

Custom validator:

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

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

Validation Summary:

- Validation summary provides all the validation errors as a list at one time at the end.
- 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.

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

Validation Groups

- Complex pages have different groups of information provided in different panels. In such situation, a need might arise for performing validation separately for separate group. This kind of situation is handled using validation groups.
- To create a validation group, you should put the input controls and the validation controls into the same logical group by setting their **ValidationGroup** property.

Explain various List Controls in ASP.Net

ASP.NET provides the following controls

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

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

List box and Drop-down List

- Basic syntax of list box control:

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

- Basic syntax of drop-down list control:

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

- Common properties of list box and drop-down Lists:

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

- Common properties of each list item objects:

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

- To work with the items in a drop-down list or list box, you use the Items property of the control. This property returns a ListItemCollection object which contains all the items of the list.

- The SelectedIndexChanged event is raised when the user selects a different item from a drop-down list or list box.

The ListItemCollection

- The ListItemCollection object is a collection of ListItem objects. Each ListItem object represents one item in the list. Items in a ListItemCollection are numbered from 0.
- When the items into a list box are loaded using strings like: lstcolor.Items.Add("Blue"), then both the Text and Value properties of the list item are set to the string value you specify. To set it differently you must create a list item object and then add that item to the collection.
- The ListItemCollection Editor is used to add item to a drop-down list or list box. This is used to create a static list of items. To display the collection editor, select edit item from the smart tag menu, or select the control and then click the ellipsis button from the Item property in the properties window.
- Common properties of ListItemCollection

Property	Description
Item(integer)	A ListItem object that represents the item at the specified index.
Count	The number of items in the collection.

- Common methods of ListItemCollection:

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

Radio Button list and Check Box list

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

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

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

- Common properties of check box and radio button lists:

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

Bulleted lists and Numbered lists

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

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

- Common properties of the bulleted list:

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

Explain various Data Controls of ASP.Net

- Data Bound controls are used to display data from the DataBase.
- The results can be customized according to the query style and user preferences.
- Custom edit, paging and select record type functionalities are ready made available.
- Just drag and drop data controls and display the records according to your need.
- Some Controls are mentioned and explained below:

GridView

- The GridView control is used to display the data in tabular form. In this form, each column represents a field and each row represents a record.
- With GridView control, a developer can display an entire collection of data and add sorting or paging option in the table.
- This control also allows editing and deleting data from database.
- The GridView control exists within the System.Web.UI.WebControls namespace.
- Column Field Type of GridView Class:
    - **BoundField:** This column displays the value of a field in the data source. It is the default column type.
    - **ButtonField :** This column displays a button for each item in the list.
    - **CheckBoxField :** This column shows a check box for each item in the list.
    - **CommandField :** This column perform selection, editing or deleting operation.
    - **HyperLinkField :** It shows a value of data source content as a hyperlink.
    - **ImageField :** It displays an image for each item.
    - **TemplateField :** It is used to create a customized column field.
- **Example Code**

```
<asp:GridView ID="GridView1" runat="server" AllowPaging="True" AllowSorting="True"
AutoGenerateColumns="False" DataKeyNames="Id" DataSourceID="SqlDataSource1">
<Columns>
        <asp:BoundField DataField="Id" HeaderText="Id" InsertVisible="False" ReadOnly="True"
SortExpression="Id" />
        <asp:BoundField DataField="Name" HeaderText="Name" SortExpression="Name" />
        <asp:BoundField DataField="College" HeaderText="College"  SortExpression="College" />
</Columns>
</asp:GridView>
```

The RepeaterControl

- The Repeater control is a data bound control that is used to display data by using customized layouts.
- This control does not support in-built layout or style.
- This control does not have a default layout, so you need to declare all layout or styles to use this control.
- The RepeaterControl is very helpful to display data in the form of table, grid and bulleted list or other format.
- The RepeaterControl exists within the System.Web.UI.WebControls namespace.
- Some important properties of ReperatorControl class are given below:
    - **AlternatingItemTemplate**: It defines how alternating items in the control are displayed.
    - **DataMember**: Sets a table in database to bind to the repeater control.
    - **DataSource**: Sets the data source that provides data for populating the list.
    - **DataSourceID**: Sets the ID of the Data source control that is used by the repeater to retrieve its data source.
- Example Code

```
<asp:Repeater ID="Repeater1" runat="server" DataSourceID="SqlDataSource2">
<HeaderTemplate>
        <table border=1 width="150px">
                <tr><td>Id</td>
                <td>Name</td>
                <td>College</td></tr>
        </table>
</HeaderTemplate>
<ItemTemplate>
<table border=1 width="150px">
        <tr><td><asp:Label ID="id" Text='<%#Eval("Id") %>' runat="server">
</asp:Label></td>
        <td><asp:Label ID="Name" Text='<%#Eval("Name") %>' runat="server">
</asp:Label></td>
        <td><asp:Label ID="College" Text='<%#Eval("College") %>' runat="server">
</asp:Label></td></tr></table></ItemTemplate>
</asp:Repeater>

<asp:SqlDataSource ID="SqlDataSource2" runat="server"  ConnectionString="<%$
ConnectionStrings:ConnectionString %>"  SelectCommand="SELECT * FROM
[Student]"></asp:SqlDataSource>
```

DataList Control

- The DataList control is a control that is used to display the data rows in a changeable format by using templates.
- The DataList control exists within the System.Web.UI.WebControls namespace.
- There are several templates that are supported by DataList control are given below:
  - **AlternatingItemTemplate**: It provides the format and content for alternating cell.
  - **EditItemTemplate**: It provides the format and content for the item to be edited.
  - **FooterTemplate** : This template is used to describe the footer message which a developer wants to display in the footer row.
  - **HeaderTemplate**: This template is used to describe the header message which a developer wants to display in the header row.
  - **ItemTemplate** : It provides the format and content for items in DataList Control.
  - **SelectedItemTemplate**: It is used to set the format of selected items in DataList Control.
  - **SeparatorTemplate**: It is used to separate data rows from each other with a customizable format and content.

```
<asp:DataList ID="DataList1" runat="server" DataKeyField="Id" DataSourceID="SqlDataSource3"
RepeatDirection="Horizontal">
        <ItemTemplate>
                Id:<asp:Label ID="Id" runat="server" Text='<%# Eval("Id") %>' /><br />
                Name:<asp:Label ID="Name" runat="server" Text='<%# Eval("Name") %>' />
                College:<asp:Label ID="College" runat="server" Text='<%# Eval("College") %>' />
        </ItemTemplate>
</asp:DataList>
<asp:SqlDataSource ID="SqlDataSource3" runat="server" ConnectionString="<%$
ConnectionStrings:ConnectionString %>"
SelectCommand="SELECT * FROM [Student]">
</asp:SqlDataSource>
```

ListView

- The ListView data bound control is used to display data in any format using templates and styles.
- This control enables a developer to edit, insert, delete and sort the page data.
- The ListView control provides different types of templates and styles according to user requirement.

- By default, the ListView control has no layout. So to use ListView control, you need to configure ListView by selecting a template and style as a screenshot is given below.

The DetailsView

- The DetailsView data bound control is a control that is used to display a single record at a time.
- It places each piece of information in a separate row of a table.
- DetailsView control also supports insert, update and delete operations.
- The DetailsView control exists within the System.Web.UI.WebControls namespace.

	au_id	au_lname	au_fname	state
Select	712-45-1855	test	test	AL
Select	712-45-1867	123123	Controller	AL

Author Details	
au_id	712-45-1855
au_lname	test
au_fname	test
phone	0898558555
address	154
city	454
state	AL
zip	12100
contract	☐
Edit	

The FormView

- Similar to DetailsView control, the FormView control is a control that is used to display a single record at a time.
- Each row of the table displays each field of the record.
- A developer can modify the layout of FormView for displaying the record.
- The FormView control exists within the System.Web.UI.WebControls namespace.
- There are several templates that are supported by FormView control are given below:
    - **EditItemTemplate**: It contains input controls and command buttons to edit the existing record.
    - **ItemTemplate** : It puts the FormView into edit mode.
    - **InsertItemTemplate**: This template is used to add a new record.
    - **FooterTemplate** : This template is used to describe the footer message which a developer wants to display in the footer row.
    - **HeaderTemplate**: This template is used to describe the header message which a developer wants to display in the header row.
    - **EmptyDataTemplate**: This template is used to display the message for the data row that does not contain any record.
    - **PagerTemplate**: This template is used to describe the content for pager row.

Explain various Rich server Controls in ASP.Net

- Rich server controls are used to deploy rich applications in very easy manner.
- Such as to rotate advertisements in webpage.
- Rich customized online calendar control.

Ad rotator

- Displays different ad images and, when clicked, will navigate to the URL associated with that image.
- You can define the rotation schedule in an XML file.
- Ad rotator controls are used to rotate advertisements on web screen using XML.
- XML file make the control more effective.

```
<asp:AdRotator
        id="Value"
        AdvertisementFile="AdvertisementFile"
        KeyWordFilter="KeyWord"
        Target="Target"
        OnAdCreated="OnAdCreatedMethod"
        runat="server"/>
```

- **Advertisement File Format**
- The AdRotator control uses a separate XML advertisement file to store the advertisement information, such as the location of the image to display and the URL of the page to link to. The AdvertisementFile property of the AdRotator control specifies the path to this file.

Element	Description
<ImageUrl>	The absolute or relative URL to an image file (optional).
<NavigateUrl>	The URL of a page to link to if the user clicks the ad (optional). Note If this element is not set, the HRef property is not rendered on the anchor tag.
<AlternateText>	The text display in place of the image when the image specified by the ImageUrl property is not available (optional). In some browsers, this text also appears as a ToolTip for the advertisement.
<Keyword>	A category for the advertisement (for example, "computers") that you can filter by (optional).
<Impressions>	A number that indicates the importance of the ad in the schedule of rotation relative to the other ads in the file (optional). The larger the number, the more often the ad is displayed. The total of all Impressions values in the XML file cannot exceed 2,047,999,999. If it does, the AdRotator throws a run-time exception.

- The following example demonstrates how to format an XML advertisement file.

```
<Advertisements>
        <Ad>
                <ImageUrl>image1.jpg</ImageUrl>
                <NavigateUrl>http://www.microsoft.com</NavigateUrl>
                <AlternateText>Microsoft Main Site</AlternateText>
                <Impressions>80</Impressions>
                <Keyword>Topic1</Keyword>
                <Caption>This is the caption for Ad#1</Caption>
        </Ad>
        <Ad>
                <ImageUrl>image2.jpg</ImageUrl>
                <NavigateUrl>http://www.wingtiptoys.com</NavigateUrl>
                <AlternateText>Wing Tip Toys</AlternateText>
                <Impressions>80</Impressions>
                <Keyword>Topic2</Keyword>
                <Caption>This is the caption for Ad#2</Caption>
        </Ad>
</Advertisements>
```

Calendar Control

- Calendar controls are used to maintain online calendar.
- It displays a one-month calendar that allows the user to select dates and move to the next or previous month.
- By setting the SelectionMode property, you can specify whether the user can select a single day, a week, or a month, or you can disable date selection entirely.
- Customized calendar is developed with reminders, birthdays etc.

```
<asp:Calendar id="Calendar1"
        CellPadding="pixels"
        CellSpacing="pixels"
        DayNameFormat="FirstLetter|FirstTwoLetters|Full|Short"
        FirstDayOfWeek="Default|Monday|Tuesday|Wednesday|
                        Thursday|Friday|Saturday|Sunday"
        NextMonthText="HTML text"
        NextPrevFormat="ShortMonth|FullMonth|CustomText"
        PrevMonthText="HTML text"
        SelectedDate="date"
        SelectionMode="None|Day|DayWeek|DayWeekMonth"
        SelectMonthText="HTML text"
        SelectWeekText="HTML text"
        ShowDayHeader="True|False"
        ShowGridLines="True|False"
        ShowNextPrevMonth="True|False"
        ShowTitle="True|False"
        TitleFormat="Month|MonthYear"
        TodaysDate="date"
        VisibleDate="date"
        OnDayRender="OnDayRenderMethod"
        OnSelectionChanged="OnSelectionChangedMethod"
        OnVisibleMonthChanged="OnVisibleMonthChangedMethod"
        runat="server"></asp:Calendar>
```

Your Birthday:

≤	October 2013						≥
>>	Sun	Mon	Tue	Wed	Thu	Fri	Sat
≥	29	30	1	2	3	4	5
≥	6	7	8	9	10	11	12
≥	13	14	15	16	17	18	19
≥	20	21	22	23	24	25	26
≥	27	28	29	30	31	1	2
≥	3	4	5	6	7	8	9

Todays date is: 11/15/2013

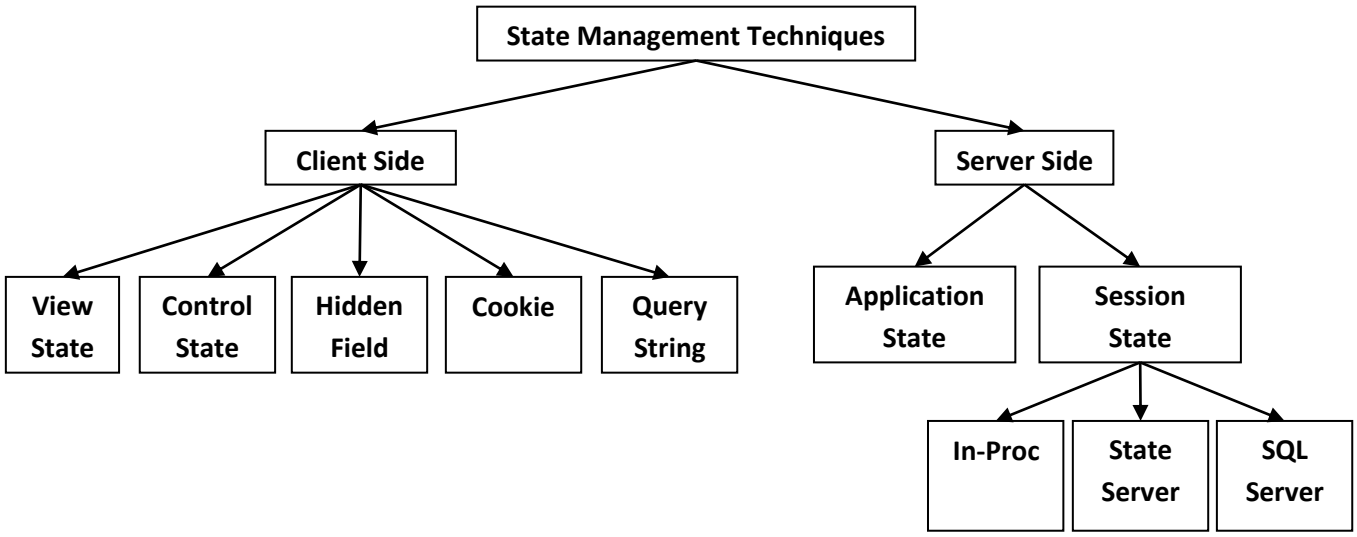
Your Birthday is: 10/22/2013

Explain various State management policies in ASP.Net

State Overview

- Whenever we visit a website, our browser communicates with the respective server using the HTTP or HTTPs protocol.
- As HTTP is a stateless protocol. It just cleans up or we can say removes all the resources/references that were serving a specific request in the past.
- These resources can be:
  - Objects
  - Allocated Memory
  - Sessions ID's
  - Some URL info
  - and so on.
- In traditional Web programming, this would typically mean that all information associated with the page and the controls on the page would be lost with each round trip.
- To overcome this inherent limitation of traditional Web programming, ASP.NET includes several options that help you preserve data on both a per-page basis and an application-wide basis.
- The state of a web application helps you to store the runtime changes that have been made to the web application.
- For example, a user selects and saves some products in the shopping cart of an online shopping websites.
- For that you have to store it to state, otherwise the changes are discarded.

State Management Types



View State (Page Level State)

- The ViewState property provides a dictionary object for retaining values between multiple requests for the same page.
- This is the default method that the page uses to preserve page and control property values between round trips.
- The page framework uses view state to persist control settings between postbacks.
- You can use view to do the following:
  - Keep values between postbacks without storing them in session state or in a user profile.
  - Store the values of page or control properties that you define.
  - Create a custom view state provider that lets you store view state information in a SQL Server database or in another data store.
- Some of the features of view state are:
  - It is page-level State Management
  - Used for holding data temporarily
  - Can store any type of data
  - Property dependent

```
// Page Load Event
protected void Page_Load(object sender, EventArgs e) {
        if (IsPostBack)
        {
                if (ViewState["count"] != null) {
                        int ViewstateVal = Convert.ToInt32(ViewState["count"]) + 1;
                        View.Text = ViewstateVal.ToString();
                        ViewState["count"]=ViewstateVal.ToString();
                }
                else {
                        ViewState["count"] = "1";
                }
        }
}

 // Click Event
protected void Submit(object sender, EventArgs e) {
        View.Text=ViewState["count"].ToString();
}
```

Control State

- Sometimes you need to store control-state data in order for a control to work properly. For example, if you have written a custom control that has different tabs that show different information, in order for that control to work as expected, the control needs to know which tab is selected between round trips.
- The ViewState property can be used for this purpose, but view state can be turned off at a page level by developers, effectively breaking your control.

- To solve this, the ASP.NET page framework exposes a feature in ASP.NET called control state.
- The ControlState property allows you to persist property information that is specific to a control and cannot be turned off like the ViewState property.

Hidden Field

- ASP.NET allows you to store information in a HiddenField control, which renders as a standard HTML hidden field.
- A hidden field does not render visibly in the browser, but you can set its properties just as you can with a standard control.
- When a page is submitted to the server, the content of a hidden field is sent in the HTTP form collection along with the values of other controls.
- A hidden field acts as a repository for any page-specific information that you want to store directly in the page.
- A HiddenField control stores a single variable in its Value property and must be explicitly added to the page.
- Some features of hidden fields are:
    - Contains a small amount of memory
    - Direct functionality access

Cookie

- A cookie is a small amount of data that is stored either in a text file on the client file system or in-memory in the client browser session.
- It contains site-specific information that the server sends to the client along with page output.
- Cookies can be temporary (with specific expiration times and dates) or persistent.
- You can use cookies to store information about a particular client, session, or application.
- The cookies are saved on the client device, and when the browser requests a page, the client sends the information in the cookie along with the request information.
- The server can read the cookie and extract its value.
- A typical use is to store a token (perhaps encrypted) indicating that the user has already been authenticated in your application.
- Points to Remember
    - Some features of cookies are:
    - Store information temporarily
    - It's just a simple small sized text file
    - Can be changed depending on requirements
    - User Preferred
    - Requires only a few bytes or KBs of space for creating cookies

```
int postbacks = 0;
if (Request.Cookies["number"] != null) {
        postbacks = Convert.ToInt32(Request.Cookies["number"].Value) + 1;
}
// Generating Response
else {
        postbacks = 1;
}
Response.Cookies["number"].Value = postbacks.ToString();
Result.Text = Response.Cookies["number"].Value;
```

- **Persistent Cookie:** Cookies having an expiration date is called a persistent cookie. This type of cookie reaches their end as their expiration dates comes to an end. In this cookie we set an expiration date.

```
Response.Cookies["UserName"].Value = "Abhishek";
Response.Cookies["UserName"].Expires = DateTime.Now.AddDays(1);

HttpCookie aCookie = new HttpCookie("Session");
aCookie.Value = DateTime.Now.ToString();
aCookie.Expires = DateTime.Now.AddDays(1);
Response.Cookies.Add(aCookie);
```

- **Non-Persistent Cookie:** Non-persistent types of cookies aren't stored in the client's hard drive permanently. It maintains user information as long as the user access or uses the services. It's simply the opposite procedure of a persistent cookie.

```
HttpCookie aCookie = new HttpCookie("Session");
aCookie.Value = DateTime.Now.ToString();
aCookie.Expires = DateTime.Now.AddDays(1);
Response.Cookies.Add(aCookie);
```

- **Advantages of using cookies:**
  - **Configurable expiration rules:** The cookie can expire when the browser session ends, or it can exist indefinitely on the client computer, subject to the expiration rules on the client.
  - **No server resources are required**: The cookie is stored on the client and read by the server after a post.
  - **Simplicity:** The cookie is a lightweight, text-based structure with simple key-value pairs.
  - **Data persistence:** Although the durability of the cookie on a client computer is subject to cookie expiration processes on the client and user intervention, cookies are generally the most durable form of data persistence on the client.
- **Disadvantages of using cookies:**

- **Size limitations**: Most browsers place a 4096-byte limit on the size of a cookie, although support for 8192-byte cookies is becoming more common in newer browser and client-device versions.
- **User-configured refusal**: Some users disable their browser or client device's ability to receive cookies, thereby limiting this functionality.
- **Potential security risks:** Cookies are subject to tampering. Users can manipulate cookies on their computer, which can potentially cause a security risk or cause the application that is dependent on the cookie to fail. Also, although cookies are only accessible by the domain that sent them to the client, hackers have historically found ways to access cookies from other domains on a user's computer. You can manually encrypt and decrypt cookies, but it requires extra coding and can affect application performance because of the time that is required for encryption and decryption.

Query String

- A query string is information that is appended to the end of a page URL.
- A typical query string might look like the following example:
  - http://www.gtu.ac.in/result.aspx?category=BE&semester=7
- In the URL path above, the query string starts with a question mark (?) and includes two attribute/value pairs, one called "category" and the other called "semester"
- Query strings provide a simple but limited way to maintain state information. For example, they are an easy way to pass information from one page to another, such as passing a product number from one page to another page where it will be processed.
- **Advantages of using query strings:**
  - **No server resources are required**: The query string is contained in the HTTP request for a specific URL.
  - **Widespread support**: Almost all browsers and client devices support using query strings to pass values.
  - **Simple implementation**: ASP.NET provides full support for the query-string method, including methods of reading query strings using the Params property of the HttpRequest object.
- **Disadvantages of using query strings:**
  - **Potential security risks:** The information in the query string is directly visible to the user via the browser's user interface. A user can bookmark the URL or send the URL to other users, thereby passing the information in the query string along with it. If you are concerned about any sensitive data in the query string, consider using hidden fields in a form that uses POST instead of using query strings
  - **Limited capacity:** Some browsers and client devices impose a 2083-character limit on the length of URLs.

```
// Getting data
if (Request.QueryString["number"] != null)  {
        View.Text = Request.QueryString["number"];
}

// Setting query string
int postbacks = 0;
if (Request.QueryString["number"] != null)  {
        postbacks = Convert.ToInt32(Request.QueryString["number"]) + 1;
}
else  {
        postbacks = 1;
}
Response.Redirect("default.aspx?number=" + postbacks);
```

Server Side State management:

- As name implies, state information will be maintained on the server.
- Application, Session, Cache and Database are different mechanisms for storing state on the server.
- Care must be taken to conserve server resources. For a high traffic web site with large number of concurrent users, usage of sessions object for state management can create load on server causing performance degradation

Application object:

- Application object is used to store data which is visible across entire application and shared across multiple user sessions.
- Data which needs to be persisted for entire life of application should be stored in application object.
- In classic ASP, application object is used to store connection strings. It's a great place to store data which changes infrequently.
- We should write to application variable only in application_Onstart event (global.asax) or application.lock event to avoid data conflicts.
- Below code sample gives idea

```
Application.Lock();
Application["mydata"]="mydata";
Application.UnLock();
```

Session Object

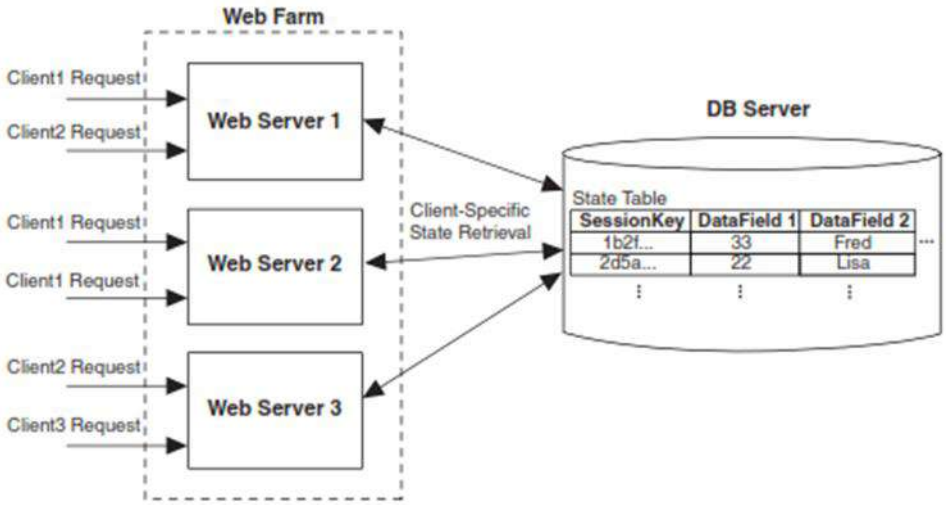
- Session object is used to store state specific information per client basis.
- It is specific to particular user.
- Session data persists for the duration of user session you can store session's data on web server in different ways.

- Session state can be configured using the <session State> section in the application's web.config file.
- Configuration information:

```
<sessionState mode = <"inproc" | "sqlserver" | "stateserver">
        cookieless = <"true" | "false">
        timeout = <positive integer indicating the session timeout in minutes>
        sqlconnectionstring = <SQL connection string that is only used in the SQLServer mode>
        server = <The server name that is only required when the mode is State Server>
        port = <The port number that is only required when the mode is State Server>
```

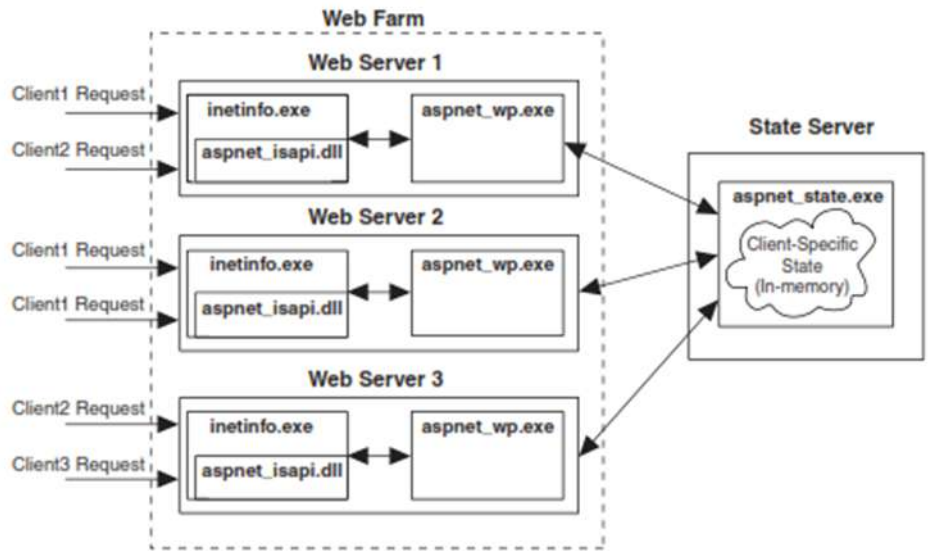
- **Mode:**
  - This setting supports three options. They are InProc, SQLServer, and State Server
- **Cookie less:**
  - This setting takes a Boolean value of either true or false to indicate whether the Session is a cookie less one.
- **Timeout:**
  - This indicates the Session timeout vale in minutes. This is the duration for which a user's session is active. Note that the session timeout is a sliding value; Default session timeout value is 20 minutes
- **SqlConnectionString:**
  - This identifies the database connection string that names the database used for mode SQLServer.
- **Server:**
  - In the out-of-process mode State Server, it names the server that is running the required Windows NT service: aspnet_state.
- **Port:**
  - This identifies the port number that corresponds to the server setting for mode State Server. Note that a port is an unsigned integer that uniquely identifies a process running over a network.
- Session state in ASP.NET can be configured in different ways based on various parameters including scalability, maintainability and availability
  - **In process mode (in-memory)**- State information is stored in memory of web server
  - **Out-of-process mode-** Session state is held in a process called aspnet_state.exe that runs as a windows service.
  - **Database mode-** Session state is maintained on a SQL Server database.
- **In process mode:**
  - This mode is useful for small applications which can be hosted on a single server. This model is most common and default method to store session specific information.
  - Session data is stored in memory of local web server
  - Advantages:
    - Fastest mode
    - Simple configuration
  - Disadvantages:
    - Session data will be lost if the worker process or application domain recycles

- Not ideal for web gardens and web farms



Maintaining Client-Specific State in a Web Farm Deployment

- **Out-of-process Session mode (state server mode):**
  - This mode is ideal for scalable and highly available applications.
  - Session state is held in a process called aspnet_state.exe that runs as a windows service which listens on TCP port 42424 by default.
  - You can invoke state service using services MMC snap-in or by running following net command from command line. **Net start aspnet_state**
  - Advantages:
    - Supports web farm and web garden configuration
    - Session data is persisted across application domain recycles. This is achieved by using separate worker process for maintaining state
  - Disadvantages:
    - Out-of-process mode provides slower access compared to In process
    - Requires serializing data



Using a State Server in a Web Farm Deployment

- **SQL-Backed Session state:**
  - ASP.NET sessions can also be stored in a SQL Server database.
  - Storing sessions in SQL Server offers resilience that can serve sessions to a large web farm that persists across IIS restarts.
  - SQL based Session state is configured with aspnet_regsql.exe. This utility is located in .NET Framework's installed directory
  - C:\<windows>\microsoft.net\framework\<version>. Running this utility will create a database which will manage the session state.
  - Advantages:
    - Supports web farm and web garden configuration
    - Session state is persisted across application domain recycles and even IIS restarts when session is maintained on different server.
  - Disadvantages:
    - Requires serialization of objects

Explain the concepts of Master Page and Nested Master Page in ASP.Net

Master page

- ASP.NET master pages allow you to create a consistent layout for the pages in your application.
- A single master page defines the look and feel and standard behavior that you want for all of the pages (or a group of pages) in your application.
- You can then create individual content pages that contain the content you want to display.

- When users request the content pages, they merge with the master page to produce output that combines the layout of the master page with the content from the content page.

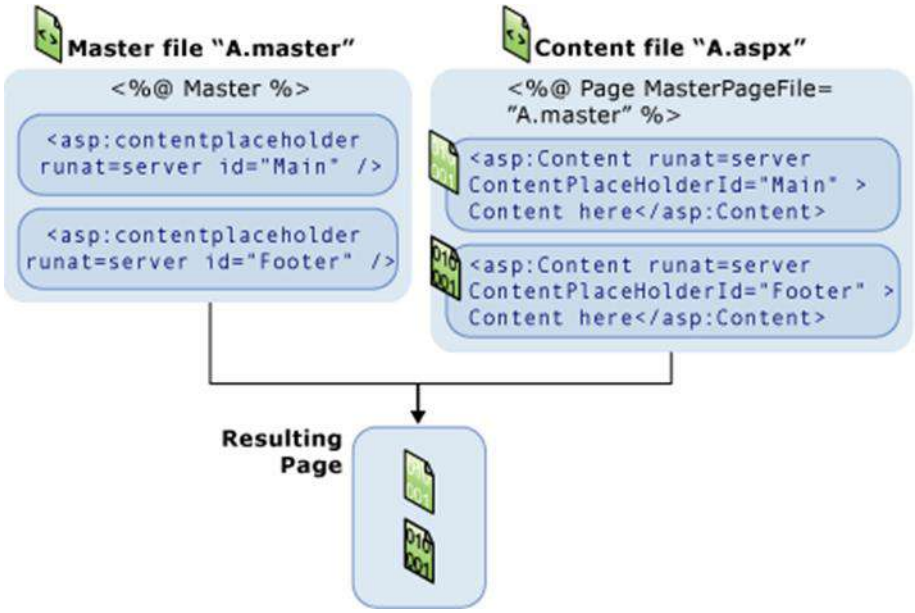
How Master Pages Work

- Master pages actually consist of two pieces, the master page itself and one or more content pages.
- **Master Pages**
  - A master page is an ASP.NET file with the extension .master (for example, MySite.master) with a predefined layout that can include static text, HTML elements, and server controls.
  - The master page is identified by a special @ Master directive that replaces the @ Page directive that is used for ordinary .aspx pages.
- **Replaceable Content Placeholders**
  - In addition to static text and controls that will appear on all pages, the master page also includes one or more ContentPlaceHolder controls.
  - These placeholder controls define regions where replaceable content will appear.
  - In turn, the replaceable content is defined in content pages. After you have defined the ContentPlaceHolder controls, a master page might look like the following.

```
<%@ Master Language="C#" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
        <head runat="server" >
                <title>Master page title</title>
        </head>
        <body>
                <form id="form1" runat="server">
                <table>
                        <tr> <td><asp:contentplaceholder id="Main" runat="server" /></td>
                         <td><asp:contentplaceholder id="Footer" runat="server" /></td></tr>
                </table> </form></body>
</html>
```

- **Content Pages**
  - You define the content for the master page's placeholder controls by creating individual content pages, which are ASP.NET pages (.aspx files and, optionally, code-behind files) that are bound to a specific master page.
  - The binding is established in the content page's @ Page directive by including a MasterPageFile attribute that points to the master page to be used.
  - After creating Content controls, you add text and controls to them.
  - In a content page, anything that is not inside the Content controls (except script blocks for server code) results in an error. You can perform any tasks in a content page that you do in an ASP.NET page. For example, you can generate content for a Content control using server controls and database queries or other dynamic mechanisms.
  - A content page might look like the following.

```
<% @ Page Language="C#" MasterPageFile="~/Master.master" Title="Content Page 1" %>
<asp:Content ID="Content1" ContentPlaceHolderID="Main" Runat="Server">
        Main content.
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="Footer" Runat="Server" >
        Footer content.
</asp:content>
```



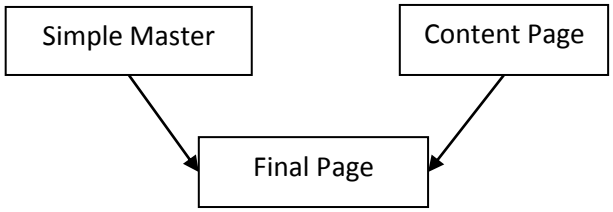
Advantages of Master Pages

- They allow you to centralize the common functionality of your pages so that you can make updates in just one place.
- They make it easy to create one set of controls and code and apply the results to a set of pages. For example, you can use controls on the master page to create a menu that applies to all pages.

- They give you fine-grained control over the layout of the final page by allowing you to control how the placeholder controls are rendered.
- They provide an object model that allows you to customize the master page from individual content pages

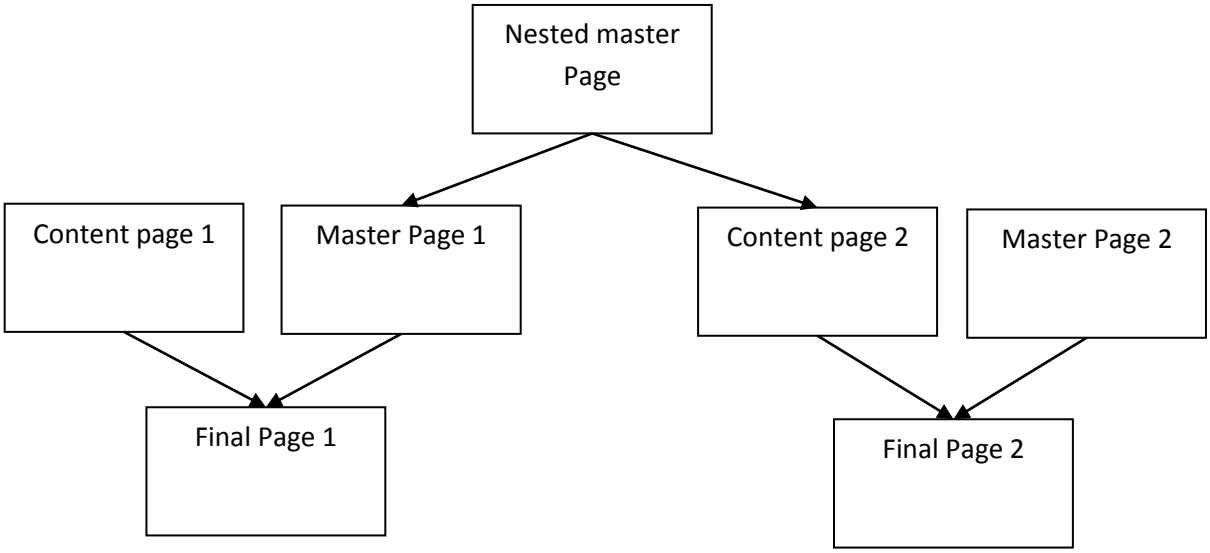
Master pages are of two types:

Simple Master Page



Nested Master Page:

- Here you can merge the master page and content page.
- Simply, Main master page → another master page for some selected pages that is called nested master page.
- It is used to give common and distinguish look from master page for some special module.



Explain the ASP.Net Themes with Skin file

- A theme is a collection of property settings that allow you to define the look of pages and controls, and then apply the look consistently across pages in a Web application, across an entire Web application, or across all Web applications on a server.
- **Themes and Control Skins**
  - Themes are made up of a set of elements: skins, cascading style sheets (CSS), images, and other resources. At a minimum, a theme will contain skins.
  - Themes are defined in special directories in your Web site or on your Web server.
- **Skins**
  - A skin file has the file name extension .skin and contains property settings for individual controls such as Button, Label, TextBox, or Calendar controls.
  - Control skin settings are like the control markup itself, but contain only the properties you want to set as part of the theme.
  - For example, the following is a control skin for a Button control:
    <asp:button runat="server" BackColor="lightblue" ForeColor="black" />
  - You create .skin files in the Theme folder.
  - A .skin file can contain one or more control skins for one or more control types.
  - You can define skins in a separate file for each control or define all the skins for a theme in a single file.
  - There are two types of control skins, default skins and named skins:
    - **A default skin** automatically applies to all controls of the same type when a theme is applied to a page. A control skin is a default skin if it does not have a SkinID attribute. For example, if you create a default skin for a Calendar control, the control skin applies to all Calendar controls on pages that use the theme. (Default skins are matched exactly by control type, so that a Button control skin applies to all Button controls, but not to LinkButton controls or to controls that derive from the Button object.)
    - **A named skin** is a control skin with a SkinID property set. Named skins do not automatically apply to controls by type. Instead, you explicitly apply a named skin to a control by setting the control's SkinID property. Creating named skins allows you to set different skins for different instances of the same control in an application.
- **Cascading Style Sheets**
  - A theme can also include a cascading style sheet (.css file).
  - When you put a .css file in the theme folder, the style sheet is applied automatically as part of the theme.
  - You define a style sheet using the file name extension .css in the theme folder.
- **Theme Graphics and Other Resources**
- Themes can also include graphics and other resources, such as script files or sound files. For example, part of your page theme might include a skin for a TreeView control. As part of the theme, you can include the graphics used to represent the expand button and the collapse button.

- Typically, the resource files for the theme are in the same folder as the skin files for that theme, but they can be elsewhere in the Web application, in a subfolder of the theme folder.

Scoping Themes

- You can define themes for a single Web application, or as global themes that can be used by all applications on a Web server.
- After a theme is defined,
  - it can be placed on individual pages using the Theme or StyleSheetTheme attribute of the @ Page directive
  - It can be applied to all pages in an application by setting the <pages> element in the application configuration file. If the <pages> element is defined in the Machine.config file, the theme will apply to all pages in Web applications on the server.

Page Themes

- A page theme is a theme folder with control skins, style sheets, graphics files and other resources created as a subfolder of the \App_Themes folder in your Web site.
- Each theme is a different subfolder of the \App_Themes folder.
- The following example shows a typical page theme, defining two themes named BlueTheme and PinkTheme.

```
MyWebSite
        App_Themes
                BlueTheme
                        Controls.skin
                         BlueTheme.css
                PinkTheme
                        Controls.skin
                        PinkTheme.css
```

What is web Service in .Net? Write a Program to create Web Service and another program to consume same web service. (Take simple web method to add two integer numbers)

- A web service is an entity that you can develop to provide particular functionality over the internet.
- For example weather information, live score of cricket, simple and compound interest formula.
- For such purposes you make a web service for each. It will useful not only for one website but also for many other websites those are providing same facilities.
- Developer just needs to consume the web services and then its required function is working.
- So, it is better to develop a web service common to all instead of writing the same business logic for all websites.
- Web service provides common functionalities and results for all websites.
- Irrespective from any programming language as web service is based on XML so can be useful to all language based applications.
- Web Service messages are formatted as XML, a standard way for communication between two incompatible system, and this message is sent via HTTP, so that they can reach to any machine on the internet without being blocked by firewall.
- A web service is
    - Language Independent.
    - Protocol Independent.
    - Platform Independent.
    - It assumes stateless service architecture.
    - Scalable (e.g. multiplying two numbers together to an entire customer-relationship management system).
    - Programmable (encapsulates a task).
    - Based on XML (open, text-based standard).
    - Self-describing (metadata for access and use).
    - Discoverable (search and locate in registries)- ability of applications and developers to search for and locate desired Web services through registries. This is based on UDDI.

Creating a Web Service

- Let us write a simple module that will provide basic arithmetic operations to other applications. The user of this application will have to provide us 2 numbers and it will perform the requested operation like Addition, Subtraction and Multiplication.
- So, the first thing to do is to create our web service. This can be done by creating a website of type ASP.NET Web service. This will give a skeleton to write web service code.

```
[WebService(Namespace = "http://tempuri.org/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
public class Service : System.Web.Services.WebService
{
    public Service ()
    {
        //Uncomment the following line if using designed components
        //InitializeComponent();
    }

    [WebMethod]
    public int Add(int x, int y)
    {
        return x+y;
    }

    [WebMethod]
    public int Subtract(int x, int y)
    {
        return x - y;
    }

    [WebMethod]
    public int Multiply(int x, int y)
    {
        return x * y;
    }
}
```

Consuming a WebService

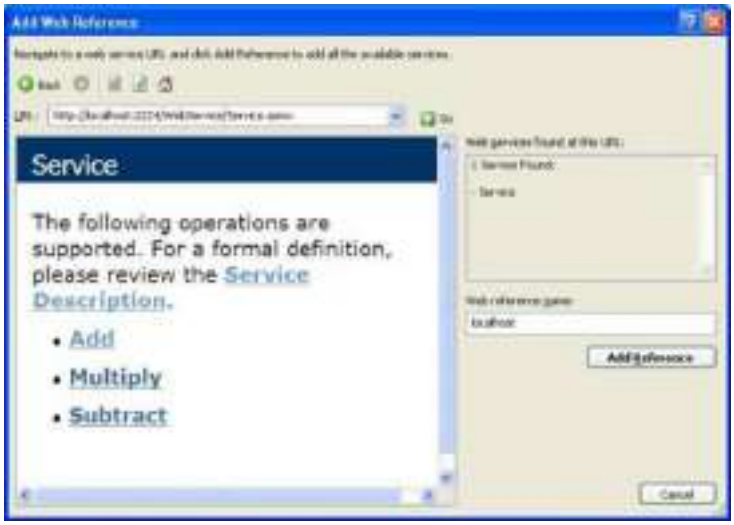
There are three ways we can consume a WebService:

1. Using HTTP-POST method.
2. Using XMLHttp that will use SOAP to call our the service
3. Using WSDL generated proxy class

1. The HTTP-Post method can be used by calling *.asmx* file directly from client. We can directly use the method name and the parameter names will be taken from our input fields on form.

```
<html>
    <body>
        <form action="http://localhost/.../Service.asmx/Multiply"  method="POST">
                <input name="x"></input>
                <input name="y"></input>

                <input type="submit" value="Enter"> </input>
        </form>
    </body>
</html>
```

2. The second method where we can use the `XMLHttp` over SOAP to access the webservice is used when we want to use the web service using full capability of SOAP.
3. The third way of using the web service is by generating a Proxy class for the web service and then using that proxy class. This method is more type safe and less error prone as once we have the proxy class generated, it can take care of SOAP messages, serialization and ensure that all the problems can be handled at compile time instead of runtime. To use this service, we need to do "Add Web reference" and add the webservice reference. Alternatively we can also use WSDL.exe, a command line tool, to generate the proxy classes and use them. "Add web Reference" is also using the same WSDL.exe to generate the proxy classes



Now let us write some code to use the proxy class to perform the operations, the proxy class will then communicate to the web service using SOAP and get back the results to us.

```
protected void Button1_Click(object sender, EventArgs e)
{
    if (txtX.Text != string.Empty
        && txtY.Text != string.Empty)
    {
        try
        {
            int x = Convert.ToInt32(txtX.Text);
            int y = Convert.ToInt32(txtY.Text);

            ArithmeticService.Service service = new ArithmeticService.Service();

            Label1.Text = "Sum: " + service.Add(x, y).ToString();
            Label2.Text = "Difference: " + service.Subtract(x, y).ToString();
            Label3.Text = "Multiplication: " + service.Multiply(x, y).ToString();
        }
        catch(Exception)
        {
            throw;
        }
    }
}
```

Advanced .Net Concepts

Explain WPF with its features

- **Windows Presentation Foundation (WPF)** is a next-generation presentation system for building Windows client applications with visually stunning user experiences.
- With WPF, we can create a wide range of both standalone and browser-hosted applications.
- The core of WPF is a resolution-independent and vector-based rendering engine that is built to take advantage of modern graphics hardware.
- WPF extends the core with a comprehensive set of application-development features that include Extensible Application Markup Language (XAML), controls, data binding, layout, 2-D and 3-D graphics, animation, styles, templates, documents, media, text, and typography.
- WPF is included in the Microsoft .NET Framework, so we can build applications that incorporate other elements of the .NET Framework class library.

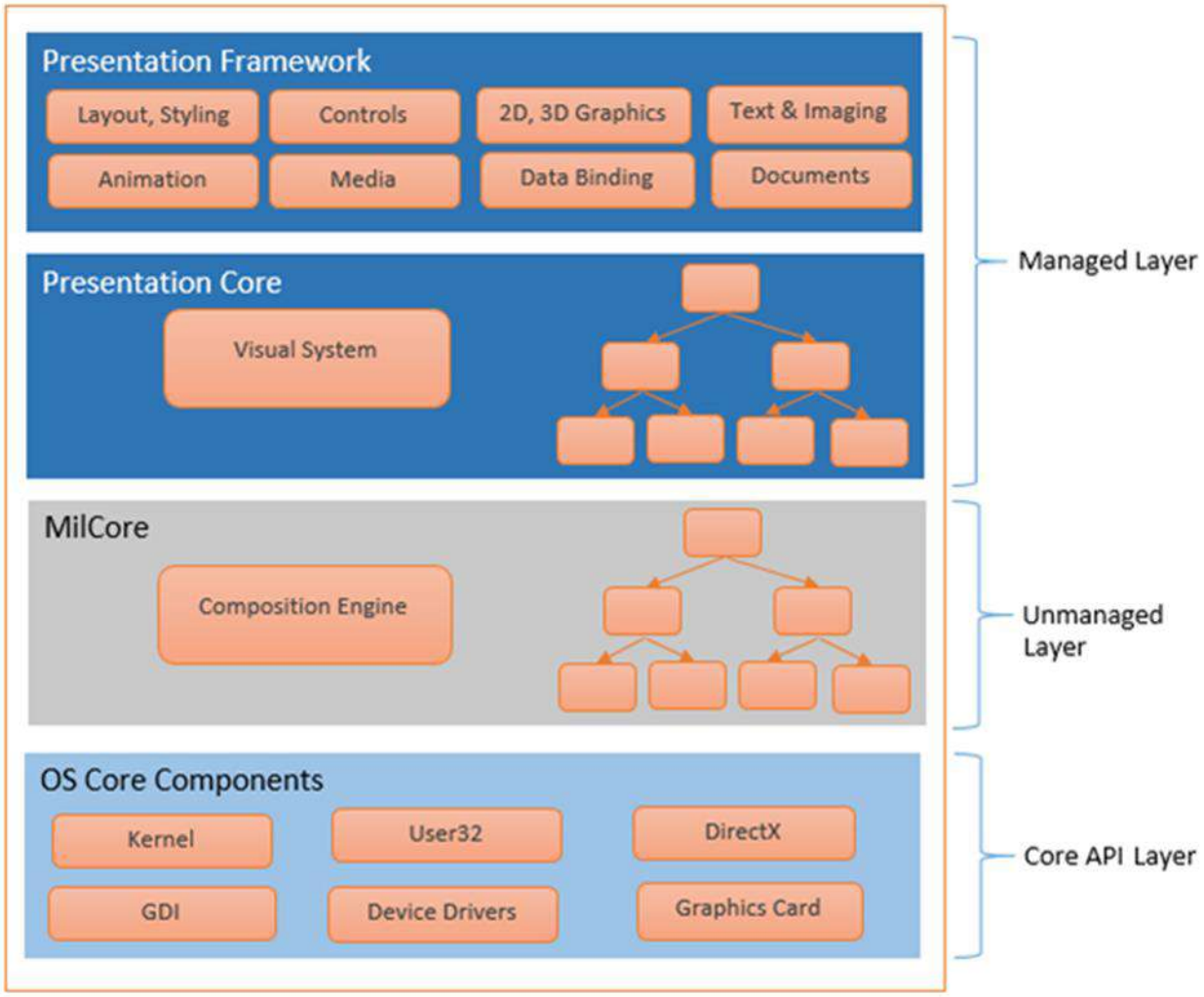
Features of WPF:

- **Resolution Independence**
  WPF is resolution independence since all measures in WPF are logical units not pixels. A logical unit is a 1/96 of an inch. So, with changing the screen resolution setting in WPF each control will look same for each resolution. It is not based on Dots per inch (DPI) setting of the device.
- **Separation of appearance and behaviors**
  WPF separates the appearance of an UI from its behavior. The appearance is specified by XAML and behavior is specified by a managed programming language like C# or VB.
- **Built-In support for graphics and animation**
  WPF applications run within DirectX environment, hence it has major support of graphics and animation capabilities. WPF has a separate set of classes that are specifically deal with animation effects and graphics.
- **Supports for Audio and Video**
  WPF has support for playing any audio or video file supported by Windows Media Player. It also gives you the tools to integrate video content into your rich UI such as placing a video window on a spinning 3-D cube.
- **Highly customizable**
  WPF supports separation of appearance and behaviors; hence we can easily change the look of a control or a set of controls. This concept of styling controls in WPF is almost like CSS in HTML. In WPF, you we store styles, controls, animations, and even any object as a resource and we may associate that resource to the controls. Each resource is declared once when the form loads itself.

Architecture of WPF

WPF is actually a set of assemblies that build up the entire framework. These assemblies can be categorized as:

1. Managed Layer
2. UnManaged Layer
3. Core API



WPF Architecture

1. **Managed Layer :**
   Managed layer has two main components – Presentation Framework and Presentation Core.
   a. **Presentation Framework** provides the required functionalities that we need to build the WPF applications such as controls, data bindings, styling, shapes, media, documents, annotations, animation and more. PresentationFamework.dll is responsible for this purpose.

b. **Presentation Core** acts as a managed wrapper around MILCore and provides public interface for MIL. Presentation Core is the home for WPF Visual System and provides classes for creating application visual tree. The Visual System creates visual tree which contains applications Visual Elements and rendering instructions. PresentationCore.dll is responsible for this purpose.

2. **Unmanaged Layer**

This layer is also called milcore or Media Integration Library Core. MilCore is written in unmanaged code in order to enable tight integration with DirectX. DirectX engine is underlying technology used in WPF to display all graphics, allowing for efficient hardware and software rendering. MIL has Composition System that receives rendering instructions from Visual System and translates into data that can be understood by DirectX to render user interface.

3. **Core API Layer**
   This layer has OS core components like Kernel, User32, GDI, Device Drivers, Graphic cards etc. These components are used by the application to access low level APIs. User32 manages memory and process separation.

Explain WCF with Example

- **Windows Communication Foundation - WCF** (Code named Indigo) is a programming platform and runtime system for building, configuring and deploying network-distributed services.
- It is the latest service oriented technology; Interoperability is the fundamental characteristics of WCF.
- It is unified programming model provided in .Net Framework 3.0.
- WCF is a combined feature of Web Service, Remoting, MSMQ and COM+.
- WCF provides a common platform for all .NET communication.

Advantages of WCF

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

Difference between WCF and Web service

Features	Web Service	WCF
Hosting	It can be hosted in IIS	It can be hosted in IIS, windows activation service, Self-hosting, Windows service
Programming	[WebService] attribute has to be added to the class	[ServiceContract] attribute has to be added to the class
Model	[WebMethod] attribute represents the method exposed to client	[OperationContract] attribute represents the method exposed to client
Operation	One-way, Request- Response are the different operations supported in web service	One-Way, Request-Response, Duplex are different type of operations supported in WCF
XML	System.Xml.serialization name space is used for serialization	System.Runtime.Serialization namespace is used for serialization
Encoding	XML 1.0, MTOM(Message Transmission Optimization Mechanism), DIME, Custom	XML 1.0, MTOM, Binary, Custom
Transports	Can be accessed through HTTP, TCP, Custom	Can be accessed through HTTP, TCP, Named pipes, MSMQ,P2P, Custom
Protocols	Security	Security, Reliable messaging, Transactions

A WCF Service is composed of three components parts viz,

1. **Service Class -** A WCF service class implements some service as a set of methods.
2. **Host Environment -** A Host environment can be a Console application or a Windows Service or a Windows Forms application or IIS as in case of the normal asmx web service in .NET.
3. **Endpoints -** All communications with the WCF service will happen via the endpoints. The endpoint is composed of 3 parts (collectively called as ABC's of endpoint) as defines below:
    a. **Address:** The endpoints specify an Address that defines where the endpoint is hosted. It's basically url.
    b. **Binding:** The endpoints also define a binding that specifies how a client will communicate with the service and the address where the endpoint is hosted.
    c. **Contract:** The endpoints specify a Contract that defines which methods of the Service class will be accessible via the endpoint; each endpoint may expose a different set of methods.

Creating simple application using WCF (Demo of the Calculator Example)

Step By Step Procedure

1. Creating WCF Service
2. Consuming WCF Service in Web Application using Visual Studio

Creating WCF Service.

Steps:

1. Open Visual Studio > Projects > Select C# > WCF > WCF Service Library.

2. Type name of the service file as u need. In this ex .MyWCFSolution. then click OK.
3. New Window will open with one Class.cs and Interface .cs File which can be seen in Solution Explore.
4. Add the Namespace System.ServiceModel and also add Reference in Solution Explore.
5. Type the Code in Interface file and class file as shown below.

Code in Interface(EX: IService1.cs)

```
[ServiceContract]
public interface IService1
{
        [OperationContract]
        int add(int a,int b);

        [OperationContract]
        int Sub(int a, int b);

        [OperationContract]
        int Mul(int a, int b);

        [OperationContract]
        int Div(int a, int b);

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

Code in Class file(EX: Service1.cs)

```
public class Service1 : IService1
{
      int IService1.add(int a, int b)
      {
            return (a + b);
      }

      int IService1.Sub(int a, int b)
      {
            if (a >= b)
            {
                  return (a - b);
            }
            else
            {
                  return (b - a);
            }
      }

      public int Mul(int a, int b)
      {
            return (a * b);
      }

      public int Div(int a, int b)
      {
            if(a>=b)
            {
                  try
                  {     return (a / b);   }
                  catch
                  {     return 0;   }
            }
            else
            {
                  return (b / a);
            }
      }
}
```

Consuming WCF Service in Web Application.

Steps:

1. Create web Application as shown in attachment please download the File.
2. After Creating Design of the Calculator web application the add reference System. Service Model namespace into web application in Solution Explore.

3. Add the WCF Service in visual studio by right click on Reference and select Add service Reference
4. Give the Service hosted Address and click on Go button and select the service which u created I mean in Ex service1.
5. Give the namespace: Name as Myservice and the click OK .
   I. Creating WCF Client.
      - In Order to call the Methods (services from WCFService) we need to Create the porxy WCF Client.
      - Once service is consumed in Application, WCF Client is Created, So Using the name of the Client Name we can call the methods in out applications
      - EX. In Our WCF Service we have 4 methods (add, sub, mul, div).
   II. To get the WCF client name Double click on Service i.e Myservice and take the copy of the Client name.
   III. Paste the Client Name in Default.cs File get the following code then create the instance of the client as show in below code. Using this porxy client u can call the services of the WCF service as shown in below code.
   IV. On paste u get MY.Myservice.Service1Client
   V. Code to be Right in Default.cs in web application(EX:Calculator)

```
namespace Calculator
{
    public partial class _Default : System.Web.UI.Page
    {
      Calculator.Myservice.Service1Client Client
      = new Calculator.Myservice.Service1Client();
      //Creating proxy client instance "Client"
        int a, b;
        protected void Page_Load(object sender, EventArgs e)
        {
            Response.Write("MY Calculator");
        }

        protected void btnAdd_Click(object sender, EventArgs e)
        {
            a = Convert.ToInt32(txt1st.Text);
            b = Convert.ToInt32(txt2nd.Text);

            int Addition = Client.add(10,20);
            txtResult.Text = Addition.ToString();
        }

        protected void btnsub_Click(object sender, EventArgs e)
        {
            a = Convert.ToInt32(txt1st.Text);
            b = Convert.ToInt32(txt2nd.Text);
            int Addition = Client.Sub(10, 20);
            txtResult.Text = Addition.ToString();
         }

        protected void btnmul_Click(object sender, EventArgs e)
        {
            a = Convert.ToInt32(txt1st.Text);
            b = Convert.ToInt32(txt2nd.Text);
            int Addition = Client.Mul(10, 20);
            txtResult.Text = Addition.ToString();
        }

        protected void btndiv_Click(object sender, EventArgs e)
        {
            a = Convert.ToInt32(txt1st.Text);
            b = Convert.ToInt32(txt2nd.Text);
            int Addition = Client.Div(10, 20);
            txtResult.Text = Addition.ToString();
         }
    }
}
```