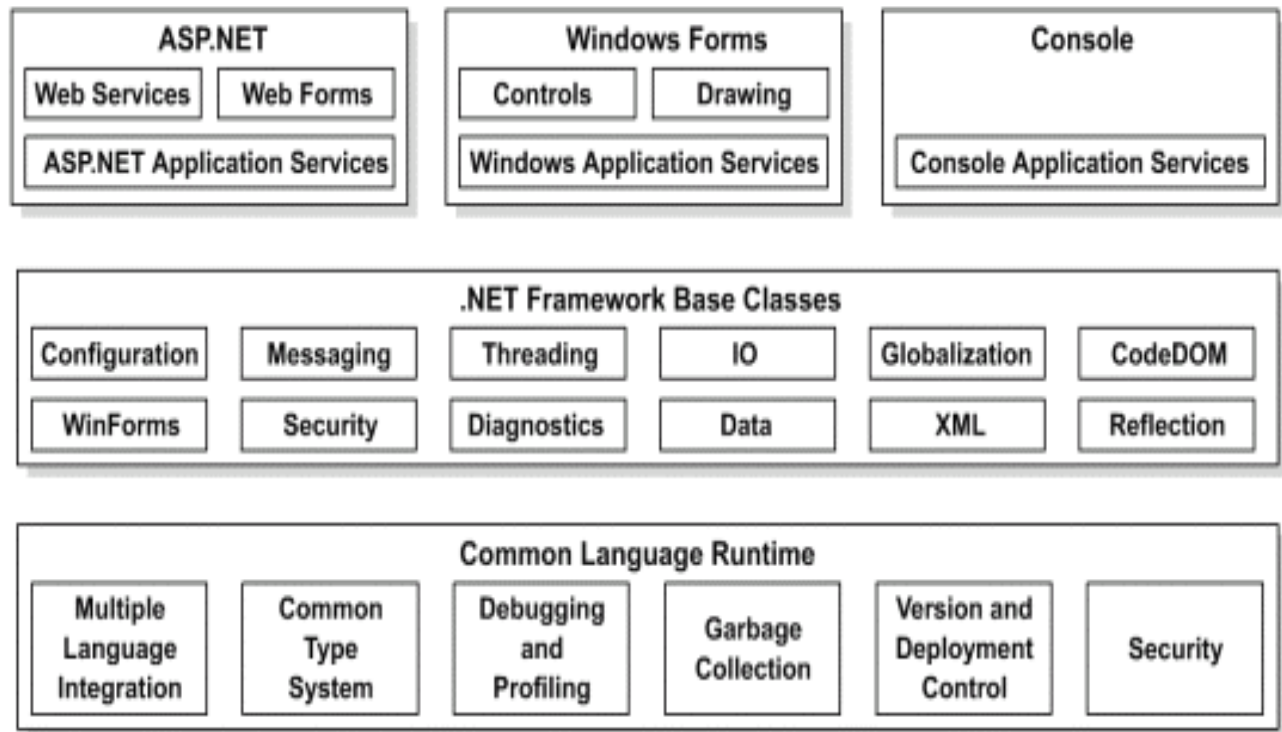


Chapter-1 :- Framework & Visual Studio IDE , Language Basics

• .NET Framework :

- The .NET Framework is a service or platform for building, deploying, and running applications. The .NET Framework consists of 2 main parts: common language runtime and class libraries



- There are many features provided by .NET Framework which has made .NET popular and reliable in software development and web development industry. Following are features of .NET Platform / .NET Framework.

1) Multilanguage Development

C#.NET supports multiple languages. This is definitely one of the biggest advantages of .NET Framework because programmers having ability in their own languages, can use their skills in their languages. Another advantage of Multilanguage is that all are developed under same basic environment.

2) Multi-Device Development

Apart from that .NET supports multiple developments. You can create Mobile Application, PDA Application, etc.

3) Platform and Processor independence

Generally when you compile a code written in some language, it is converted directly to Native Code i.e. EXE or DLL. In C#.NET execution of programs is done in two process. First program is converted from language code to IL Code and then from IL Code to Native Code, which makes .NET application to become Platform and Processor independence.

4) Automatic memory management

Memory managed is always one of biggest headache of Developers. C#.NET handles memory managed by itself. Under Garbage Collection method, it automatically collects the objects which are no longer needed and removes it from memory

5) Easy Deployment

In many languages, Deployment is one of the tedious task. Using C#.NET application becomes easy to deploy. You can create Deployment project easily which helps to deploy application on target machines.

6) Distributed Architecture

C#.NET applications have capability to be executed on Distributed Architecture. You can create applications which can be executed on Distributed Architecture.

7) Interoperability with Unmanaged code

Because interaction between new and older applications is commonly required, the .NET Framework provides means to access functionality that is implemented in programs that execute outside the .NET environment. So, Interoperability with Unmanaged Code is provided.

8) Security

The design is meant to address some of the vulnerabilities, such as buffer overflows, that been exploited by malicious software. Additionally, .NET provides a common security model for all applications.

9) Performance and Scalability

As far as Performance and Scalability is concerned, .NET based applications give better performance in terms of memory, device management, etc. You can create Robust Application with full scalability provided by application.

10) XML Support

Today XML is used widely for the transportation of data between Client and Server via HTTP. Because XML works in Text which can be understood by all OS and Hardwares. .NET supports writing, manipulating and transforming of XML documents.

Components Of .NET Architecture:

CLR (Common Language Runtime):

- It is the execution engine for .NET Framework applications.
- It is the heart or backbone of the .NET.
- It's is the runtime engine provided by the .NET framework.
- It provides an infrastructure for run programs and allows them to communicate with other parts of the .NET framework.
- It provides a number of services, including the following:
- Code loading and execution
- Application memory isolation
- Verification of type safety
- Conversion of IL to native code
- Access to metadata
- Managing memory for managed objects
- Enforcement of code access security
- Exception handling, including cross-language exceptions

CTS (Common Type System):

- CTS allow programs written in different programming languages to easily share information.
- A class written in C# should be equivalent to a class written in vb.
- Languages must agree on the meanings of these concepts before they can integrate with one another.
- CTS forms a subset of CLS. This implies that all the rules that apply to CTS apply to CLS also.

- **It defines rules** that a programming language must follow to ensure that objects written in different programming languages can interact with each other.
- CTS provide cross language integration.
- The common type system supports two general categories of types:

- 1) Value types
- 2) Reference types

CLS (Common Language Specification):

- CLS includes basic language features needed by almost all the applications.
- It serves as a guide for library writers and compiler writers.
- The Common Language Specification is a subset of the common type system.
- The Common Language Specification is also important to application developers who are writing code that will be used by other developers.

Assembly:

- An assembly is the primary building block of a .NET Framework **application.**
- An Assembly is a logical DLL.
- It consists of DLLs or executables.
- It is a collection of functionality that is built, versioned, and deployed as a single implementation unit (as one or more files).
- **All managed types and resources are marked either as accessible only within their implementation unit or as accessible by code outside that unit.**
- **There are mainly two types of assemblies:**
 - 1) **Private Assembly**
 - 2) **Shared Assembly**
- A private assembly is used only by a single application, and is stored in that application's install directory (or a subdirectory therein).
- A shared assembly is one that can be referenced by more than one application.
- In order to share an assembly, the assembly must be explicitly built for this purpose by giving it a cryptographically strong name (referred to as a strong name).
- By contrast, a private assembly name need only be unique within the application that uses it.

Meta Data:

- Metadata stored within the Assembly.
- NET records information about compiled classes as Metadata.
- Metadata means data about data.
- A .NET language compiler will generate the metadata and store this in the assembly.
- On the .NET Platform programs are compiled into .NET PE (Portable Executable) files.
- The header section of every .NET PE file contains a special new section for Metadata.
- Metadata is nothing but a description of every namespace, class, method, property etc. contained within the PE file.
- The CLR uses this metadata to
 - Locate classes
 - Load classes
 - Generate native code
 - Provides security

FCL (Framework Class Library):

- In C, <conio.h>, <stdio.h> etc. are header files. We add those header files in our program to use inbuilt functions.
- Same here, the .NET Framework are collection of classes or namespace that can be used to develop applications.
- The class library consists of data classes, XML classes, Web Forms classes and Windows Forms classes, Smart device classes, Input Output classes.
- Other name of FCL is BCL - Base class library.

Namespace:

- As above, the .NET Framework class library is collection of namespaces.
- Namespace is a logical naming scheme for types that have related functionality.
- Namespace means nothing but a logical container or partition.
- It's like Drives of our computer.
- Like my computer contain C:, D:, E: and F: . My F: contains songs and videos my C: contains installed file so on.
- For example, my friend wants songs. So I will directly go to my computer's F: because songs are placed there.

○ **Common Namespaces**

Namespace	What It Contains	Example Classes and Subnamespaces
System.Collections	Creation and management of various types of collections	Arraylist, Hashtable, SortedList
System.Data	Classes and types related to basic database management (see Chapter 11 for details)	DataSet, DataTable, DataColumn,
System.Diagnostics	Classes to debug an application and to trace the execution of code	Debug, Trace
System.IO	writing to and from files and other data streams	File, FileStream, Path, StreamReader, StreamWriter
System.Math	Members to calculate common mathematical quantities, such as trigonometric and logarithmic functions	Sqrt (square root), Cos (cosine), Log (logarithm), Min (minimum)
System.Reflection	Capability to inspect metadata	Assembly, Module
System.Security	Types that enable security capabilities (see Chapter 24 for details)	Cryptography, Permissions, Policy

an object is no

es the garbage

collection mechanism.

- The CLR's garbage collector (GC) manages the allocation and release of memory for an application.
- We do not have to write code to perform memory management tasks when you develop managed applications.

- There are some steps that are carried out while executing C#.NET program. They are :

Program Execution of C#.NET

- Select Compiler
- Compile Code to MSIL
- Convert MSIL to Native Code
- Execute Code



Step-1 :

First of all C#.NET finds that which program is being compiled. According to the program, C#.NET selects compiler. For C#.NET it selects C#C (C#Compiler), For C# it selects CSC (CSharpCompiler), and similar to different languages which are supported.

Step-2 :

After selecting compiler, it will convert the language code to Intermediate Language. So that, can be clubbed together with other ILs which are produced by other languages. This Intermediate Language (Code) is known as MSIL (MicroSoft Intermediate Language), CIL (Common Intermediate Language) or IL (Intermediate Language).

Step-3 :

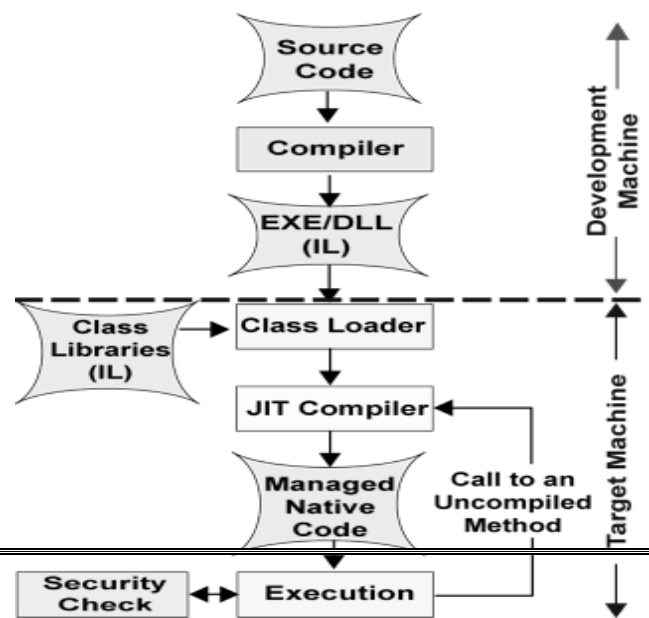
After converting program to MSIL. It is converted to Native Code (Binary Code) so that it can be executed on machine.

Step-4 :

Now as MSIL is converted into Native Code (Binary Code), it can be executed.

Microsoft Intermediate Language (MSIL):

- MSIL is the CPU-independent instruction set into which .NET Framework programs are compiled.
- It contains instructions for loading, storing, initializing, and calling methods on objects.
- Combined with metadata and the common type system, MSIL allows for true cross-language integration.
- MSIL also known as CIL - Common Intermediate Language or IL-Intermediate



Language

Just In Time Compiler (JIT):

- It stands for "just-in-time".
- It's a smart compiler.
- JIT does not compile whole program each time and every time. It compiles only that portion of the program which functions are called that time. And suppose Native code is already present then that data will not again be compiled. If changes are made then it is possible that it will again generate MSIL to Native.
- Firstly any program is compiled by its own compiler then it will convert into MSIL then with the help of JIT; MSIL is compiled into Native code but CLR does not convert whole MSIL code to Native code on load time of that application; instead of it compiles the MSIL instructions as they are called.
- There are 3 types of JIT
 - **Pre-JIT** - It compiles complete program into native code in a single compilation cycle. This work is done at the time of deployment of the program.
 - **Econo-JIT** - It compiles only those methods that are called at runtime.
 - **Normal-JIT** - It's like Econo-JIT. The methods are compiled the 1st time they are stored in cache. When the same methods are called again the compiled code from cache is used for execution.

Managed & Unmanaged Code:

Managed Code

- Managed Code is what C#.NET, J# and vb.net Compilers create.
- Code that targets the CLR (Common Language Runtime), the foundation of .NET Framework, is known as Managed Code.
- It compiles IL (Intermediate Code), not to machine code that could run directly on your computer. The IL is kept in a file called an Assembly which is known as **AssemblyInfo.C#** file., along with metadata that describes the classes, methods, and attributes of the code which you have created. You can copy it to another server / pc to deploy the assembly there – and often that copying is the only step required in the deployment.
- Managed code runs in the CLR. The runtime offers a wide variety of services to your running code.
- The managed code is always executed by a managed runtime execution environment rather than the operating system directly.
- Applications written in Java, C#, vb.NET, etc. target a runtime environment which manages the execution and the code written using these types of languages is known as Managed Code.
- Managed Code is always compiled to IL, so it provides platform independence.
- Managed Code provides information to allow the CLR to locate methods encoded in assembly modules, store and retrieve security information, handle exception, and walk the program stack.
- Managed code can access both managed data and unmanaged data.

Unmanaged Code

- Code that does not target the CLR (Common Language Runtime) is known as Unmanaged Code.
- Unmanaged code is what you use to make before C#.NET 2003 was released.
- Code that is directly executed by the OS is known as Unmanaged Code.
- Typically applications written in C# 6, C++, C, COM components, ActiveX components are an example of Unmanaged Code.
- Unmanaged Code typically targets the processor architecture and is always dependent on the computer architecture.
- Unmanaged code is always compiled to target a specific architecture and will only run on the intended platform, this means if you want to execute the same application on different machines, you need to recompile your program again.
- Unmanaged code is always compiled to the native code which is architecture specific.

Reflection:

- Reflection is a feature that enables you to obtain information about a type.
- Using this information you can construct and use objects at runtime.
- Reflection is the ability of managed code to read its own metadata for the purpose of encapsulating the type information.
- In other words, Reflection is a powerful mechanism because it allows learning and using the capabilities of types that are known only at runtime.
- Reflection is similar to C++ RTTI (Run Time Type Information).
- The "System.Reflection" namespace contains classes and interfaces that provide a value of methods, fields and types.
- C# code for reflection uses the type of operator and GetTypes() method.
- There are four key Reflection techniques :
 1. Obtaining information about methods.
 2. Invoking methods.
 3. Constructing objects.
 4. Loading types from assemblies.

○ **Example :**

```
using System.Reflection ;

class MyClass
{
    public int Add(int a, int b)
    {
        return (a+b);
    }
}

class Demo
{

```

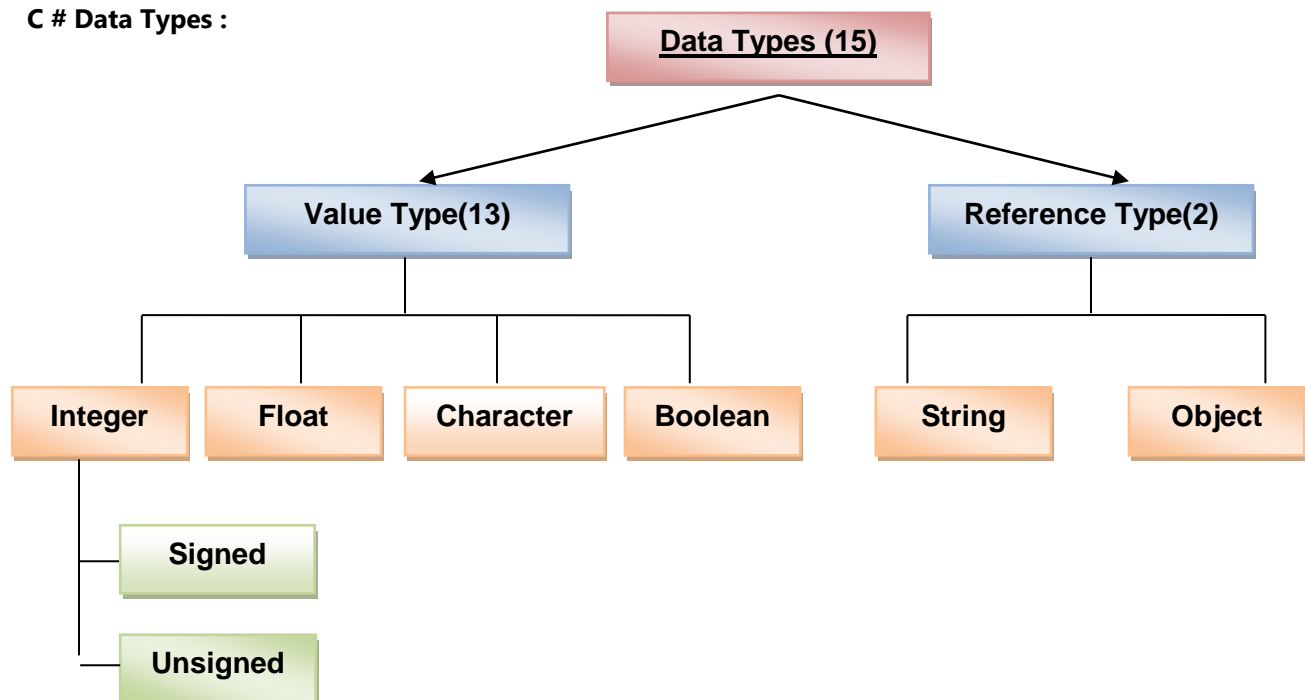
In above example GetType() is used to get the information of class type.

So, it is possible to check whether the class is an abstract class or regular class by :

➤ **Why we need Reflection (Use of Reflection) :**

- To create an instance of a class whose name is not known until the runtime.
- To invoke a method of an object even method is not known until the runtime.
- To create a new array whose size and component type is not known until the runtime.
- To define and execute dynamic method.
- So, Reflection is a powerful mechanism because it allows to learn and use the capability of types that are known only at runtime

➤ **C # Data Types :**



➤ **Signed Integers (Value Type):**

Name	Full Name	Size	Range
Sbyte	System.Sbyte	8-Bits	-2^7 to 2^7-1
Short	System.Int16	16-Bits	-2^{15} to $2^{15}-1$
Int	System.Int32	32-Bits	-2^{31} to $2^{31}-1$
Long	System.Int64	64-Bits	-2^{63} to $2^{63}-1$

➤ **Unsigned Integers (Value Type):**

Name	Full Name	Size	Range
Byte	System.Byte	8-Bits	0 to 2^8-1
Ushort	System.UInt16	16-Bits	0 to $2^{16}-1$
UInt	System.UInt32	32-Bits	0 to $2^{32}-1$
Ulong	System.UInt64	64-Bits	0 to $2^{64}-1$

➤ **Floating Point (Value Type):**

Name	Full Name	Size	Range
Float	System.Single	32-Bits	Single precision No.
Double	System.Double	64-Bits	Double precision No.
Decimal	System.Decimal	128-Bits	High precision No.

➤ **Character (Value Type):**

Name	Full Name	Size	Range
Char	System.Character	8-Bits	Any one character

➤ **Boolean (Value Type):**

Name	Full Name	Size	Range
Bool	System.Boolean	1-Bit	True(1) / False(0)

➤ **String (Reference Type):**

Name	Full Name	Size	Range
String	System.String	-	-

➤ **Object (Reference Type):**

Name	Full Name	Size	Range
Object	System.Object	-	-

➤ **Boxing and UnBoxing :**

- **Boxing** : when an object refers to a value type, a process is known as "boxing".
 - In simple words, casting of value type to object type is called "boxing".
 - No explicit type casting is required, it occurs automatically.
 - It causes the value of value type to be stored in object type.
 - Thus, value type is boxed inside the object.
- **UnBoxing** : the reverse process of "boxing" is called "unboxing".
 - In simple words, casting of object type to value type is called "unboxing".
 - This action is performed using explicit cast from object type to corresponding value type.
 - It is necessary to unbox object type to only that value type from which it was boxed.
 - Attempting to unbox an object into different type will result in runtime error
- Example of Boxing/UnBoxing is shown below.

class Demo

{

➤ **Operators in C# :**

○ **Arithmetic Operators :**

Operator	Meaning
+	Addition
-	Subtraction / Unary minus
*	Multiplication
/	Division
%	Modulus
++	Increment
--	Decrement

Relational and Logical Operators :

Operator	Meaning
==	Equal to
!=	Not Equal to
>	Greater than
<	Less than
>=	Greater than or Equal to
<=	Less than or Equal to

○ **Bitwise Operators :**

Operator	Meaning
&	Bitwise AND
	Bitwise OR
^	Bitwise exclusive OR (XOR)
>>	Shift right
<<	Shift left
~	One's complement (NOT)

➤ **Array in C# :**

One-Dimensional Arrays

Multi-Dimensional Arrays

- An array is a collection of variables of the same type that are referred to by a common name.
- In c#, array can have one or more dimensions.
- Arrays are used for variety of purposes because they offer a convenient means of grouping together related variables.
- The advantage of an array is that it organizes data in such a way that it can be easily manipulated.

➤ One-dimensional Array :

- A one-dimensional array is a list of related variables.
- Because arrays in c# are implemented as objects, two steps are needed to obtain an array for use it in program.
- First, declare a variable that can refer to an array.
- Second, create an instance of an array by use of "new".
- Syntax or general form to declare one-dimensional array is as follows :

type[] array-name = new type [size] ;

A	B	C	D	E	F	G	H	I	J
Index : [0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

➤ Multi-dimensional Array : If array has more than one row and column, it's called Multi Dimension Array. There are two kinds of Multi Dimension Array In C# :

- Rectangular Array
- Jagged Array
- A multi-dimensional array is an array that has two or more dimensions, and an individual element is accessed through the combination of two or more indices.
- In two-dimensional array the location of any specific element is specified by two indices (x,y).
- Two-dimensional array can be same as table, one index indicates the row, the other index indicates the column.
- Syntax or general form to declare two-dimensional array is as follows :

type[,] array-name = new type [row-size,column-size] ;

- To declare a two-dimensional array table of size 3, 5 :

int[,] table = new int [3,5] ;

	0	1	2	3	4
0	[0,0]	[0,1]	[0,2]	[0,3]	[0,4]
1	[1,0]	[1,1]	[1,2]	[1,3]	[1,4]
2	[2,0]	[2,1]	[2,2]	[2,3]	[2,4]

- :

Array-name[row-index,column-index] = value ;

➤ **Jagged Array :**

- C# allows creating special type of two-dimensional array called Jagged Array.
- Jagged array is an "array of array" in which the length of each array can differ.
- Thus, a jagged array can be used to create a table in which the lengths of the rows are not same.
- Jagged array are declared by using sets of square brackets to indicate each dimension.
- To declare two-dimensional jagged array :

type[][] array-name = new type [row-size][] ;

- The size indicates number of rows in the jagged array.
- Length of each row is vary :

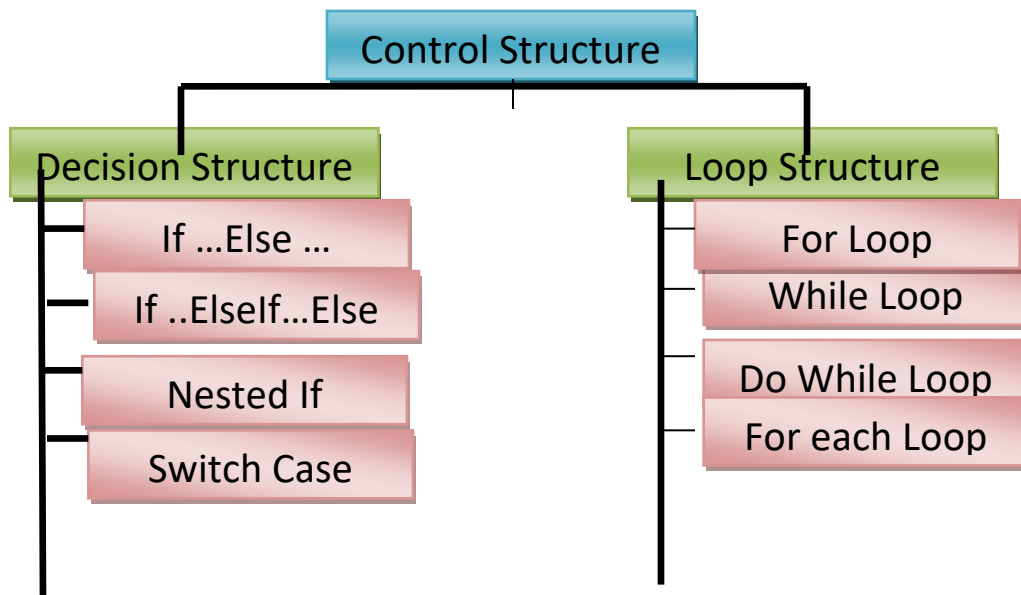
array-name[0] = new type [column-size] ;

- Example :

```
Int [ ][ ] table = new int [3][ ] ;  
table[0] = new int [3] ;  
table[1] = new int [2] ;  
table[2] = new int [4] ;
```

[1][0]	[1][1]		
[2][0]	[2][1]	[2][2]	[2][3]

➤ **Control Structure of C# :**



- **The if statement :**
 - If statement checks the condition, if the condition is true, the target statement of if will be executed, otherwise if exists, and the target statements of else will be executed.
 - At no time both of them will be executed.
 - Either the statements of "if" or "else" will be executed.
 - "else" clause is optional.

- The general form or syntax of "if" is as follows :

```

if (condition)
{
    statements;
}
else
{
    statements;
}

```

- The conditional expressions are evaluated from the top to downward.
- As soon as a true condition is found, the statement associated with it is executed and the rest of statements are bypassed.
- If none of the conditions is true, then the final "else" clause will be executed.
- The final "else" is like a default condition.
- If there is no final "else" and all other conditional tests fails, then no action will take place.
- The general form or syntax is as follows :

```

if (condition1)
{
    statement;
    .....
}
else if (condition2)
{
    statement;
    .....
}
else if (condition3)
{
    statement;
    .....
}
.....
.....
.....

```

○ **Nested ifs :**

- A nested if is an if statement that is the target of another if or else.
- In general nested if is a "if within if".
- The if which contains another if is called "Outer if".
- The if which is inside another if is called "Inner if".
- The general form or syntax is as follows :

```

if (condition)
{
    if (condition)
    {

```

- **The switch statement :**

- The switch case provides for a multiway branch.
- It enables a program to select among several alternatives.
- The value of an expression is successively tested against a list of constants.
- When match is found, the statement sequence associated with that match is executed.
- The general form or syntax is as follows :

```
switch (expression)
{
    case constant1 :
        Statements;
        break ;

    case constant2 :
        Statements;
        break ;

    case constant3 :
        Statements;
        break ;

    .....
    .....
    default :
        Statements;
        break ;
}
```

- The switch expression must be of an integer,char,byte,or string type.
- The "default case" is executed if no "case" constant matches the expression.
- The "default case" is optional.
- The statements associated with any case are executed until the break is encountered.

- **The for loop :**

- For loop is used to repeatedly execute a sequence of code.
- The general form or syntax is as follows :

```
for (initialization ; condition ; increment)
```

```
{
```

```
    Statements;
```

```
}
```

- The "initialization" portion of the for loop sets a loop control variable to an initial value.
- The "condition" is a Boolean expression that tests the loop control variable.
- If test is "true", the for loop continues to iterate.
- If it is "false", the loop terminates.
- The "increment" expression determines how the loop control variable is changed each time the loop iterates.

○ **The while loop :**

- Like other loop "while loop" is also used to repeatedly execute statements.
- The general form or syntax is as follows :

```
while (condition)
```

```
{
```

```
    Statements;
```

```
}
```

- Statement can be a single statement or a block of statements.
- Condition defines the condition that controls the loop and may be valid Boolean expression.
- The statement is executed while the condition is "true".
- When the condition becomes "false", program control passes to the line immediately following the loop.

○ **The do-while loop :**

- Unlike the for and while loops, in which the condition is tested at the top of the loop, the do-while loop checks its condition at the bottom of the loop.
- This means that a do-while loop will always execute at least once.
- The do-while loop executes as long as the conditional expression is true.

- The general form or syntax is as follows :

```
do  
  
{  
  
    Statements;  
  
}  
  
while (condition) ;
```

- A collection is a group of objects.
- C# defines several types of collections, of which one is an array.
- The general form or syntax of for each loop is as follows :

Foreach (type loopvariable in collection)

```
{  
  
    Statements;  
  
}
```

- The variable receives the value of next element in the collection each time the foreach loop iterates.
- When loop begins, the first element of the collection is obtained and assigned to loopvariable.
- Each subsequent iteration obtains the next element from the collection.
- The loop ends when there is no more elements to obtain.
- Thus, "foreach" loop cycles through the collection one element at a time, from start to finish.

➤ **Exception Handling in C# :**

- Exception means runtime errors.
- Exception handling is a technique to handle runtime errors without terminating execution of program.
- In c#, Exception Handling is managed via four keywords : try, catch, throw and finally.
- Program statements that are supposed to generate errors are contained within a try block.
- If an exception occurs within the try block, it is thrown.
- Exception is caught and handled by using catch block.
- System generated exceptions are automatically thrown by the c# runtime system.
- To manually throw an exception, the keyword "throw" is used.
- The general form of "try-catch" is shown below.


```

try
{
    //block of code to monitor for errors
}
catch (ExceptionType1 object)
{

```

```

try
{
    //block of code to monitor for errors
}
catch
{
    //handler for Exceptions
}

```

- Sometimes when a try/catch block is left and exception might terminates the current block.
- To specify a block of code to execute when an exception can not be caught by particular "catch" clause and exception must be handled, the "finally" is used.
- The general form of "try/catch" which includes "finally" is shown below

```

try
{
    //block of code to monitor for errors
}
catch
{
    //handler for Exceptions
}
...

```

- The nested try blocks means one try within another is also possible.

➤ **“throw” an exception manually :**

- It is also possible to throw an exception manually by using the “throw” statement.
- The general form is shown below :

```
try
{
    throw exception-object;
}
catch(exception-type)
{
    //handler for Exceptions }
```

- Commonly used exceptions are :
 - **DivideByZeroException**
 - **IndexOutOfRangeException**
 - **OverflowException**
 - **InvalidCastException** ,etc...

Chapter :-2 Class & Inheritance, Property, Indexer, Pointers, Delegates, Event, Collections

➤ **Class :**

- Classes are the basic ingredients of Object-Oriented languages.
- It is used to define custom data and methods.
- Classes can be declared by using the "class" keyword followed by the name of class and the brackets surrounding the body of the class.
- Supports member variables and methods.

➤ **Object :**

- An object is a variable or an instance of the type Class.
- It can access all the members or properties of class by using dot (.) Operator.

➤ **Structure :**

- It is the user defined datatype provided by c#.
- Like classes, structures can also contain both data and method definitions.
- Also like a class, a structure can contain constructors, constants, fields, methods, properties, indexers, operators and nested types.

➤ **Namespace :**

- Namespace is used to organize classes and other types into a single hierarchical structure.
- Namespace is the logical collection of related classes.
- The proper use of namespaces will make classes easy to use and prevent collisions with classes written by other authors.
- The namespace can contain classes and other namespaces.
- In C# "System" is the root namespace.
- All the data related classes are within System.Data namespace.
- All Input/Output related classes are within System.IO namespace, etc.

➤ **"using" Keyword :**

- To use the classes within the particular namespace, the namespace must be inherited before the use.
- The namespace is inherited in C# program by using "using" keyword.
- EX : using System.IO;

➤ **Encapsulation** means that a group of related properties, methods, and other members are treated as a single unit or object..

➤ **Polymorphism** means that you can have multiple classes that can be used interchangeably, even though each class implements the same properties or methods in different ways.

➤ **Properties & Fields:**

- Fields and properties represent information that an object contains. Fields are like variables because they can be read or set directly.

- Both C# and Visual Basic allow you either to create a private field for storing the property value or use so-called auto-implemented properties that create this field automatically behind the scenes and provide the basic logic for the property procedures.

➤ **Methods & Events:**

- A method is an action that an object can perform. A class can have several implementations, or overloads, of the same method that differ in the number of parameters or parameter types.
- Events enable a class or object to notify other classes or objects when something of interest occurs. The class that sends (or raises) the event is called the publisher and the classes that receive (or handle) the event are called subscribers. For more information about events, how they are raised and handled.
- To declare an event in a class, use the [event \(C# Reference\)](#) keyword.
- To raise an event, invoke the event delegate.
- To subscribe to an event, use the += operator; to unsubscribe from an event, use the -= operator.

➤ **Methods with “ref” and “out” parameters:**

Ref

The ref keyword is used to pass an argument as a reference. This means that when value of that parameter is changed in the method, it gets reflected in the calling method. An argument that is passed using a ref keyword must be initialized in the calling method before it is passed to the called method.

Out

The out keyword is also used to pass an argument like ref keyword, but the argument can be passed without assigning any value to it. An argument that is passed using an out keyword must be initialized in the called method before it returns back to calling method.

Example:

```
public class Example
{
    public static void Main() //calling method
    {
        int val1 = 0; //must be initialized
        int val2; //optional

        Example1(ref val1);
        Console.WriteLine(val1); // val1=1

        Example2(out val2);
        Console.WriteLine(val2); // val2=2
    }

    static void Example1(ref int value) //called method
    {
        value = 1;
    }
    static void Example2(out int value) //called method
    {
        value = 2; //must be initialized
    }
}
/* Output
```

➤ **Static and Non-Static Members:**

Static Members	Non-Static Members
1) Static members are one per class	1) Non-Static members are one per instance

2) Static members are accessed by class name which encapsulates	2) Non-Static members are accessed by object reference
3) Static members can not use non static methods without instantiating an object	3) Non-Static members can use static members directly.

➤ **Constructor:**

- Constructors are class methods that are executed automatically when an object of a given type is created. Constructors usually initialize the data members of the new object. A constructor can run only once when a class is created. Furthermore, the code in the constructor always runs before an other code in a class. However, you can create multiple constructor overloads in the same way as for any other method.

```
public class SampleClass
{
    public SampleClass()
    {
        // Add code here
    }
}
```

➤ **Overloading Constructor:**

Constructor overloading in c# is one type of static polymorphism. Using constructor overloading, any number of constructors can be defined for same class. But ensure each constructor must have different number and type of parameters defined.

```
using System;
namespace ConsoleApplication3
{
    class Sample
    {
        public string param1, param2;

        public Sample() // Default Constructor
        {
            param1 = "Hi";
            param2 = "I am Default Constructor";
        }

        public Sample(string x, string y) // Declaring Parameterized constructor with
Parameters
        {
            param1 = x;
            param2 = y;
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            Sample obj = new Sample(); // Default Constructor will Called
        }
    }
}
```

```

        Sample obj1=new Sample("Welcome","Aspdotnet-Suresh"); // Parameterized
        Constructor will Called
        Console.WriteLine(obj.param1 + " , "+obj.param2);
        Console.WriteLine(obj1.param1 +" to " + obj1.param2);
        Console.ReadLine();
    }
}

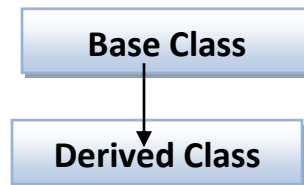
```

➤ **Inheritance in C# :**

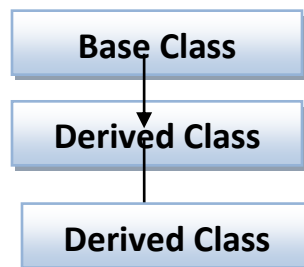
- Inheritance is the process by which one object can acquire the properties of another object.
- Inheritance is one of the principle of object-oriented programming supports the concept of hierarchical(top-down) classification.
- Using inheritance the object can inherits its general attributes from its parent.
- It is the concept of reusing something that is already exists rather than to create the same again and again.
- The class that is inherited is called "base class" or "parent class" and the class that does the inheriting is called "derieved class" or "child class".

➤ **Types of Inheritance :**

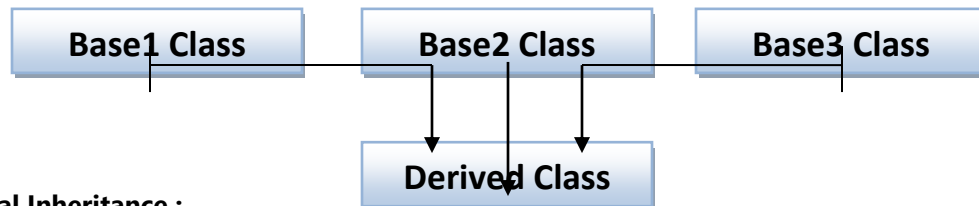
(1) **Single Inheritance :**



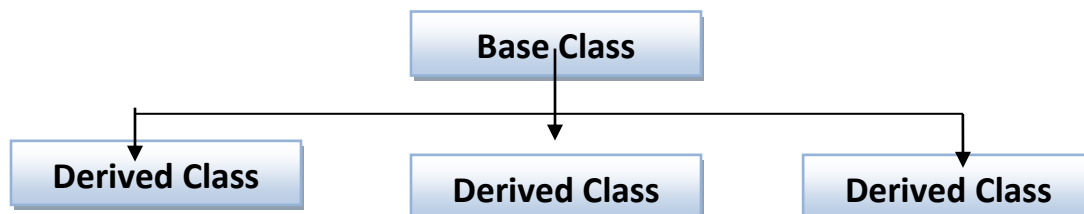
(2) **Multi-Level Inheritance :**



(3) **Multiple Inheritance :**

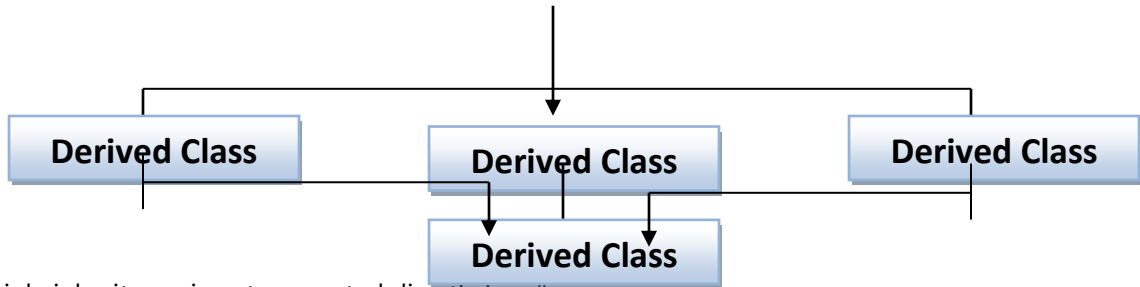


(4) **Hierarchical Inheritance :**



(5) **Hybrid Inheritance :**





- Multiple inheritance is not supported directly by c#.
 - The general form of inheritance is shown below.
 - In following example Class A is a "base" class and class B is "derived" class.
 - The object of Class B has accessibility of members(variables and methods) of Both Class A and Class B.

```

class A
{
    Variables and Methods of Class A;
}

class B : A
{
    Variables and Methods of Class B;
}
  
```

- **Sealed class :**
 - It is a powerful concept used when you want to prevent the concept of "Inheritance".
 - If you want to create a class which will never inherited by any derived class then the class must be declared as "Sealed" class.
 - To prevent a class from being inherited, precede it declaration with "sealed" keyword.
 - Sealed class is totally opposite of "Abstract" class.
 - So it is illegal to declare a class as both "sealed" and "abstract".
 - Sealed can also be used on virtual methods to prevent further overrides.
 - The example of sealed class is -

```

Sealed class A
{
    Variables and Methods of Class A;
}
  
```

- Now, inheriting above class A will causes the error.

➤ **Abstract class :**

- The [abstract](#) keyword enables you to create classes and [class](#) members that are incomplete and must be implemented in a derived class
- Classes can be declared as abstract by putting the keyword abstract before the class definition. For example:

```
public abstract class A
{
    // Class members here.
}
```

- An abstract class cannot be instantiated. The purpose of an abstract class is to provide a common definition of a base class that multiple derived classes can share.
- For example, a class library may define an abstract class that is used as a parameter to many of its functions, and require programmers using that library to provide their own implementation of the class by creating a derived class.
- Abstract classes may also define abstract methods. This is accomplished by adding the keyword abstract before the return type of the method.
- For example: abstract methods have no implementation, so the method definition is followed by a semicolon instead of a normal method block.
- Derived classes of the abstract class must implement all abstract methods. When an abstract class inherits a virtual method from a base class, the abstract class can override the virtual method with an abstract method. For example:

```
public class D
{
    public virtual void DoWork(int i)
    {
        // Original implementation.
    }
}
```

```
public abstract class E : D
{
    public abstract override void DoWork(int i);
}
```

```
public class F : E
{
    public override void DoWork(int i)
    {
        // New implementation.
    }
}
```

➤ **Overriding Methods:**

- Creating a method in derived class with same signature as a method in base class is called as method overriding.
- Same signature means methods must have same name, same number of arguments and same type of arguments.

- Method overriding is possible only in derived classes, but not within the same class.
- When derived class needs a method with same signature as in base class, but wants to execute different code than provided by base class then method overriding will be used.
- To allow the derived class to override a method of the base class, C# provides two options, **virtual** methods and **abstract** methods.

Examples for Method Overriding in C#

```
using System;
namespace methodoverriding
{
    class BaseClass
    {
        public virtual string YourCity()
        {
            return "New York";
        }
    }

    class DerivedClass : BaseClass
    {
        public override string YourCity()
        {
            return "London";
        }
    }

    class Program
    {

        static void Main(string[] args)
        {
            DerivedClass obj = new DerivedClass();
            string city = obj.YourCity();
            Console.WriteLine(city);
            Console.Read();
        }
    }
}
```

- Output

London

➤ **Overloading Methods :**

- Overloading is the concept of polymorphism.
- In c#, two or more methods within the same class can share the same name, as long as their parameter declarations are different.
- C# supports several versions of the method that have different signatures(argument list and return type).
- They must differ in their type and number of parameters.
- When the method is called by its name, the method is called whose return type and parameter list is matched.
- The example of method overloading is shown below.

```

class MyClass
{
    int add(int x, int y)
    {
        return (x+y);
    }
    float add(float x, float y)
    {
        return (x+y);
    }
}
class Demo
{
    static void Main( )
    {
        MyClass obj=new MyClass( );

        int result = obj.add(10,20); // calling first method

        float answer = obj.add(10.5,20.5); // calling second method

    }
}

```

➤ **Ove**

-
-
-

action of the operator.

- When operator is overloaded, none of its original meaning is lost.
- There are two forms of operator methods :
 1. For "unary" operators.
 2. For "binary" operators.

➤ **Overloading "unary" operator :**

- Operators which take single Operand are known as "unary" operators.
- Ex. Unary + (++ Increment), Unary Minus (- - Decrement), etc.
- The general form for unary operator overloading is shown below :

defines the

```
public static ret-type operator op(type operand)
```

```
{
```

```
    // operations
```

```
}
```

- The "ret-type" in above syntax specifies the type of value returned by operation.
- "ret-type" can be any type, but it is often of the same type as the class for which the operator is being overloaded.
- For unary operator, the operand is passed in "operand".
- Operand must be of the same type of class.
- Note : operator methods must be both "public" and "static".

➤ **Overloading "binary" operator :**

- Operators which take two Operands are known as "binary" operators.
- Ex. Binary + (Addition), Binary Minus (Subtraction), etc.
- The general form for binary operator overloading is shown below :

```
public static ret-type operator op(type1 operand1,type2 operand2)
```

```
{
```

```
    // operations
```

```
}
```

- For binary operator, the operands are passed in "operand1" and "operand2".
- Atleast one of the Operands must be of the same type of its class.
- Note : operator methods must be both "public" and "static".

Properties,Events and Methods in C# :

- It is a one type of class member.
- Property combines the field (variable) with the methods that access it.
- A property consists of a name along with get and set accessors.
- The accessors are used to get and set the value of variable.
- Accessors are similar to the method except that it does not declare a return type or parameters.
- The key benefit of property is that its name can be used in expressions and assignments like a normal variable, and get and set are automatically invoked.
- General form of property is :

```
type name
```

```
{
```

➤ **Indexers :**

- An indexer allows an object to be indexed like an array.
- The main use of indexers is to support the creation of specialized arrays supports one or more constraints.
- Indexers can have one or more dimensions.

➤ **Creating One dimensional Indexers :**

- The general form of One-dimensional indexer is :

```
element-type this[int index]
{
    get
    {
        // return the value specified by index
    }
    set
    {
        // set the value specified by index
    }
}
```

- The "element-type" is the datatype of element of the indexer.
- So, each element accessed by the indexer will be of the type element-type of an array.
- The parameter "index" receives the index of the element being accessed.
- Because of index of an array is generally of type integer, the parameter "index" is of type "int".

- Inside the body of indexer two accessors "get" and "set" are defined.
- Accessors are similar to the method except that it does not declare a return type or parameters.
- The Accessors are automatically called when the indexer is used, and both the accessors receive index as a parameter.
- If the indexer is on the left side of an assignment statement, then "set" accessor is called and the element specified by the index must be set by the implicit parameter called "value".
- Otherwise, the "get" accessor is called and the value associated with index must be returned.

➤ **Creating Multi dimensional Indexers :**

- It is also possible to create indexer for multi-dimensional array.
- The general form of One-dimensional indexer is :

```
element-type this[int index1,int index2]
{
    get
    {
        // return the value specified by [index1,index2]
    }
    set
    {
        // set the value specified by [index1,index2]
    }
}
```

➤ **Overloading Indexers :**

- An indexer can also be overloaded.
- The version of an indexer is invoked which matches the number of parameter and type.

```
element-type this[int index]
{
    get {
        // get accessor code }
    set {
        // set accessor code }
    }

element-type this[double index]
{
    get {
```

Creating and Using Pointers (Unsafe Concept)

- A **pointer** is a variable whose value is the address of another variable i.e., the direct address of the memory location. Like any variable or constant, you must declare a pointer before you can use it to store any variable address.
- Unsafe keyword denotes an unsafe context, which is required for any operation involving pointers. You can use unsafe modifier in the declaration of a type or a member . The entire textual extent of the type or member is therefore considered an unsafe context.
- The general form of a pointer variable declaration is:
`type *var-name;`
- Following are valid pointer declarations:
`int *ip; /* pointer to an integer */
double *dp; /* pointer to a double */
float *fp; /* pointer to a float */
char *ch /* pointer to a character */`
- The following example illustrates use of pointers in C#, using the unsafe modifier:

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

Output:
Data is: 20
Address is: 99215364

- **Creating and using Delegates (Single / Multicasting):-**

- In C#, delegates represent methods that are callable without knowledge of the target object. Delegates enable scenarios that some other languages have addressed with function pointers. However, unlike function pointers, delegates are object-oriented and type-safe.
- There are three steps in defining and using delegates: declaration, instantiation, and invocation.

➤ **Delegate Declaration:**

```
public class DelegateClass
{
    public DelegateClass()
    {
    }
}

// Declare a delegate
public delegate void MessageHandler(string message);
// The use of the delegate.

public void Process(MessageHandler handler)
{
    if (handler != null)
        handler("Begin Message");
    if (handler != null)
        handler("End Message");
}
}
```

Function Class:

This class contains the functions will be called by delegates

```
public class FunctionClass
{
    public FunctionClass()
    {
    }
}

//This method will show alert message to user
public static void AlertMessage(string s)
{
    System.Windows.Forms.MessageBox.Show(s);
}

//This method will write output to console
public static void ConsoleMessage(string s)
{
    Console.WriteLine(s);
}
}
```

Instantiation and Invocation:

Client Application to use the defined Delegate

Simple Delegate

Create a windows form and add one button to it and paste the following code on its click event.

```
private void button1_Click(object sender, System.EventArgs e)
{
    //Instantiating a delegate
    DelegateClass dc = new DelegateClass();
    DelegateClass.MessageHandler ll = null;
    ll += new DelegateClass.MessageHandler(FunctionClass.AlertMessage);

    //Calling a delegate
    dc.Process(ll);
}
```

When you will click on button1, it will show 'Begin Message' and 'End Message' message box.

Multicast Delegate

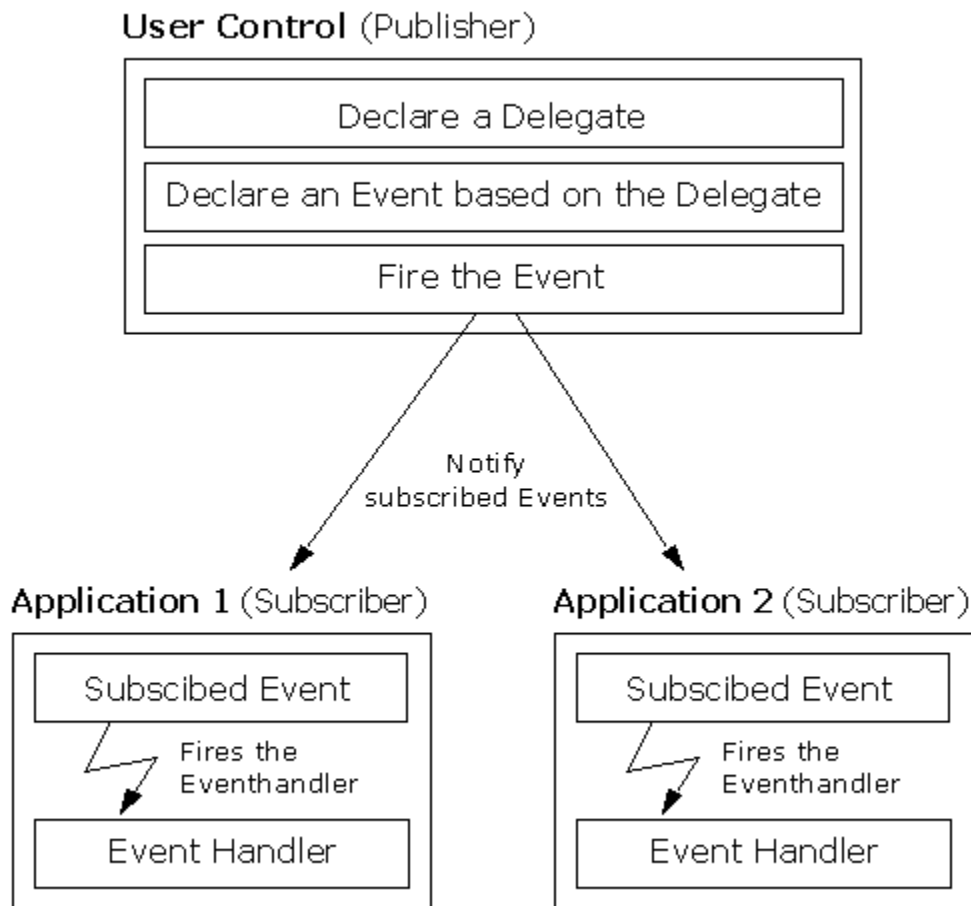
Add one more button to form with following code.

```
private void button2_Click(object sender, System.EventArgs e)
{
    DelegateClass dc = new DelegateClass();
    DelegateClass.MessageHandler ll = null;
    ll += new DelegateClass.MessageHandler(FunctionClass.AlertMessage);
    ll += new
    DelegateClass.MessageHandler(FunctionClass.ConsoleMessage);
    dc.Process(ll);
}
```

➤ Creating and using Events with Event Delegate:-

- ❖ A delegate in C# is similar to a function pointer in C or C++. Using a delegate allows the programmer to encapsulate a reference to a method inside a delegate object. The delegate object can then be passed to code which can call the referenced method, without having to know at compile time which method will be invoked.
- ❖ The Event model in C# finds its roots in the event programming model that is popular in asynchronous programming. The basic foundation behind this programming model is the idea of "publisher and subscribers."
- ❖ In this model, you have ***publishers*** who will do some logic and publish an "event." Publishers will then send out their event only to ***subscribers*** who have subscribed to receive the specific event.

- ❖ In C#, any object can *publish* a set of events to which other applications can *subscribe*. When the publishing class raises an event, all the subscribed applications are notified. The following figure shows this mechanism.



Conventions

- The following important conventions are used with events:
 - Event Handlers in the .NET Framework return void and take two parameters.
 - The first parameter is the source of the event; that is the publishing object.
 - The second parameter is an object derived from EventArgs.
 - Events are properties of the class publishing the event.
 - The keyword event controls how the event property is accessed by the subscribing classes.

➤ Collections (ArrayList, Hashtable, Stack, Queue, SortedList) and their differences.

- ❖ **ArrayList** - automatically growing array. Adds more overhead. Can enum., probably slower than a normal array but still pretty fast. These are used a lot in .NET
- ❖ **Hashtable** - plain old hashtable. O(1) to O(n) worst case. Can enumerate the value and keys properties, and do key/val pairs.
- ❖ **Stack**. We push and pop elements onto the top of a Stack. With this functionality, we implement certain kinds of parsers. An array would work, but the Stack has a clearer interface.
- ❖ **Queue**. This removes elements that were added first (FIFO, first-in-first-out). So we keep items in order and service the ones that were added first, like a line at a government agency.

- ❖ **SortedList** - a sorted generic list. Slowed on insertion since it has to figure out where to put things. Can enum., probably the same on retrieval since it doesn't have to resort, but deletion will be slower than a plain old list.

Windows Forms :

- In C# .Net it's these Forms with which we work. They are the base on which we build/ develop all our user interface and they come with a rich set of classes.
- Forms allow us to work visually with controls and other items from the toolbox.
- In C# .NET forms are based on the System.Windows.Forms namespace and the form class is System.Windows.Forms.Form.
- The form class is based on the Control class which allows it to share many properties and methods with other controls.
- Forms can be standard windows, multiple document interface (MDI) windows, dialog boxes, or display surfaces for graphical routines. The easiest way to define the user interface for a form is to place controls on its surface.
- Forms are objects that expose properties which define their appearance, methods which define their behavior, and events which define their interaction with the user.
- By setting the properties of the form and writing code to respond to its events, you customize the object to meet the requirements of your application.

MessageBox() :

- It will display message or string in another window.
- The following argument of messagebox can be given in the MsgBox().

Properties Of MsgBox :

Name	Description
Text	It will display text given in this argument on the messagebox.
Caption	The text in this argument is display in the titlebar of the MsgBox.
Button	The argument values will be buttons to be displayed in the MsgBox.
Icon	The icon name even in the argument is displayed on the msgbox.
DefaultButton	The value set will specify each default button is to be display in the msgbox.

- Button property can have following values:

1. **AbortRetryIgnore**
2. **Ok**
3. **OkCancel**
4. **RetryCancel**
5. **YesNo**
6. **YesNoCancel**

- Icon can have following values:

1. **warning**

4. **Stop**

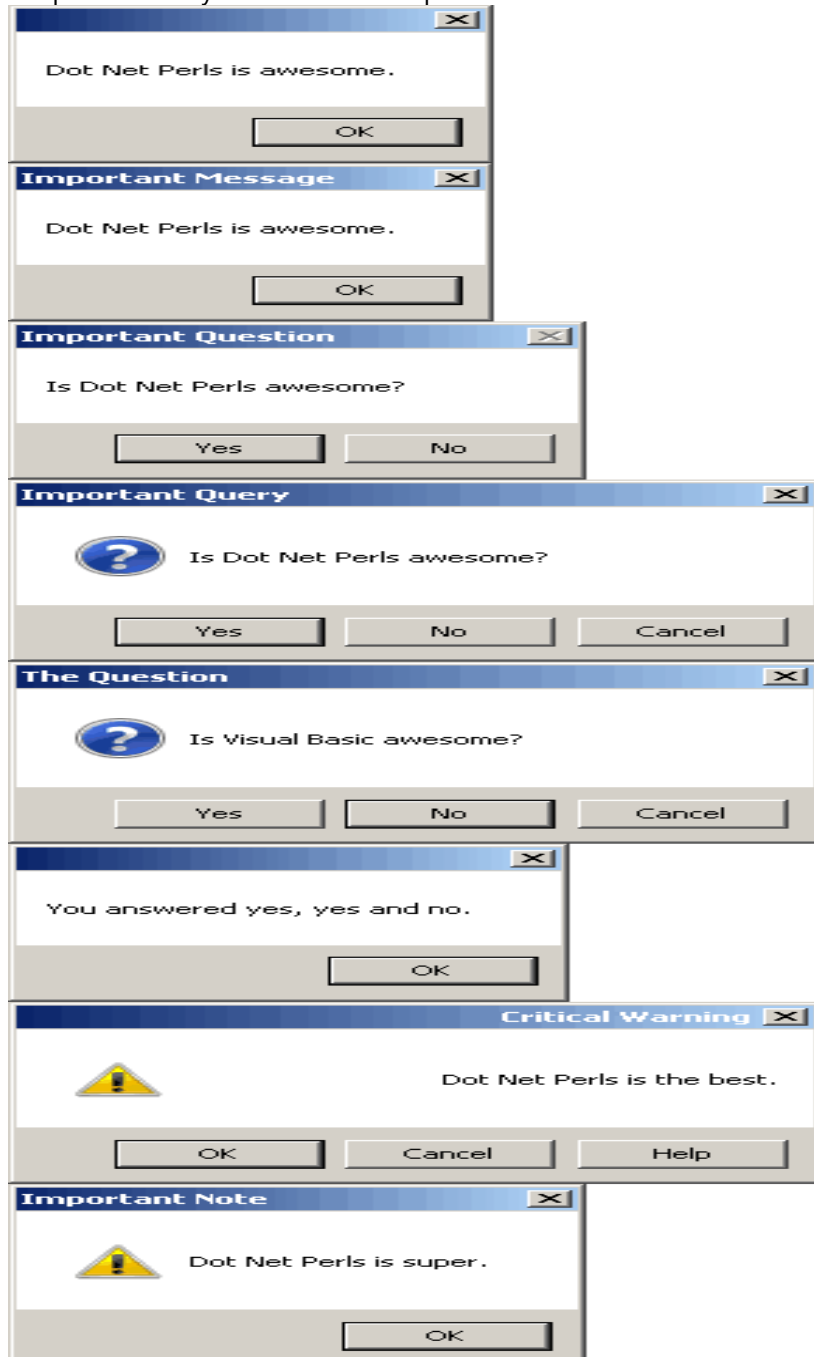
7. **Question**

- | | | |
|----------------|----------|----------------|
| 2. None | 5. Error | 8. Astrik |
| 3. Exclamation | 6. Hand | 9. Information |

o Default Button can have following values:

- | | |
|-----------------------------|-------------------------------|
| o DefaultDesktopOnly | 3. RTLReading |
| o RightAlign | 4. ServiceNotification |

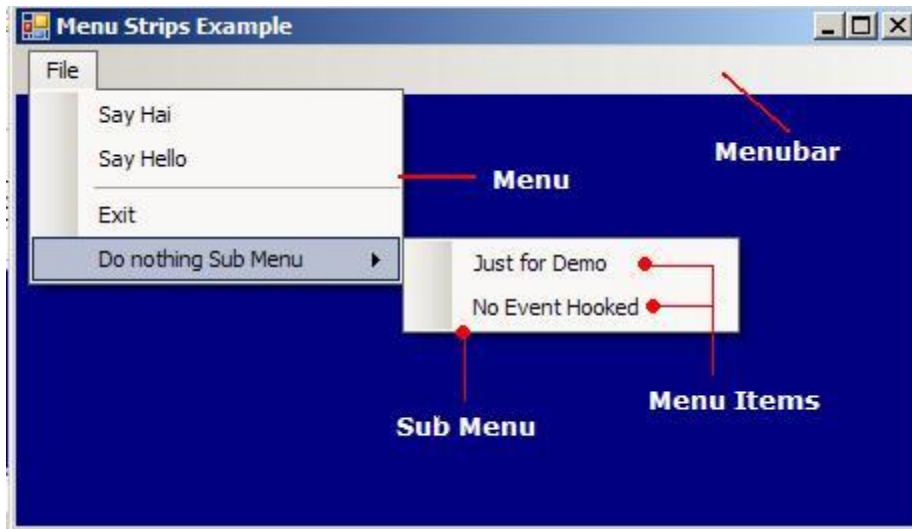
➤ Dialog boxes interrupt users. They force users to respond before further action is taken. This is necessary in



some situations.

➤ **Menu (MenuStrip, ContextMenuStrip) :**

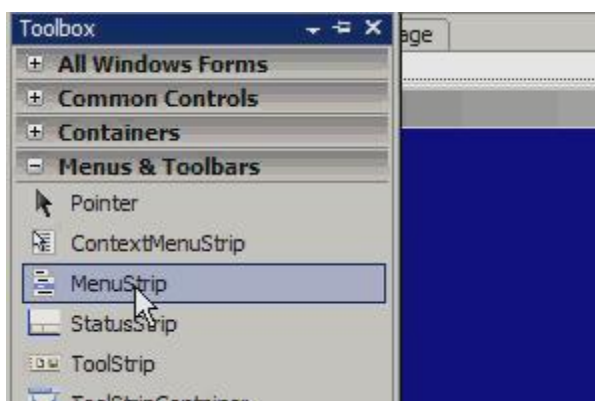
- ❖ Strip is a relatively narrow piece of something. Dot net has three important strip controls namely **MenuStrip**, **StatusStrip** and **ToolStrip**. In this article we will start with the **menustrip** control.
- ❖ All the strips controls accommodate some other UI elements in it. A menu strips allows you to add Menu and Menu allows you to add menu items. Similarly the toolstrip control allows you add one or more tool bar buttons in it. OK, let us go to the MenuStrip. Look at the below screen shot:



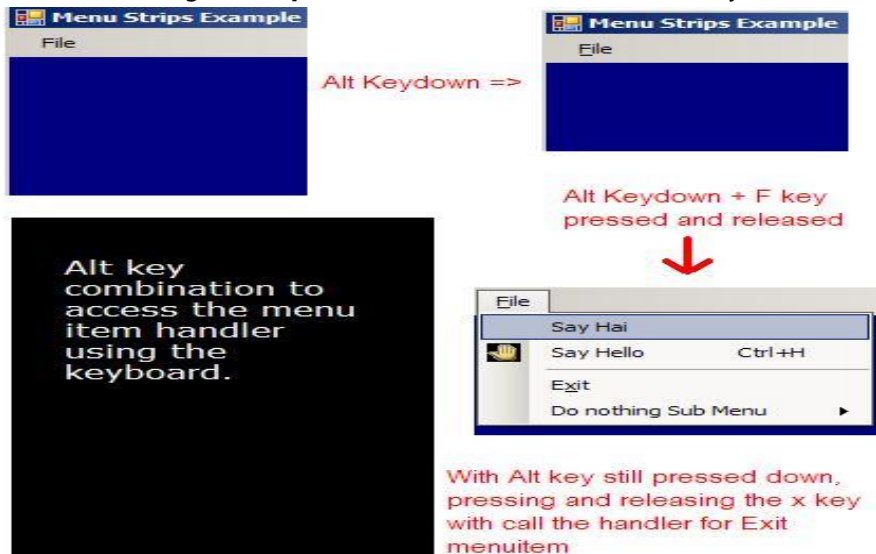
- ❖ A Menubar, dot net calls it Menu strip can accommodate multiple **Menus**. In the above picture only File menu is added to it. **Menu items** are usually added to the menu. Say for example in the above picture, the **Exit** and **Just for Demo** are menu items. Menu items are usually linked to the command handler and when you click the menu item the corresponding handler gets called.
- ❖ If a menu item is linked with one more menu, then the resulting menu is called as Sub-Menu. The above picture shows one such sub-menu with two menu items in it.
- ❖ In this example, we will create the above shown form with menu and menu items. Then we will provide some simple handler for it.

2. Adding menu Strips control

- ❖ The menu strip is grouped in the toolbox under the Menus & Toolbars group. This is shown in the below picture. To use menustrip in the form drag the menustrip from the toolbox and drop it to the form.



- ❖ In the below video, a menu strip is added to the form. To add a **separator** in the menu just type the **hyphen** (-) for the menu name.
- ❖ The placement **ampersand** (&) in the menu item name displays underline for letter next to it.
- ❖ This under line is displayed at runtime when alt key is pressed and typing the letter with underline (alt key is still kept down) will call the menu item handler.
- ❖ The usage of **ampersand** in the menu item with alt key is shown below:

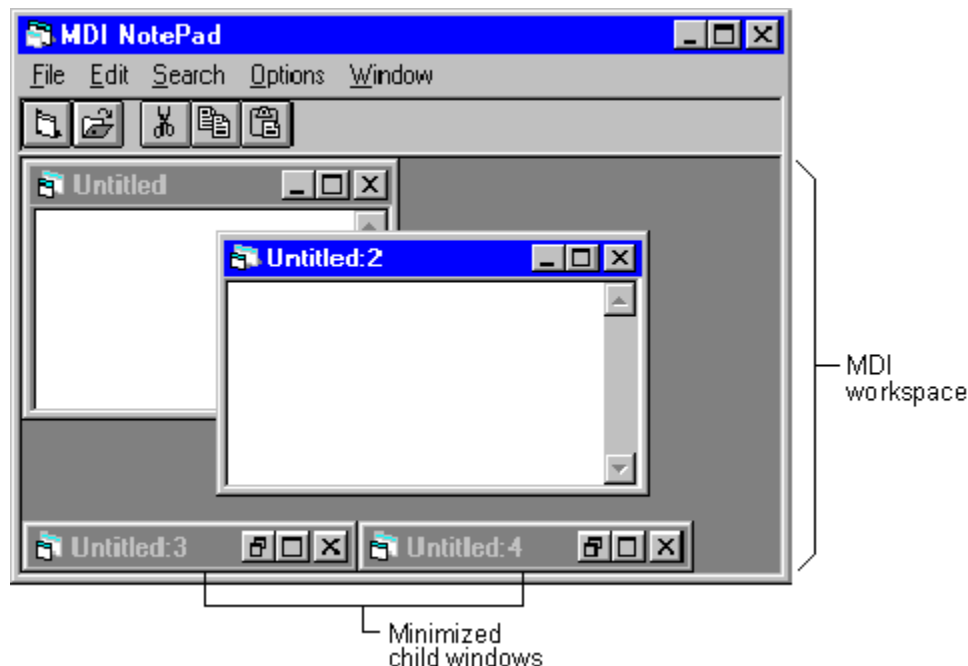


- ❖ So in the above picture in place of clicking the Exit menu item, you can call the handler by using keys **Alt+F,X**

➤ MDI Concept with MDI Notepad

- ❖ The multiple-document interface (MDI) allows you to create an application that maintains multiple forms within a single container form. Applications such as Microsoft Excel and Microsoft Word for Windows have multiple-document interfaces.
- ❖ An MDI application allows the user to display multiple documents at the same time, with each document displayed in its own window. Documents or *child windows* are contained in a *parent window*, which provides a workspace for all the child windows in the application. For example, Microsoft Excel allows you to create and display multiple-document windows of different types. Each individual window is confined to the area of the Excel parent window. When you minimize Excel, all of the document windows are minimized as well; only the parent window's icon appears in the task bar.
- ❖ A child form is an ordinary form that has its MDIChild property set to True. Your application can include many MDI child forms of similar or different types.
- ❖ At run time, child forms are displayed within the *workspace* of the MDI parent form (the area inside the form's borders and below the title and menu bars). When a child form is minimized, its icon appears within the workspace of the MDI form instead of on the taskbar, as shown in Figure 6.4.

Figure 6.4 Child forms displayed within the workspace of the MDI form



Note Your application can also include standard, non-MDI forms that are not contained in the MDI form. A typical use of a standard form in an MDI application is to display a modal dialog box.

- ❖ An MDI form is similar to an ordinary form with one restriction. You can't place a control directly on a MDI form unless that control has an `Align` property (such as a picture box control) or has no visible interface (such as a timer control).

Creating an MDI Application

Use the following procedure to create an MDI form and its child forms.

To create an MDI application

1. Create an MDI form.
From the **Project** menu, choose **Add MDI Form**.
Note An application can have only one MDI form. If a project already has an MDI form, the Add MDI Form command on the Project menu is unavailable.
2. Create the application's child forms.
To create an MDI child form, create a new form (or open an existing one) and set its `MDIChild` property to `True`.

➤ **Concept of Inheriting Form:-**

- ❖ Creating new Windows Forms by inheriting from base forms is a handy way to duplicate your best efforts without going through the process of entirely recreating a form every time you require it.
- ❖ For more information about inheriting forms at design time using the Inheritance Picker dialog box and how to visually distinguish between security levels of inherited controls, see [How to: Inherit Forms Using the Inheritance Picker Dialog Box](#).
- ❖ **Note** In order to inherit from a form, the file or namespace containing that form must have been built into an executable file or DLL. To build the project, choose Build from the Build menu. Also, a reference to the namespace must be added to the class inheriting the form. The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose Import and Export Settings on the Tools menu. For more information, see [Customizing Development Settings](#).

To inherit a form programmatically

1. In your class, add a reference to the namespace containing the form you wish to inherit from.
2. In the class definition, add a reference to the form to inherit from. The reference should include the namespace that contains the form, followed by a period, then the name of the base form itself.

➤ **Controls used in Form :**

- Controls are an extremely important part of any interactive application.
- They give information to the user (Label, ToolTip, TreeView, PictureBox, etc.) and organize the information so that it's easier to understand (GroupBox, Panel, TabControl).
- They enable the user :
 - To enter data (TextBox, RichTextBox, ComboBox, MonthCalendar)
 - To select options (RadioButton, CheckBox, ListBox)
 - To control the application (Button, MenuStrip, ContextMenuStrip)
 - To interact with objects outside of the application (OpenFileDialog, SaveFileDialog, PrintDocument, PrintPreviewDialog)
 - Some controls also provide support for other controls (ImageList, ToolTip, ContextMenuStrip, and ErrorProvider).

Before going for specific control, here is the list of some common Properties, Methods and Events.

○ **Common Properties :**

Name	Description
ContextMenuStrip	Gets or sets the ContextMenuStrip associated with this control.
Dock	Gets or sets which control borders are docked to its parent control and determines control is resized with its parent.
Enabled	Gets or sets a value indicating whether the control can respond to user interaction.
Font	Gets or sets the font of the text displayed by the control.
Name	Gets or sets the name of the control.
TabIndex	Gets or sets the tab order of the control within its container.
TabStop	Gets or sets a value indicating whether the user can give the focus to this control the TAB key.
Visible	Gets or sets a value indicating whether the control and all its child controls are displayed.
Size	Gets or sets the height and width of the control.

○ **Common Methods :**

Name	Description
Focus()	Sets input focus to the control.

○ **Common Events :**

Name	Description
FontChanged	Occurs when the Font property value changes.
GotFocus	Occurs when the control receives focus.
LostFocus	Occurs when the control loses focus.
MouseDown	Occurs when the mouse pointer is over the control and a mouse button is pressed.
MouseEnter	Occurs when the mouse pointer enters the control.

MouseMove	Occurs when the mouse pointer is moved over the control.
MouseUp	Occurs when the mouse pointer is over the control and a mouse button is released.
MouseWheel	Occurs when the mouse wheel moves while the control has focus.
Resize	Occurs when the control is resized.

➤ **Basic GUI Controls :**

➤ **Label :**

- You use labels for just what they sound like—to label other parts of your application. Labels usually are used to display text that cannot be edited by the user. Your code can change the text displayed by a label.
- The caption for a label is stored in the **Text** property. Because you can change that caption in code, labels can act a little like non-editable text boxes, displaying text and messages to the user. The **TextAlign** (formerly **Alignment**) property allows you to set the alignment of the text within the label.

- **Properties of Label :**

Name	Description
AutoSize	Gets or sets a value indicating whether the control is automatically resized to display its contents.
Left	Gets or sets the distance, in pixels, between the left edge of the control and the left edge of its container's client area.
Text	Gets or sets the text associated with this control.
TextAlign	Gets or sets the alignment of text in the label.

- **Methods of Label :**

ResetText()	Resets the Text property to its default value.
---------------------	--

- **Events of Label :**

Name	Description
AutoSizeChanged	Occurs when the value of the AutoSize property changes.
VisibleChanged	Occurs when the Visible property value changes.

➤ **TextBox :**

- Windows users should be familiar with textboxes. This control looks like a box and accepts input from the user.
- The TextBox is based on the TextBoxBase class which is based on the Control class.
- TextBoxes are used to accept input from the user or used to display text.
- By default we can enter up to 2048 characters in a TextBox but if the Multiline property is set to True we can enter up to 32KB of text.
- The TextBox control allows the user to enter text in an application. This control has additional functionality that is not found in the standard Windows text box control, including multiline editing and password character masking.

- **Properties of TextBox :**

Name	Description
MaxLength	Gets or sets the maximum number of characters the user can type or paste into the text box control.
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.
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.
Text	Gets or sets the current text in the TextBox.
TextAlign	Gets or sets how text is aligned in a TextBox control.
WordWrap	Indicates whether a multiline text box control automatically wraps words to the beginning of the next line when necessary.
SelectedText	Gets or sets a value indicating the currently selected text in the control.
SelectionLength	Gets or sets the number of characters selected in the text box.

○ **Methods of TextBox :**

Name	Description
Clear()	Clears all text from the text box control.
Copy()	Copies the current selection in the text box to the Clipboard.
Cut()	Moves the current selection in the text box to the Clipboard.
Paste()	Replaces the current selection in the text box with the contents of the Clipboard.
SelectAll()	Selects all text in the text box.
DeselectAll()	Specifies that the value of the SelectionLength property is zero so that no characters are selected in the control.
Undo()	Undoes the last edit operation in the text box.

○ **Event of TextBox :**

Name	Description
KeyDown	Occurs when a key is pressed while the control has focus.
KeyPress	Occurs when a key is pressed while the control has focus.
KeyUp	Occurs when a key is released while the control has focus.
MultilineChanged	Occurs when the value of the Multiline property has changed.
TextChanged	Occurs when the Text property value changes.

➤ **RichTextBox :**

- The RichTextBox control is one of the most powerful controls provided in C#.Net.
- In a nutshell, it's a text box that's able to display text stored in Rich Text Format (RTF), a standard format recognized by virtually all word processors, including Microsoft WordPad (not surprisingly, since WordPad internally uses the RichTextBox control).
- This control supports multiple fonts and colors, left and right margins, bulleted lists, and more.

- The RichTextBox control is data-aware and therefore exposes the usual Dataxxxx properties that let you bind the control to a data source. In other words, you can write entire TXT or RTF documents in a single field of a database.
- **Changing character attributes :**
- The RichTextBox control exposes many properties that affect the attributes of the characters in the selected text: These are FontName, FontSize, Color, Bold, Italic, Underline, etc.
- Their names are self-explanatory, so won't describe what each one does. You might find it interesting to note that all of the properties work as they would within a regular word processor.
- It supports the properties, events, and methods of TextBox control.

➤ **RadioButton :**

- The RadioButton control can display text, an Image, or both.
- When the user selects one option button (also known as a radio button) within a group, the others clear automatically. All RadioButton controls in a given container, such as a Form, constitute a group.
- To create multiple groups on one form, place each group in its own container, such as a GroupBox or Panel control.
- RadioButton and CheckBox controls have a similar function: they offer choices a user can select or clear.
- The difference is that multiple CheckBox controls can be selected at the same time, but option buttons are mutually exclusive. Use the Checked property to get or set the state of a RadioButton.
- The option button's appearance can be altered to appear as a toggle-style button or as a standard option button by setting the Appearance property.

➤ **Properties of RadioButton :**

Name	Description
Checked	Gets or sets a value indicating whether the control is checked.
Enabled	Gets or sets a value indicating whether the control can respond to user interaction.

➤ **Methods of RadioButton :**

- RadioButton supports all the general methods, which are already discussed in the beginning.

○ **Events of RadioButton :**

Name	Description
CheckedChanged	Occurs when the value of the Checked property changes.
Click	Occurs when the control is clicked.

➤ **CheckBox :**

- Checkboxes are also familiar controls—you click a checkbox to select it, and click it again to deselect it. When you select a checkbox, a check appears in it, indicating that the box is indeed selected. You use a checkbox to give the user an option, such as true/false or yes/no. The checkbox control can display an image or text or both.
- RadioButton and CheckBox controls have a similar function: they offer choices a user can select or clear. The difference is that multiple CheckBox controls can be selected at the same time, but option buttons are mutually exclusive.

○ **Properties of CheckBox :**

Name	Description
Checked	Gets or sets a value indicating whether the control is checked.
Enabled	Gets or sets a value indicating whether the control can respond to user interaction.

○ **Methods of CheckBox :**

Name	Description
AddText()	Adds a specified text string to the CheckBox.
Enabled()	Gets or sets a value indicating whether the control can respond to user interaction.

○ **Events of CheckBox :**

Name	Description
Checked	Occurs when the Checkbox is checked.
Unchecked	Occurs when the Checkbox is unchecked.

➤ **ListBox :**

- As you know, list boxes display a list of items from which the user can select one or more. If there are too many items to display at once, a scroll bar automatically appears to let the user scroll through the list.
- The items in list boxes are stored in the **Items** collection; the **Items.Count** property holds the number of items in the list. (The value of the **Items.Count** property is always one more than the largest possible **SelectedIndex** value because **SelectedIndex** is zero-based.) To add or delete items in a **ListBox** control, you can use the **Items.Add**, **Items.Insert**, **Items.Clear**, or **Items.Remove** methods. You also can add a number of objects to a list box at once with the **AddRange** method. Or you can add and remove items to the list by using the **Items** property at design time.
- You also can support multiple selections in list boxes. The **SelectionMode** property determines how many list items can be selected at a time; you can set this property to **None**, **One**, **MultiSelect**, or **MultiExtended**:
 - **MultiExtended**— Multiple items can be selected, and the user can use the Shift, Ctrl, and arrow keys to make selections.
 - **MultiSimple**— Multiple items can be selected.
 - **None**— No items may be selected.
 - **One**— Only one item can be selected.
- When you support multiple selections, you use the **Items** property to access the items in the list box, the **SelectedItems** property to access the selected items, and **SelectedIndices** property to access the selected indices.

○ **Properties of ListBox :**

Name	Description
DataSource	Gets or sets the data source for this ListBox.
DropDownStyle	Gets or sets a value specifying the style of the combo box. The DropDownStyle property specifies whether the list is always displayed or whether the list is displayed in a drop

	The DropDownStyle property also specifies whether the text portion can be edited.
SelectionMode	As listboxes can be used for Multiple selection. This specified how many list items selected at a time. You can set this property to None , One , MultiSelect , or MultiExtended .
SelectedIndex	Gets or sets the index specifying the currently selected item. If more then one item selected it gives index no. of first item.
SelectedItem	Gets or sets currently selected item in the ListBox. If more then one items are selected gives first selected item.
SelectedText	Gets or sets the text that is selected in the editable portion of a ListBox.
SelectedIndices	Similar to SelectedIndex property. SelectedIndex property gives index no. of first selected item where as SelectedIndices gives you index nos of all the selected items in the ListBox.
SelectedItems	It gives list of all selected item. By using loop of SelectedItems you can find the selected items in ListBox.
SelectionLength	Gets or sets the number of characters selected in the editable portion of the combo box.
SelectionStart	Gets or sets the starting index of text selected in the combo box.
Sorted	Gets or sets a value indicating whether the items in the combo box are sorted.
Items.Count	Displays the no. of items available in ListBox.

○ **Methods of ListBox :**

Name	Description
FindString(String)	Returns the index of the first item in the ListBox that starts with the specified string.
Items.Add	Allows you to add item to the ListBox. The added item is kept last if sorted property set to true.
Items.Insert	Inserts (Adds) the item at specified position in ListBox.
Items.Remove	Allows you to remove particular item by specifying text of item.
Items.RemoveAt	Allows you to remove particular item by specifying index of item.
Items.Clear	Clears the items which are available in ListBox.
Items.IndexOf	Gives the index no. of specified item in ListBox

○ **Events of ListBox :**

Name	Description
Click	Occurs when the control is clicked.
SelectedIndexChanged	Occurs when the SelectedIndex property has changed.
SelectedValueChanged	Occurs when the SelectedValue property changes.
TextChanged	Occurs when the Text property value changes.

➤ **CheckedListBox :**

- CheckedListBox is the ordinary ListBox with checkbox for each items in the list.
- It supports all properties, methods, and events of ListBox.
- It provides the facility to check the item from the list.

➤ **ComboBox :**

- The Windows forms combo box control is used to display data in a drop-down combo box. The combo box is made up of two parts: The top part is a text box that allows the user to type in all or part of a list item. The other part is a list box that displays a list of items from which the user can select one or more. You can allow the user to select an item from the list, or enter their own data.
- ComboBox is called as List Control which is used to display list of items.
- **Properties of ComboBox :**

Name	Description
DataSource	Gets or sets the data source for this ComboBox.
DropDownStyle	Gets or sets a value specifying the style of the combo box. The DropDownStyle property specifies whether the list is always displayed or whether the list is displayed in a drop-down list. The DropDownStyle property also specifies whether the text portion can be edited.
SelectedIndex	Gets or sets the index specifying the currently selected item.
SelectedItem	Gets or sets currently selected item in the ComboBox.
SelectedText	Gets or sets the text that is selected in the editable portion of a ComboBox.
SelectionLength	Gets or sets the number of characters selected in the editable portion of the combo box.
SelectionStart	Gets or sets the starting index of text selected in the combo box.
Sorted	Gets or sets a value indicating whether the items in the combo box are sorted.
Items.Count	Displays the no. of items available in ComboBox.

- **Methods of ComboBox :**

Name	Description
FindString(String)	Returns the index of the first item in the ComboBox that starts with the specified string.
Items.Add	Allows you to add item to the Combobox. The added item is kept last if sorted property is not set to true.
Items.Insert	Inserts (Adds) the item at specified position in ComboBox.
Items.Remove	Allows you to remove particular item by specifying text of item.
Items.RemoveAt	Allows you to remove particular item by specifying index of item.
Items.Clear	Clears the items which are available in ComboBox.
Items.IndexOf	Gives the index no. of specified item in ComboBox

- **Events of ComboBox :**

Name	Description
Click	Occurs when the control is clicked.
SelectedIndexChanged	Occurs when the SelectedIndex property has changed.
SelectedValueChanged	Occurs when the SelectedValue property changes.
TextChanged	Occurs when the Text property value changes.

➤ **ListView :**

- If tree views are all about displaying node hierarchies, like the folder hierarchy on a disk, then list views are all about displaying lists of items. You can see a list view in the right pane in the Windows Explorer (the part that displays what files are in a folder).
- You can add list views to a Windows form just like any other control—just drag it from the toolbox to the form at design time. To add items to a list box, open the **Items** property in the Properties window.
- There are four views you can use in a list view. You select one by assigning a value (**View.LargeIcon**, **View.SmallIcon**, **View.List**, or **View.Details**) to the list view's **View** property:
- **Large icon mode**—displays large icons (large icons are 32×32 pixels) next to the item text.
- **Small icon mode**—is the same as the large icon mode except that it displays items using small icons (small icons are 16×16 pixels).
- **List mode**—displays small icons, and always in one column.
- **Report mode** (also called the details mode)—displays items in multiple columns, displaying column headers and fields.
- **Properties of ListView :**

Name	Description
CheckBoxes	Gets/sets if every item should show a checkbox.
CheckedIndices	Gets the indices of currently checked items.
CheckedItems	Gets the currently checked items.
Columns	Gets a collection of columns.
FullRowSelect	Gets/sets whether selecting an item will select the entire row.
GridLines	Gets/sets whether grid lines are drawn between items and their subitems.
Items	Gets the list items.
LargeImageList	Gets/sets the ImageList for large icon view.
MultiSelect	Gets/sets whether multiple items can be selected.
SelectedIndices	Gets the indices of the selected items.
SelectedItems	Gets the selected items.
SmallImageList	Gets/sets the small icon image list.
Sorting	Gets/sets the sort order for the items.
View	Gets/sets the current view mode.

- **Methods of ListView :**

Name	Description
ArrangeIcons	Arranges the displayed items in Large Icon or Small Icon view.
Clear	Removes all items from the list view.
GetItemAt	Gets the item corresponding to the given X,Y coordinate.

Events of ListView :

Name	Description
ColumnClick	Occurs when a column is clicked.
ItemCheck	Occurs when an item is checked.
SelectedIndexChanged	Occurs when the selected index changes.

➤ **TreeView :**

- A **TreeView** control displays a hierarchical list of **Node** objects, each of which consists of a label and an optional bitmap. A **TreeView** is typically used to display the headings in a document, the entries in an index, the files and directories on a disk, or any other kind of information that might usefully be displayed as a hierarchy.
- You use a tree view to display a hierarchy of nodes. Each node is not only displayed visually, but also can have child nodes. An example of this is the Windows Explorer, which uses a tree view in its left pane to display the hierarchy of folders on disk. You can expand and collapse parent nodes by clicking them; when expanded, their children are also visible.
- There are 3 types of nodes in tree view as follows :
- **Root Node** : The node which is at root level is called as Root Node
- **Parent Node**: The node which is a child of Root Node and has got its children is called Parent Node.
- **Child Node** : The node which comes under Parent Node and is the leaf node, is Called as Child Node. It is also known as Leaf Node.
- After creating a **TreeView** control, you can add, remove, arrange, and otherwise manipulate **Node** objects by setting properties and invoking methods. You can programmatically expand and collapse **Node** objects to display or hide all child nodes. Three events, the Collapse, Expand, and NodeClick event, also provide programming functionality.
- **Properties of TreeView :**

Name	Description
HasChildren	Gets a value indicating whether the control contains one or more child controls.
CheckBoxes	Gets/sets whether checkboxes should be displayed next to tree nodes.
ImageList	Gets or sets the ImageList that contains the Image objects that are used by the tree nodes.
Nodes	Gets the collection of tree nodes that are assigned to the tree view control.
PathSeparator	Gets or sets the delimiter string that the tree node path uses.
SelectedNode	Gets or sets the tree node that is currently selected in the tree view control.
ShowLines	Gets or sets a value indicating whether lines are drawn between tree nodes in the tree view control.
ShowPlusMinus	Gets or sets a value indicating whether plus-sign (+) and minus-sign (-) buttons are displayed next to tree nodes that contain child tree nodes.
ShowRootLines	Gets or sets a value indicating whether lines are drawn between the tree nodes that are at the root of the tree view.
TopNode	Gets or sets the first fully-visible tree node in the tree view control.
VisibleCount	Gets the number of tree nodes that can be fully visible in the tree view control.

- **Methods of TreeView**

Name	Description
BeginUpdate	Disables redrawing of the tree view.
CollapseAll	Collapses all nodes.
EndUpdate	Enables redrawing of the tree view.
ExpandAll	Expands all the nodes.
GetNodeAt	Gets the node that is at the given location.
GetNodeCount	Gets the number of nodes.

- **Methods of Nodes Collection :**

Name	Description
------	-------------

Nodes.Add	Used to add a node in tree view. The new node is added at last.
Nodes.Insert	Used to add a node at specific position.
Nodes.Remove	Used to remove node from tree view
Nodes.RemoveAt	Used to remove specific node by specifying index no.
Nodes.Clear	Used to clear all the nodes of specific node.
Nodes.Contains	Used to check whether specified node exists or not.
Nodes.IndexOf	Used to know index no. of specific node. If specific node does not exist it return

○ **Events of TreeView**

Name	Description
AfterCheck	Occurs when a node checkbox is checked.
AfterCollapse	Occurs when a tree node is collapsed.
AfterExpand	Occurs when a tree node is expanded.
AfterLabelEdit	Occurs when a tree node label text is edited.
AfterSelect	Occurs when a tree node is selected.
BeforeCheck	Occurs before a node checkbox is checked.
BeforeCollapse	Occurs before a node is collapsed.
BeforeExpand	Occurs before a node is expanded.
BeforeLabelEdit	Occurs before a node label text is edited.
BeforeSelect	Occurs before a node is selected.
ItemDrag	Occurs when an item is dragged into the tree view.

- The **Nodes** collection for a node holds the node's child **TreeNode** objects. You can add, remove, or clone a **TreeNode**, and when you do, all child tree nodes are added, removed, or cloned at the same time. Each **TreeNode** can contain a collection of other **TreeNode** objects, which means you can use expressions like this: **MyNode.Nodes(3).Nodes(5)** to refer to child nodes (in this case, actually grandchild nodes). You also can use the **FullPath** property to specify nodes in terms of their absolute, not relative, locations. And you can use the **Nodes** collection's **Add** or **Remove** methods to add or remove nodes in code.

➤ **PictureBox :**

- Picture boxes are used to display graphics from a bitmap, icon, JPEG, GIF or other image file type.
- To display an image in a picture box, you can set the **Image** property to the image you want to display, either at design time or at run time. You can clip and position an image with the **SizeMode** property, which you set to values from the **PictureBoxSizeMode** enumeration:
- **Normal**— Standard picture box behavior (the upper-left corner of the image is placed at upper left in the picture box).
- **StretchImage**— Allows you to stretch the image in code.
- **AutoSize**— Fits the picture box to the image.
- **CenterImage**— Centers the image in the picture box.
- You also can change the size of the image at run time with the **ClientSize** property, stretching an image as you want. By default, a **PictureBox** control is displayed without any borders, but you can add a standard or three-dimensional border using the **BorderStyle** property. And you can even handle events such as **Click** and **MouseDown** to convert an image into an image map.
- **Properties of PictureBox :**

Name	Description
BorderStyle	Gets/sets the border style for the picture box.
Image	Gets/sets the image that is in a picture box.

- **Methods of PictureBox :**

(There are no specific methods for PictureBox apart from common methods)

- **Events of PictureBox :**

Name	Description
Resize	Occurs when the picture box is resized.
SizeModeChanged	Occurs when SizeMode changes.

➤ **Panel :**

- Panels are those controls which contain other controls, for example, a set of radio buttons, checkboxes, etc. Panels are similar to Groupboxes but the difference, Panels cannot display captions where as GroupBoxes can and Panels can have scrollbars where as GroupBoxes can't.
- If the Panel's Enabled property is set to False then the controls which the Panel contains are also disabled. Panels are based on the ScrollableControl class.
- Notable property of the Panel control in the appearance section is the BorderStyle property. The default value of the BorderStyle property is set to None. You can select from the predefined list to change a Panels BorderStyle.
- Notable property in the layout section is the AutoScroll property. Default value is set to False. Set it to True if you want a scrollbar with the Panel.

- **Difference between Panel and GroupBox :**

Panel	GroupBox
Panel can not be used to display Caption/Text.	GroupBox can be used to display Caption/Text.
Panels can have scroll bars.	GroupBox can not have scrollbar
Generally Panel is used if you don't want to show d groups under Border. Although they have Bord property but bydefault BorderStyle is set to None.	Generally GroupBox is used to display group along Border.

- **Properties of Panel :**

Name	Description
AutoScroll	Gets or sets a value indicating whether the container enables the user to scroll controls placed outside of its visible boundaries.
BorderStyle	Indicates the border style for the control.
Enabled	Gets or sets a value indicating whether the control can respond to user interaction.

- **Methods of Panel :**

(Apart from general methods Panel control does not have any special methods)

- **Events of Panel :**

(Apart from Scroll event all other events are common to Panel control)

Name	Description
Scroll	Occurs when the user or code scrolls through the Panel.

➤ **ScrollBar :**

- Windows Forms **ScrollBar** controls are used to provide easy navigation through a long list of items or a large amount of information by scrolling either horizontally or vertically within an application or control. Scroll bars are a common element of the Windows interface, so the **ScrollBar** control is often used with controls that do not derive from the ScrollableControl class. Similarly, many developers choose to incorporate the **ScrollBar** control when authoring their own user controls.
- The **HScrollBar** (horizontal) and **VScrollBar** (vertical) controls operate independently from other controls and have their own set of events, properties, and methods. **ScrollBar** controls are not the same as the built-in scroll bars that are attached to text boxes, list boxes, combo boxes, or MDI forms (the **TextBox** control has a ScrollBars property to show or hide scroll bars that are attached to the control).
- The **ScrollBar** controls use the Scroll event to monitor the movement of the scroll box (sometimes referred to as the thumb) along the scroll bar. Using the **Scroll** event provides access to the scroll bar value as it is being dragged.
- **Value Property :**
- The Value property (which, by default, is 0) is an **integer** value corresponding to the position of the scroll box in the scroll bar. When the scroll box position is at the minimum value, it moves to the left-most position (for horizontal scroll bars) or the top position (for vertical scroll bars). When the scroll box is at the maximum value, the scroll box moves to the right-most or bottom position. Similarly, a value halfway between the bottom and top of the range places the scroll box in the middle of the scroll bar.
- **LargeChange and SmallChange Properties :**
- When the user presses the PAGE UP or PAGE DOWN key or clicks in the scroll-bar track on either side of the scroll box, the **Value** property changes according to the value set in the LargeChange property. When the user presses one of the arrow keys or clicks one of the scroll-bar buttons, the **Value** property changes according to the value set in the SmallChange property.
- **Properties of HScrollBar / VScrollBar :**

Name	Description
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.
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.
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.

- **Methods of HScrollBar / VScrollBar :**
(There are no specific methods used for HScrollBar or VScrollBar instead of general methods)
- **Events of HScrollBar / VScrollBar :**

Name	Description
Scroll	Occurs when the scroll box has been moved by either a mouse or keyboard action.
ValueChanged	Occurs when the Value property is changed.

➤ **Tooltip :**

- A ToolStrip is a container for ToolStripItem elements. Each individual element on the ToolStrip is a ToolStripItem that manages the layout and event model for the type it contains. For example, elements visible on the toolbar, such as buttons, text boxes, labels, or combo boxes, or visible on the menu bar.
- ToolStrip is a container for ToolStripItem elements. Each individual element on the ToolStrip is a ToolStripItem that manages the layout and event model for the type it contains. . The ToolStrip controls provide a common interface for Menus and Strips in Windows Forms.
- Following is a list of ToolStripItem controls you can place within a ToolStrip.

Control Implementation	Description
ToolStripLabel	Used to display normal text, hyperlinks, and images.
ToolStripButton	Provides a typical pushbutton that you can configure to support both text and image.
ToolStripComboBox	A ComboBox with methods/properties to configure various styles.
ToolStripSeparator	A separator that you can use to visually separate groups of ToolStripItem elements.
ToolStripDropDown	This control provides a button which, when clicked, displays a ToolStripDropDown control. You'll implement a sample of this to get a better idea of how to use it.
ToolStripTextBox	A normal textbox which can be used to enter text.
ToolStripMenuItem	A special menu control built specifically for use with the MenuStrip and ContextMenuStrip controls.
ToolStripProgressBar	A specialized progress bar implementation for use within a ProgressBar control.
ToolStripSplitButton	A combination of normal button and a DropDownButton.
ToolStripControlHost	A control that acts as a host to your customized implementations or any other Windows Forms controls.

- All these controls are usual controls which you use in developing Windows Forms. The only difference is they appear under ToolStrip. ToolStrip is just a container of all these controls. All the related properties, methods and events are used as we use in programming.
- ToolStrip has got one important property called "Items" which is collection of all the items (toolbar items) which are placed under.

➤ **DialogBox:**

- There are a number of built-in dialog boxes in Visual Basic, which is great, because developing your own file open, file save, and other dialog boxes not only takes a lot of work, but gives your program a different look from what Windows users are already used to. They are listed as follows
 - ❖ **Color Dialog Box (ColorDialog)**
 - ❖ **Font Dialog Box (FontDialog)**
 - ❖ **Open File Dialog Box (OpenFileDialog)**
 - ❖ **Save File Dialog Box (SaveFileDialog)**
 - ❖ **Print File Dialog Box (PrintDialog)**
- There are also other built-in dialog boxes like Print Preview Dialog Box, Page Setup Dialog Box, etc. which are used as per requirement.
- You use the **ShowDialog** method to display the dialog at run time and can check its return value (such as **DialogResult.OK** or **DialogResult.Cancel**) to see which button the user has clicked. Here are the possible return values from this method, from the **DialogResult** enumeration:

Name	Description
Abort	The dialog box return value is Abort (usually from a button labeled Abort).
Cancel	The dialog box return value is Cancel (usually from a button labeled Cancel).
Ignore	The dialog box return value is Ignore (usually from a button labeled Ignore).
No	The dialog box return value is No (usually from a button labeled No).
None	Nothing is returned from the dialog box. This means that the modal dialog Continues running.
OK	The dialog box return value is OK (usually from a button labeled OK).
Retry	The dialog box return value is Retry (usually from a button labeled Retry).
Yes	The dialog box return value is Yes (usually from a button labeled Yes).

- Now all the dialog boxes are explained in detail.

➤ **Color Dialog Box (ColorDialog)**

- Color dialogs let the user select a color in an easy way. The principal property you use of these dialogs is the **Color** property, which returns a **Color** object, ready for use.
- If you set the **AllowFullOpen** property to **False**, on the other hand, the Define Custom Colors button is disabled and the user can select colors only from the predefined colors in the palette. Note also that if you set the **SolidColorOnly** property to **True**, the user can select only solid (not dithered) colors.

○ **Properties of ColorDialog :**

Property	Description
AllowFull	Gets/sets whether the user can use the dialog box to define custom colors.
AnyColo	Gets/sets whether the dialog box displays all available colors in the set of basic colors.
Color	Gets/sets the color selected by the user.
CustomC	Gets/sets the set of custom colors shown in the dialog box.
FullOpen	Gets/sets whether the controls used to create custom colors are visible when the dialog opened
ShowHe	Gets/sets whether a Help button appears in the color dialog box.
SolidCol	Gets/sets whether the dialog box will restrict users to selecting solid colors only.

○ **Methods of ColorDialog :**

Method	Means
Reset	Resets all dialog options to their default values.
ShowDialog	Shows the dialog.

○ **Event of ColorDialog :**

Event	Means
HelpRequest	Occurs when the user clicks the Help button.

➤ **Open File Dialog Box(OpenFileDialog) and Save File Dialog Box (SaveFileDialog)**

- As you'd expect from its name, the Open File dialog and Save File dialog lets the user select a file to open or save. In fact, it's the same Open File dialog and Save File dialog used by Windows itself.
- There is no difference between OpenFileDialog and SaveFileDialog except OpenFileDialog is used to open a file and SaveFileDialog is used to save a file.
- Open File dialogs are supported with the **OpenFileDialog** class. And Save File dialogs are supported with the **SaveFileDialog** class. You can let users select multiple files with the **Multiselect** property. You can use the **ShowReadOnly** property 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 choices that appear in the "Files of type" box in the dialog box. The name and path the user selected is stored in the **FileName** property of the **OpenFileDialog** or **SaveFileDialog** object—and there's a neat shortcut here: you can use the **OpenFile** method to open the selected file directly.
- **Properties of OpenFileDialog / SaveFileDialog :**

Property	Means
CheckFile	Gets/sets if the dialog box displays a warning if the user specifies a nonexistent file.
CheckPath	Gets/sets whether the dialog box displays a warning if the user gives a path that does not
DefaultExt	Gets/sets the default file extension.
FileName	Gets/sets the file name selected in the file dialog box.
FileNames	Gets the file names of all selected files.
Filter	Gets/sets the current file name filter string, which sets the choices that appear in the "Save type" or "Files of type" box.
FilterIndex	Gets/sets the index of the filter selected in the file dialog box.
InitialDir	Gets/sets the initial directory used in the file dialog box.
Multiselect	Gets/sets whether the dialog box allows multiple file selections.
ShowHelp	Gets/sets whether the Help button should be displayed.
ShowReadOnly	Gets/sets whether the dialog displays a read-only check box.
Title	Gets/sets the file dialog box title.

- **Methods of OpenFileDialog / SaveFileDialog :**

Method	Means
OpenFile	Opens the file selected by the user, with read-only permission. The file is specified FileName property.
Reset	Resets all options to their default values.
ShowDialog	Shows the dialog box.

- **Events of OpenFileDialog / SaveFileDialog :**

Event	Means
FileOk	Occurs when the user clicks the Open or Save button.
HelpRequest	Occurs when the user clicks the Help button.

➤ **Font Dialog Box (FontDialog)**

- Font dialogs let the user select a font size, face, color, and so on. To display the font dialog box, call the **ShowDialog** method. This dialog shows list boxes for **Font**, **Style**, and **Size**, checkboxes for effects like **Strikeout** and **Underline**, and a sample of how the font will appear. You can recover these settings using properties of the same names of the **Font** object returned by the **Font** property.

- The **FontDialog** class displays a dialog box that lets the user select a font. It returns a **Font** object in the **Font** property, and a **Color** object in the **Color** property. **Font** dialogs are supported by the **FontDialog** class.

- **Properties of FontDialog :**

Property	Description
Color	Gets/sets the selected font color.
FixedPitch	Gets/sets whether the dialog box allows only the selection of fixed-pitch fonts.
Font	Gets/sets the selected font.
FontMustExist	Gets/sets whether the dialog box specifies an error condition if the user attempts to select a font or style that does not exist.
MaxSize	Gets/sets the maximum point size a user can select.
MinSize	Gets/sets the minimum point size a user can select.
ShowApply	Gets/sets whether the dialog box contains an Apply button.
ShowColor	Gets/sets whether the dialog box displays the color choice.
ShowEffects	Gets/sets whether the dialog box contains controls that allow the user to specify strikethrough, underline, and text color options.
ShowHelp	Gets/sets whether the dialog box displays a Help button.

- **Methods of FontDialog :**

Method	Description
Reset	Resets all dialog options to default values.
ShowDialog	Shows the dialog.

- **Events of FontDialog :**

Event	Description
Apply	Occurs when the user clicks the Apply button.
HelpRequest	Occurs when the user clicks the Help button.

➤ **Print Dialog Box (PrintDialog)**

- Print dialogs let the user print documents, and these dialogs are supported with the **PrintDialog** class. Before displaying a Print dialog, you set the **Document** property of a **PrintDialog** object to a **PrintDocument** object, and the **PrinterSettings** property to a **PrinterSettings** object of the kind set by Page Setup dialogs.
- When the dialog is closed, you can print the document by assigning the **PrintDialog** object's **PrinterSettings** property (which returns a **PrinterSettings** object as configured by the user, indicating the number of copies to print, the printer to use, and so on) to the **PrinterSettings** property of the **PrintDocument** object and use the **PrintDocument** object's **Print** method to actually print the document.

- **Properties of PrintDialog :**

Property	Means
AllowPrintToFile	Gets/sets whether the Print to file checkbox is enabled.

AllowSelection	Gets/sets whether the Selection radio button is enabled.
AllowSomePages	Gets/sets whether the From... To... Page radio button is enabled.
Document	Gets/sets the PrintDocument used to obtain PrinterSettings .
PrinterSettings	Gets/sets the PrinterSettings dialog box to modify.
PrintToFile	Gets/sets whether the Print to file checkbox is checked.
ShowHelp	Gets/sets whether the Help button is displayed.
ShowNetwork	Gets/sets whether the Network button is displayed.
Method	Means
Reset	Resets all dialog options.
ShowDialog	Shows the dialog.

○ **Methods of PrintDialog :**

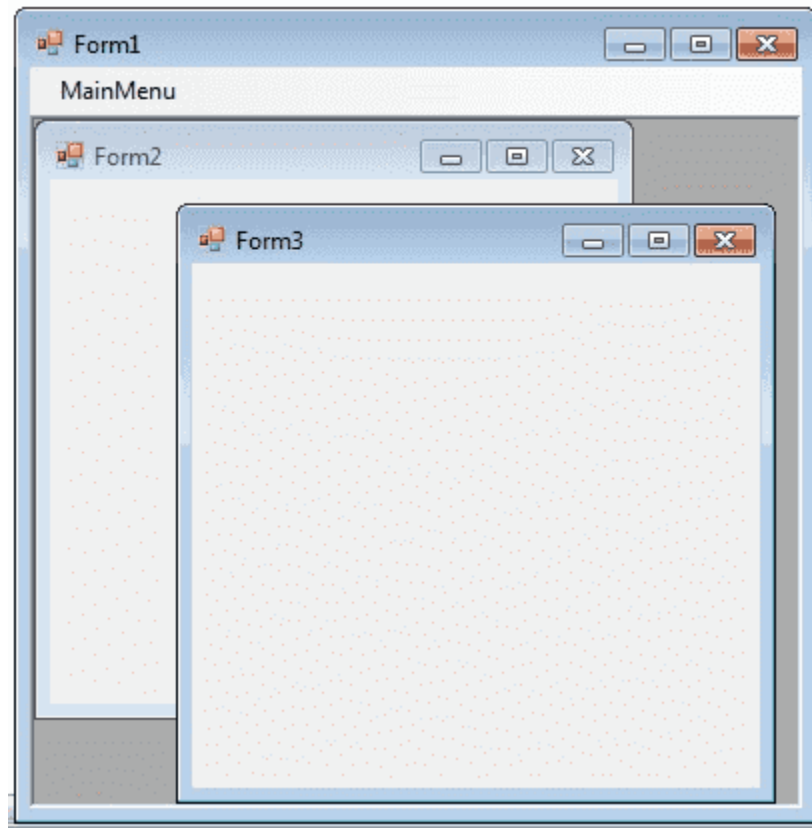
Method	Means
Reset	Resets all dialog options.
ShowDialog	Shows the dialog.

○ **Event of PrintDialog :**

Event	Means
HelpRequest	Occurs when the user clicks the Help button.

➤ **MDI Concept:**

- A Multiple Document Interface (MDI) programs can display multiple child windows inside them.
- This is in contrast to single document interface (SDI) applications, which can manipulate only one document at a time.
- Visual Studio Environment is an example of Multiple Document Interface (MDI) and notepad is an example of an SDI application.
- MDI applications often have a Window menu item with submenus for switching between windows or documents.



Any windows can become an MDI parent, if you set the `IsMdiContainer` property to `True`.

`IsMdiContainer = true;`

The following C# program shows a MDI form with two child forms. Create a new C# project, then you will get a default form Form1 . Then add two more forms in the project (Form2 , Form 3) . Create a Menu on your form and call these two forms on menu click event. Click here to see how to create a Menu on your form [How to Menu Control C#](#).

NOTE: If you want the MDI parent to auto-size the child form you can code like this.

`form.MdiParent = this;`

`form.Dock=DockStyle.Fill;`

`form.Show();`

➤ **MDI Notepad Concept:**

- The MDI NotePad sample application is a simple text editor similar to the NotePad application included with Microsoft Windows.
- The MDI NotePad application, however, uses a multiple-document interface (MDI). At run time, when the user requests a new document (implemented with the New command on the application's File menu), the application creates a new instance of the child form.
- This allows the user to create as many child forms, or documents, as necessary.

To create a document-centered application in Visual Basic, you need at least two forms — an MDI form and a child form.

At design time, you create an MDI form to contain the application and a single child form to serve as a template for the application's document.

To create your own MDI NotePad application

1. From the **File** menu, choose **New Project**.
2. From the **Project** menu, choose **Add MDI Form** to create the container form.
The project should now contain an MDI form (MDIForm1) and a standard form (Form1).
3. Create a text box (Text1) on Form1.

4. Set properties for the two forms and the text box as follows.

Object	Property	Setting
MDIFrm1	Caption	MDI New Document
Form1	Caption	Untitled
	MDIChild	True
Text1	MultiLine	True
	Text	(Empty)
	Left	0
	Top	0

- 5.
6. Using the **Menu Editor** (from the **Tools** menu), create a File menu for MDIFrm1.

Caption	Name	Indexed
&File	mnuF	No
&New	mnuN	Yes

- 7.
8. Add the following code to the mnuFileNew_Click procedure:
9. Private Sub mnuFileNew_Click ()
10. ' Create a new instance of Form1, called NewDoc.
11. Dim NewDoc As New Form1
12. ' Display the new form.
13. NewDoc.Show
14. End Sub
- This procedure creates and then displays a new instance (or copy) of Form1, called NewDoc. Each time the user chooses New from the File menu, an exact duplicate (instance) of Form1 is created, including all the controls and code that it contains.
15. Add the following code to the Form_Resize procedure for Form1:
- Private Sub Form_Resize ()
- ' Expand text box to fill the current child form.
- Text1.Height = ScaleHeight
- Text1.Width = ScaleWidth
- End Sub
- Press F5 to run the application.

➤ **Timer :**

- Timers are very useful controls, because they let you create periodic events. Strictly speaking, timers are no longer controls but components, and they do not appear in a window at run time. At design time, they appear in the component tray under the form you've added them.
- Windows timers are designed for a single-threaded (as opposed to multithreaded) environment; you set how often you want the timer to generate **Tick** events by setting the **Interval** property (in milliseconds, one thousandths of a second). Each time a **Tick** event happens, you can execute code in a handler for this event, just as you would for any other event.

- **Properties of Timer :**

Name	Description
Enabled	Gets/sets whether the timer is running.
Interval	Gets/sets the time (in milliseconds) between timer ticks.

- **Methods of Timer :**

Name	Description
Start	Starts the timer whenever called.
Stop	Stops the timer whenever called.

- **Event of Timer**

Name	Description
Tick	Occurs when the timer interval has elapsed (and the timer is enabled).

➤ **Handling Mouse Events :**

- Mice can do a lot more than nibble cheese. In a Windows UI, they can let you know when one of their buttons has been clicked or released, or when the mouse pointer leaves or enters some part of the application.
- This information is provided in the form of **MouseDown**, **MouseUp**, **MouseMove**, **MouseEnter**, **MouseLeave**, and **MouseHover** events.
- When wiring up mouse events, you can also change the mouse cursor. You typically marry the ability to change the cursor to the **MouseEnter** and **MouseLeave** events. These are used to provide feedback to the user that something is happening, not happening, or that certain areas are offlimits or welcome to the explorative nature of your cursor.
- Mouse events occur in the following order:

- 1. MouseEnter**
 - 3. MouseHover/MouseDown/MouseWheel**
 - 5. MouseLeave**

- 2. MouseMove**
 - 4. MouseUp**

Property	Purpose
Button	Tells you which mouse button was pressed
Clicks	Tells you how many times the mouse button was pressed and released
Delta	<ul style="list-style-type: none"> ○ Retrieves a signed count Keyboard events are fired when a key is pressed on a control that has the focus. ○ The key event handler receives an argument of type KeyPressEventArgs containing data related to this keypress event. Table lists the KeyPressEventArgs properties that provide information related to the event received. ○ <i>KeyPressEventArgs Properties :</i> <ul style="list-style-type: none"> of the number of <i>detents</i> the mouse wheel has rotated. A detent is one notch of the mouse wheel.
X	Retrieves the X coordinate of a mouse click
Y	Retrieves the Y coordinate of a mouse click

➤ **Handling Key Events :**

- **KeyPress** events also bubble up from the OS and are made available to you in **KeyPress**, **KeyDown**, and **KeyUp** events. Mouse event handlers receive an argument of type **EventArgs** containing data related to their events; however, key-press event handlers receive an argument of type **KeyEventArgs** (a derivative of **EventArgs**) containing data specific to the keypress or key release events.

Property	Purpose
Handled	Retrieves or returns a value to the property indicating whether the KeyPress event is handled
KeyChar	Retrieves the KeyChar (character pressed) value corresponding to the key press events occur in the following order: 1.KeyDown 2.KeyPress 3. KeyUp

Link Label:

- A Link Label control is a label control that can display a hyperlink. A Link Label control is inherited from the Label Class so it has all the functionality provided by the windows form label control.
- Link Label control does not participate in user input or capture mouse or keyboard events.

Chapter-4 :- Database Programming With ADO.NET

○ **ADO .NET :**

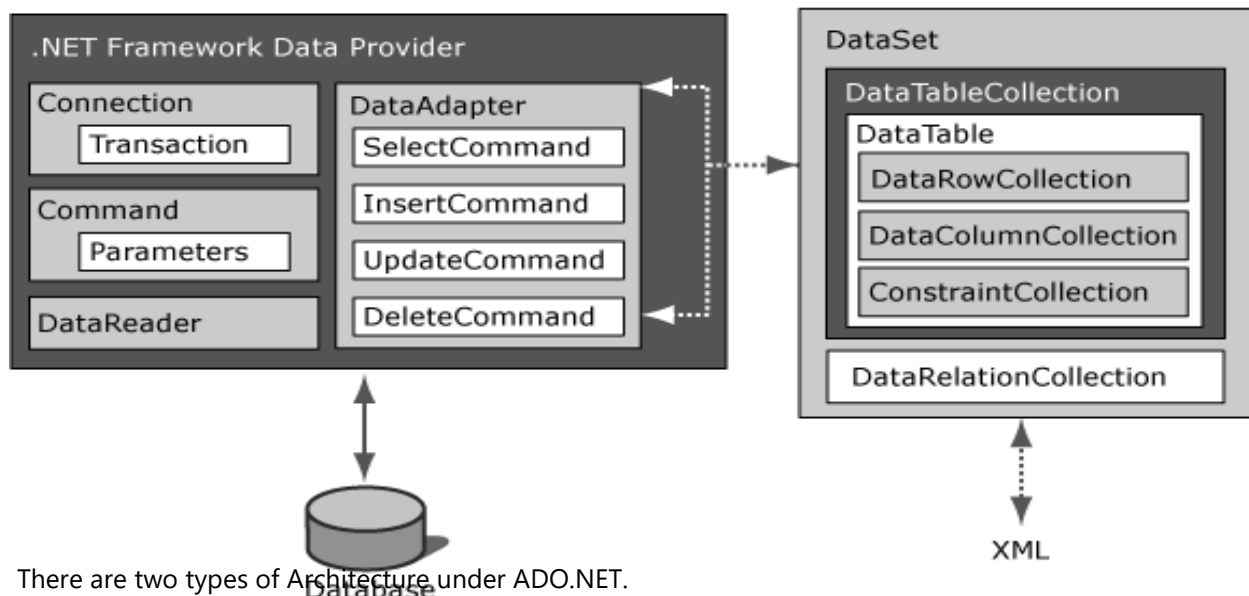
ADO.NET provides excellent support for reading data from a DBMS, including a set of classes for fast, efficient access to data in SQL Server databases and another set of classes to support OLE DB data sources. Of course you can also use other database under ADO.NET.

ADO.NET is a large set of .NET classes that enable us to retrieve and manipulate data, and update data sources, in very many different ways.

ADO.NET has several advantages. They are as follows :

- You can use ADO.NET to connect with any type of database.
- ADO.NET supports Connected as well as Disconnected Architecture. Disconnected Architecture is a new invention since ADO.NET is introduced.
- XML is supported by ADO.NET which is widely used for data transformation widely today. This makes your data transformation much more faster.
- You can create Fast, Scalable and Secured applications using ADO.NET
- Cross Language support is provided by ADO.NET as Multilanguage is a prime feature of .NET framework.

○ **Architecture of ADO .NET :**



- There are two types of Architecture under ADO.NET.

- Connected Architecture

- Disconnected Architecture

1) Connected Architecture :

- Connected Architecture simply means, you are connected with the database throughout your operations.
- Although connected architecture is faster than disconnected architecture, It has one main problem of creating more traffic at database end.
- Because you are constantly connected to database. And so do all other users.
- In Connected Architecture, you are blocking a memory portion of database which makes performance slower at some extent. Client's memory is not utilized in this.
- DataReader is Connected Architecture since it keeps the connection open until all rows are fetched one by one.
- Following set of classes are used while using Connected Architecture method :
 - **Connection** : This allows you to connect with database.
 - **Command** : This allows you to give commands like Insert, Update, Delete, Select, etc.
 - **DataReader** : This allows you to read data directly from the database..

2) DisConnected Architecture

- Disconnected Architecture is the new concept / feature introduced in ADO.NET. The "Disconnected" name itself is the answer.
- Disconnected architecture is a method of retrieving a record set from the database and storing it giving you the ability to do many CRUD (Create, Read, Update, Delete) operations on the data in memory, then it can be re-synchronized with the database when reconnecting. A method of using disconnected architecture is using a DataSet.
- DataSet is DisConnected Architecture since all the records are brought at once and there is no need to keep the connection alive.
- Following set of classes are used while using Disconnected Architecture method :
 - **Connection** : This allows you to connect with database.
 - **DataAdapter** : This plays an important role in Disconnected Architecture. DataAdapter is a mediator which provides (fills) data into DataSet / DataTable and again updates the data back to database.
 - **DataSet** : This is used if you want to load some group of tables into local memory. You can also use DataSet to load single table data. DataSet is collection of tables.
 - **DataTable** : This is used if you want to load only one table into local memory.
 - **DataRow** : This is used incase you want to add one new row into DataSet or DataTable. You can also use DataRow to remove rows from DataSet or DataTable.
 - **DataColumn** : This is used incase you want to modify any column or you want to add a new column to DataSet or DataTable.

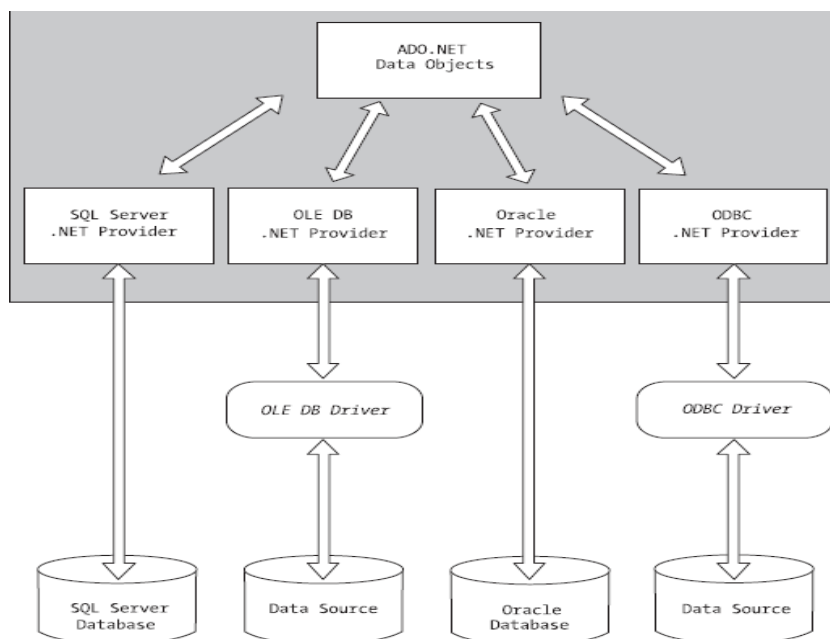
➤ Difference between Connected and Disconnected Architecture

Connected Architecture	Disconnected Architecture
In this, you are connected with database throughout your operation. Data is stored in database while doing operation.	In this, you get the copy of data and then do operation in local memory. Data is stored in local memory while operation.
This creates Heavy Traffic at database server when no. of user increases.	As data is loaded into local memory, does not create Traffic at database server side even no. of user increase
This utilizes memory of Database server as the data remains in database itself.	This utilizes memory of client as the data is loaded into client's machine.

Generally, transactional commands like Update, Delete, etc. are performed using Command and DataAdapter objects in Connected Architecture, as they do not require more time.	Generally, if you want to load data just for display purposes, Disconnected Architecture is the best choice as you just copy a copy of data into local memory which makes faster access.
Command and DataReader objects are used in Connected Architecture.	DataAdapter, DataSet, DataTable, DataRow and DataCommand objects are used in this architecture.
DataReader is used to fetch the data in Connected Architecture.	DataSet or DataTable is used to fetch the data in Disconnected Architecture.

➤ **Data Providers in ADO .NET :**

- Data Providers are different set of classes which provide functionality to access to different data sources like Access, Oracle, SQL Server, DB2, etc... databases.
- There are four types of Data Providers are provided by .Net.



- **SQL Server Provider :** This is special data provider which gives access to only SQL Server database. It supports connection to SQL Server 7.0 or later. The benefit of using this provider is it is faster as it directly deals with SQL Server.
- **Oracle Provider :** This is special data provider which gives access to only Oracle Server or client. It supports connection to Oracle 8i or later. The benefit of using this provider is it is faster as it directly deals with Oracle database.
- **OLEDB Provider :** This provider gives access to any database that has OLEDB driver. The benefit of using OLEDB provider is you can connect with any database using this.
- **ODBC Provider :** This provider gives access to any database that has ODBC (Open DataBase Connectivity) driver.

➤ **Namespaces used in ADO.NET**

Use of Namespaces depends on which provider you choose. Some of the classes are obtained from some common namespaces. But in order to use specific provider you need to import specific namespace. Following table shows list of some namespaces used in ADO.NET

Namespace	Description
System.Data	Contains fundamental classes with the core ADO.NET functionality. This includes DataSet and DataRelation, which allow you to manipulate structured relational data. These classes are totally independent of any specific type of database or way you use to connect to it.
System.Data.SqlClient	Contains the classes you use to connect to a Microsoft SQL Server database (version 7.0 or later). These classes, such as SqlCommand and SqlConnection, provide all the same properties and methods as their counterparts in the System.Data.OleDb namespace. The only difference is that they are optimized for SQL Server and provide better performance by eliminating the extra OLE DB layer (and by connecting directly to the optimized TDS interface).
System.Data.OracleClient	Contains the classes you use to connect to an Oracle database, such as OracleCommand and OracleConnection.
System.Data.OleDb	Contains the classes you use to connect to an OLE DB data source, including OleDbCommand and OleDbConnection.
System.Data.Odbc	Contains the classes you use to connect to a data source through an ODBC driver. These classes include OdbcCommand and OdbcConnection.

Depending on the namespace which you have imported, you can use provider specific classes in order to use ADO.NET. Consider following table for provider specific classes.

	SQL Server .NET Provider	OLE DB .NET Provider	Oracle .NET Provider	ODBC .NET Provider
➤ Connection :	SqlConnection	OleDbConnection	OracleConnection	OdbcConnection
○ Command :	SqlCommand	OleDbCommand	OracleCommand	OdbcCommand
○ DataReader :	SqlDataReader	OleDbDataReader	OracleDataReader	OdbcDataReader
○ DataAdapter :	SqlDataAdapter	OleDbDataAdapter	OracleDataAdapter	OdbcDataAdapter

The first thing we need to do with a database application is establish a connection to the database. ADO.NET handles this by using connection classes. In this article, Michael Youssef shows you how to start using connection classes, with examples.

- When developing database applications using .NET, the very first thing that we need is a connection to the database. ADO.NET provides us with connection classes like the SqlConnection class and OleDbConnection class. The SqlConnection class is part of the SQL Server .NET Data Provider. This data provider has been designed for performance optimization with SQL Server 7.0 and later.
- The OleDbConnection is part of the OLEDB .NET Data Provider, which is used to access a data source that has an OLEDB Provider. In this article (and actually most of my articles on ADO.NET) I use the SQL Server .NET Data Provider. So let's talk a little about the SqlConnection class before we write code.

○ **Properties of Connection Object**

Property	Description
CommandTimeout	Sets or returns the number of seconds to wait while attempting to execute a command
ConnectionString	Sets or returns the details used to create a connection to a data source
ConnectionTimeout	Sets or returns the number of seconds to wait for a connection to open
DefaultDatabase	Sets or returns the default database name
Mode	Sets or returns the provider access permission
Provider	Sets or returns the provider name
State	Returns a value describing if the connection is open or closed

Version	Returns the ADO version number
---------	--------------------------------

○ **Methods of Connection Object**

Method	Description
BeginTrans	Begins a new transaction
Close	Closes a connection
CommitTrans	Saves any changes and ends the current transaction
Open	Opens a connection
RollbackTrans	Cancels any changes in the current transaction and ends the transaction

○ **Events of Connection Object**

Event	Description
BeginTransComplete	Triggered after the BeginTrans operation
CommitTransComplete	Triggered after the CommitTrans operation
ConnectComplete	Triggered after a connection starts
Disconnect	Triggered after a connection ends
ExecuteComplete	Triggered after a command has finished executing
RollbackTransComplete	Triggered after the RollbackTrans operation
WillConnect	Triggered before a connection starts
WillExecute	Triggered before a command is executed

➤ **Command :**

- The SqlCommand class is at the heart of the Data Provider's namespace. It is used to execute operations on a database and retrieve data.
- This is specially used under Connected Architecture. Command allows you to specify different types of SQL Commands like Insert, Update, Delete, Select, etc. It has several methods which help you to manipulate or fetch the data from database.
- While using Command object, it is necessary to open the connection using Open() method before attempting any operation. It will not open the connection automatically like Disconnected Architecture. The same way after operation is done, you need to close the connection using Close() method.
- **Properties of Command Object**

Property	Description
CommandText	Contains the text of a SQL query
CommandTimeout	Contains the length of the timeout of a query, in seconds
CommandType	Specifies the type of command to be executed
Connection	Specifies the connection to the database
Parameters	Specifies a collection of parameters for the SQL query
Transaction	Specifies a transaction object, which enables developers to run queries in a transaction

- **Methods of Command Object**

Method	Description
Cancel()	Cancels the running query
CreateParameter()	Returns a new SQL parameter
ExecuteNonQuery()	Executes the CommandText property against the database and does not return a result set. This method is used in case you have given Transactional Commands like UPDATE, DELETE, etc. This method returns integer value as answer. If value is greater than 0, this means command executed successfully. 0 means it did not have any effect.
ExecuteReader()	Executes the CommandText property and returns data in a DataReader object. This method is used if you have specified SELECT. This allows command object to connect with DataReader which helps you to read the data.
ExecuteScalar()	Executes the CommandText property and returns a single value. This method is also used when you have specified SELECT command. But it returns only one value. It returns value of First Row's First Column. For example your command was "Select * from student", even though entire table will give you value of first student's first column.

- **Event of Command Object**

Event	Description
StatementCompleted	This event occurs when the Transactional-SQL command is finished at the database end.

➤ **DataReader :**

- The DataReader object defines a lightweight yet powerful object that is used to read information from a database.
- This is used under Connected Architecture. DataReader is used with Command object in order to read (fetch) the data. But remember that DataReader can be used to only read the data in sequential manner (forward only manner). You can not go back or go on particular record directly.
- **Properties of DataReader Object**

Property	Description
FieldCount	Contains the number of fields retrieved from the query
IsClosed	Contains True if the DataReader object is closed
Item	Contains A collection of values that are accessible both by field name and by ordinal number
RecordsAffected	Returns the number of records affected by an executed query
HasRows	Gets a value that indicates whether the DataReader contains one or more rows.

Methods of DataReader Object

Method	Description
Close()	Closes the DataReader object
IsDBNull()	Returns True if the field contains Null
Read()	Reads the next result in the result set into memory . The Read method of DataReader object is used to obtain a row from the results of the query. Each column of the returned row may be accessed by passing the name or reference of the column to the DataReader,

➤ **DataAdapter :**

- DataAdapter is used under Disconnected Architecture. It plays an important role in disconnected architecture. It allows you to fetch the data from database to local memory of client and also to save the updated data back to database.
- In Disconnected Architecture we use DataSet or DataTable to fill the data which resides in local memory. DataAdapter object does two important tasks as follows
- Fill data from Database to DataSet or DataTable
- Update the data back to Database
- DataAdapter fills the specified data into DataSet or DataTable which is in our (client's) local memory. After filling the data connection is automatically terminated (closed) with DataAdapter. But now the data has been loaded into DataSet or DataTable. All types of operations like INSERT, UPDATE, DELETE, SELECT, etc. are performed locally in DataSet or DataTable.
- **Properties of DataAdapter Object**

Property	Description
SelectCommand	This property works in background which generates your Select command automatically. OleDbCommandBuilder class works behind this to generate Select command automatically.
DeleteCommand	This property also works in background which generates your Delete command. If you have deleted some rows from DataSet or DataTable, OleDbCommandBuilder automatically generates Delete Command and stores in this.
InsertCommand	This property also works in background which generates Insert Command. If you insert any new row in DataSet or DataTable, OleDbCommandBuilder automatically generates Insert command and stores in this.
UpdateCommand	This property also works in background which generates Update Command. If you have done some changes in DataSet or DataTable data, OleDbCommandBuilder automatically generates Update command and stores in this.

○ **Methods of DataAdapter Object**

Method	Description
Fill	Most important method of DataAdapter. This method helps you to fill the data into DataSet or DataTable. Remember that in DataAdapter we can give only SELECT command which will be executed and data will be loaded into DataSet or DataTable. The Fill method is probably the DataAdapter method you will use most frequently. As stated, the Fill method adds data from your data source to a dataset. The Fill method also

	variety of parameters including the DataSet object to fill, a string representing the alias of the newly created DataSet object, an integer representing the lower bound of records to retrieve and an integer representing the upper bound of records to retrieve from our data source.
Update	<p>Most important method of DataAdapter. This method is used to save the changes back to the DataSet / DataTable to actual database table.</p> <p>Before using Update() method you need to use SqlCommandBuilder class so that it can generate related INSERT, UPDATE, DELETE, ALTER, etc. command automatically.</p> <p>The Update method calls the respective insert, update, or delete command for each inserted, updated, or deleted row in the DataSet. There are three different ways to call the Update method—you can pass:</p> <ul style="list-style-type: none"> An array of DataRow objects A DataSet object A DataSet object and a string representing a table name

○ Events of DataAdapter Object

Event	Description
FillError	This event is raised when there is a problem in filling data into DataSet or DataTable.
RowUpdating	This event is raised while changes are being made in Row. When you call Update() method, DataAdapter makes changes to actual database table. At that time, while being updated, for each row this event is raised.
RowUpdated	This event is raised after changes are done in Row. When you call Update() method, DataAdapter makes changes to actual database table. After changes have been done, this event is raised.

➤ DataSet :

- The DataSet object represents an in-memory group of database tables. You can have more than one tables under DataSet. It is collection of DataTables.
- The DataSet is the main in disconnected, data-driven application; it is an in-memory representation of a complete set of data, including tables, relationships, and constraints. The DataSet does not maintain a connection to a data source, enabling true disconnected data management.
- The data in a DataSet can be accessed, manipulated, updated, or deleted, and then reconciled with the original data source. Since the DataSet is disconnected from the data source, there is less contention for valuable resources, such as database connections, and less record locking.
- **Properties of DataSet Object**

Property	Description
DataSetName	Gets or sets the name of the current DataSet.
HasErrors	Gets a value indicating whether there are errors in any of the DataTable objects in this DataSet.
IsInitialized	Gets a value that indicates whether the DataSet is initialized.

Relations	Get the collection of relations that link tables and allow navigation from parent to child tables.
Tables	Gets the collection of tables contained in the DataSet.

➤ **Methods of DataSet Object**

Method	Description
AcceptChanges()	Commits all the changes made to this dataset since the last time AcceptChanges was called.
Clear()	Clears all the DataTables of DataSet
ReadXml()	Reads XML schema and data into DataSet using specified Stream or File
ReadXmlSchema()	Reads an XML schema into the DataSet using the specified stream, or File
RejectChanges	Rolls back all changes that have been made to the dataset since it was loaded last time AcceptChanges was called.
Select	Allows you to select group of data from dataset. It has different methods to select data. This is overloaded method which allows you to filter the data in variety of ways.
WriteXml()	Writes the current contents of the DataSet as XML using the specified Stream or File
WriteXmlSchema()	Writes the current data structure of the DataSet as an XML schema to the specified stream or file.

Events of DataSet Object

Method	Description
Initialized	Occurs after the DataSet is initialized.

➤ **DataTable :**

- The DataTable object represents an in-memory database table. You can add rows to a DataTable with a DataAdapter.
- A DataTable is defined in the "System.Data" namespace and represents a single table of memory-resident data. It contains a collection of columns represented by the DataColumnCollection, which defines the schema and rows of the table. It also contains a collection of rows represented by the DataRowCollection, which contains the data in the table. Along with the current state, a DataRow retains its original state and tracks changes that occur to the data.
- The DataTable class is a central class in the ADO.NET architecture; it can be used independently, and in DataSet objects. A DataTable consists of a Columns collection, a Rows collection, and a Constraints collection. The Columns collection combined with the Constraints collection defines the DataTable schema, while the Rows collection contains the data.
- **Properties of DataTable Object**

Property	Description
Columns	The Columns collection is an instance of the DataColumnCollection class, a container object for zero or more DataColumn objects. The DataColumnCollection

	define the DataTable column, including the column name, the data type, and primary key or incremental numbering information.
Constraints	The Constraints collection is an instance of the ConstraintCollection class, and is a container for zero or more ForeignKeyConstraint objects and/or UniqueConstraint objects. The ForeignKeyConstraint object defines the action to be taken on a row in a primary key/foreign key relationship when a row is updated or deleted. The UniqueConstraint is used to force all values in a column to be unique.
DataSet	Gets the DataSet to which this table belongs.
HasErrors	Gets a value indicating whether there are errors in any of the rows in any tables of the DataSet to which the table belongs.
IsInitialized	Gets a value that indicates whether the DataTable is initialized.
PrimaryKey	Gets or sets an array of columns that function as primary keys for the data table.
Rows	The Rows collection is an instance of the DataRowCollection class, and is a container for zero or more DataRow objects. The DataRow object contains the data for a row in the DataTable, as defined by the DataTable.Columns collection. Each DataRow object contains one item per DataColumn in the Columns collection.
TableName	Gets or sets the name of the DataTable.

○ **Methods of DataTable Object**

Method	Description
AcceptChanges()	Commits all the changes made to this table since the last time AcceptChanges was called.
Clear()	Clears the DataTable of all data.
NewRow()	Creates a new DataRow with the same schema as the table.
ReadXml()	Reads XML schema and data into the DataTable using the specified Stream or File.
ReadXmlSchema()	Reads an XML schema into the DataTable using the specified stream. or File.
RejectChanges	Rolls back all changes that have been made to the table since it was loaded. The last time AcceptChanges was called.
Select	Allows you to select group of data from table. It has different methods to select data. This is overloaded method which allows you to filter the data in variety of ways.
WriteXml()	Writes the current contents of the DataTable as XML using the specified Stream or File.
WriteXmlSchema()	Writes the current data structure of the DataTable as an XML schema to the specified stream or file.

○ **Events of DataTable Object**

Event	Description
Initialized	Occurs after the DataTable is initialized.

Event	Description
RowChanged	Occurs after a DataRow has been changed successfully.
RowChanging	Occurs when a DataRow is changing.
RowDeleted	Occurs after a row in the table has been deleted.
RowDeleting	Occurs before a row in the table is about to be deleted.
TableCleared	Occurs after the Table is cleared.
TableClearing	Occurs when then Table is being cleared.
TableNewRow	Occurs when you insert a new Row in table.

➤ **DataRow :**

- DataRow can contain any single row of any table. Whenever you store any DataTables single row into object of DataRow, it automatically creates its column as per source table.
- For example, your DataTable object has 6 columns, when you copy one row of DataTable into DataRow object, DataRow object automatically creates 6 columns for itself as per DataTable object.
- To populate a DataTable we add new DataRow objects to the DataTable.Rows collection. Each DataRow can reference each DataColumn in the DataTable schema. To create a new row in the DataTable, we invoke the DataTable.NewRow method, which returns a DataRow using the DataTable's current schema.
- **DataRow** objects represent rows in a **DataTable** object. You use **DataRow** objects to get access to, insert, delete, and update the records in a table.
- To create a new **DataRow** object, you usually use the **NewRow** method of a **DataTable** object, and after configuring the row with data, you can use the **Add** method to add the new **DataRow** to the table. In addition, you also can call the **AcceptChanges** method of the **DataTable** object to make that table treat the new row as it would its original data.
- You can delete a **DataRow** from the **Rows** collection in a data table by calling the **Remove** method, or by calling the **Delete** method of the **DataRow** object itself. Note that the **Remove** removes the row from the collection, and the **Delete** method simply marks the **DataRow** for deletion. (The actual deletion occurs when you use the **AcceptChanges** method.)

○ **Properties of DataRow Object**

Property	Description
HasErrors	Indicates if there are errors in the row.
Item	Gets/sets data in a specified column.
RowError	Gets/sets a row's error description.
RowState	Gets the current state of a row.
Table	Gets the table that contains this row.

○ **Methods of DataRow Object**

Method	Description
AcceptChanges	Accepts (commits) the changes made to the row.

RejectChanges	Rolls back the changes made to the table since it was created or since the AcceptChanges method was called.
BeginEdit	Begins an edit operation.
EndEdit	Ends the current edit operation.
CancelEdit	Cancels the current edit operation.
ClearErrors	Clears the errors in the row.
Delete	Deletes the row.
IsNull	Indicates if a column contains a null value.

➤ **DataColumn :**

- **DataColumn** objects represent the columns, that is, the fields, in a data table. In ADO.NET terms, the columns in a table specify its XML schema. When you create a table and add columns to it, you specify the name of the column and the type of data it stores.
- DataRow is collection of DataColumns. One DataRow can have multiple columns within it because always a row contains multiple columns.
- **Properties of DataColumn Object**

Property	Description
AllowDBNull	Gets/sets if null values are allowed.
AutoIncrement	Gets/sets if the column automatically increments the column's value when rows are added to the table.
AutoIncrementSeed	Gets/sets the starting value for an autoincrement column.
AutoIncrementStep	Gets/sets the increment for an autoincrement column.
Caption	Gets/sets the caption for the column.
ColumnName	Gets/sets the name of the column.
DataType	Gets/sets the type of data in the column.
DefaultValue	Gets/sets the default value for the column (used in new rows).
MaxLength	Gets/sets the maximum length of a text column.
ReadOnly	Gets/sets if the column is read-only.
Table	Gets the table the column belongs to.
Unique	Gets/sets if the values in this column must be unique.

- **Methods of DataColumn Object**

Method	Description
ToString	Gets the Expression value for this column, if there is one.
GetType	Gets the type of object.

➤ **DataView :**

- When you bind to a DataTable, you actually use another object called the DataView. The DataView sits between the ASP.NET web page binding and your DataTable. Usually it does little aside from providing the information from the associated DataTable. However, you can customize the DataView so it applies its own sort order. That way, you can customize the data that appears in the web page, without needing to actually modify your data.
- You can create a new DataView object by hand and bind the DataView directly to a data control.
- Data views are much like read-only mini-datasets; you typically load only a subset of a dataset into a data view.
- Data views represent a customized view of a single table that can be filtered, searched, or sorted. In other words, a data view, supported by the DataView class, is a data "**snapshot**" that takes up few resources.
- To create a filtered and sorted view of data, set the **RowFilter** and **Sort** properties. Then use the **Item** property to return a single **DataRowView**.
- **Properties of DataView Object**

Property	Description
AllowDelete	Sets or gets a value that indicates whether deletes are allowed.
AllowEdit	Gets or sets a value that indicates whether edits are allowed.
AllowNew	Gets or sets a value that indicates whether the new rows can be added by using the AddNew method.
Count	Gets the number of records in the DataView after RowFilter and RowStateFilter have been applied.
IsOpen	Gets a value that indicates whether the data source is currently open and providing views of data on the DataTable.
Item	Gets a row of data from a specified table.
RowFilter	Gets or sets the expression used to filter which rows are viewed in the DataView.
Sort	Gets or sets the sort column or columns, and sort order for the DataView.
Table	Gets or sets the source DataTable.

- **Methods of DataView Object :**

Method	Description
AddNew	Adds a new row to the DataView.
Close	Closes the DataView.
Delete	Deletes a row at the specified index.
Open	Opens a DataView.

- **Events of DataView Object**

Events	Description
ListChanged	Occurs when the list managed by the DataView changes.

➤ **GridView :**

- The GridView is an extremely flexible grid control that displays a multicolumn table. Each record in your data source becomes a separate row in the grid. Each field in the record becomes a separate column in the grid.
- GridView control is a grid control which displays data in a tabular format.
- That means you can bind table with GridView which can be displayed under GridView.
- Each record of table becomes row of GridView and each column of table becomes column of GridView control.
- GridView has inbuilt features that allows to Edit, Delete or Insert data directly into GridView.
- The GridView control looks like follows :

Column 0	Column 1	Column 2
Abc	Abc	abc
Abc	Abc	abc
Abc	Abc	abc
Abc	Abc	abc

- Following is the list of properties supported by GridView control :

Property	Description
AllowPaging	Used to set paging if table data is too long. By default "false".
AllowSorting	Used to sort the data in GridView. By default "false".
PageSize	Used to set the size of page if AllowPaging is "true". By default is 10 rows per page.
AutoGenerateDeleteButton	Displays Delete link along with each rows of GridView if set to "true".
AutoGenerateEditButton	Displays Edit link along with each rows of GridView if set to "true".
AutoGenerateSelectButton	Displays Select link along with each rows of GridView if set to "true".
Caption	Used to set heading for GridView.
DataSource	Runtime property used to display data from table.

- GridView supports following Methods :

Method	Description
DataBind() :	Used to bind data with specified data source.
DeleteRow() :	Deletes specified row from GridView.
UpdateRow() :	Updates specified row from GridView.
Sort() :	Sorts GridView data depending on the two parameters passed : 1. Sort Expression 2. Sort Direction

- GridView has define following Events :

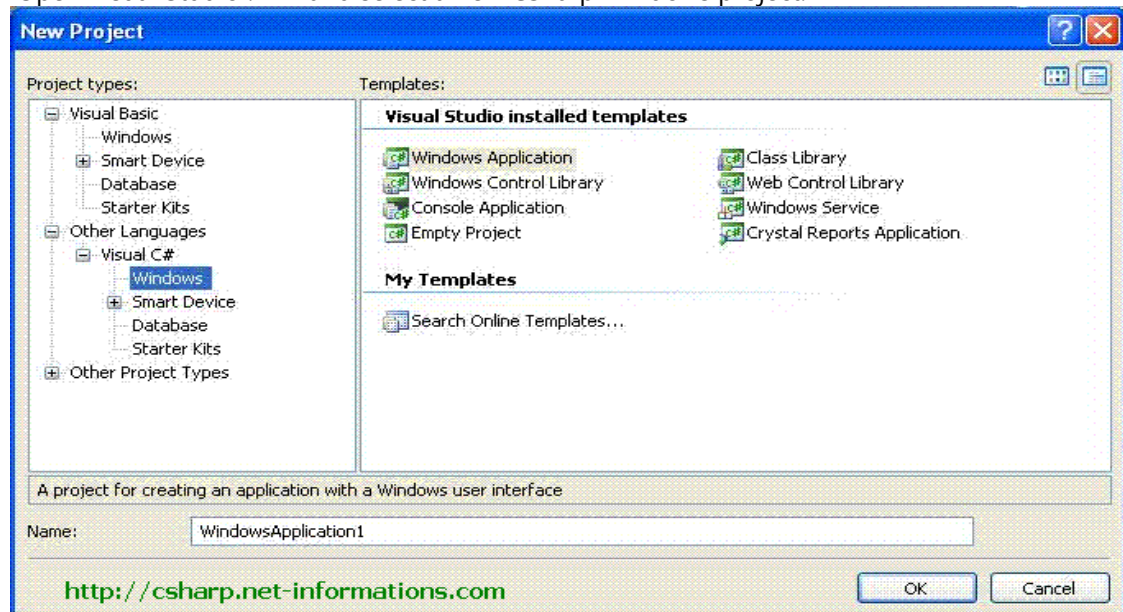
Event	Description
-------	-------------

PageIndexChanged	Occurs if AllowPaging property is set to "true" and page index is chaged.
SelectedIndexChanged	Raised after index of selected link button is chaged in GridView control.

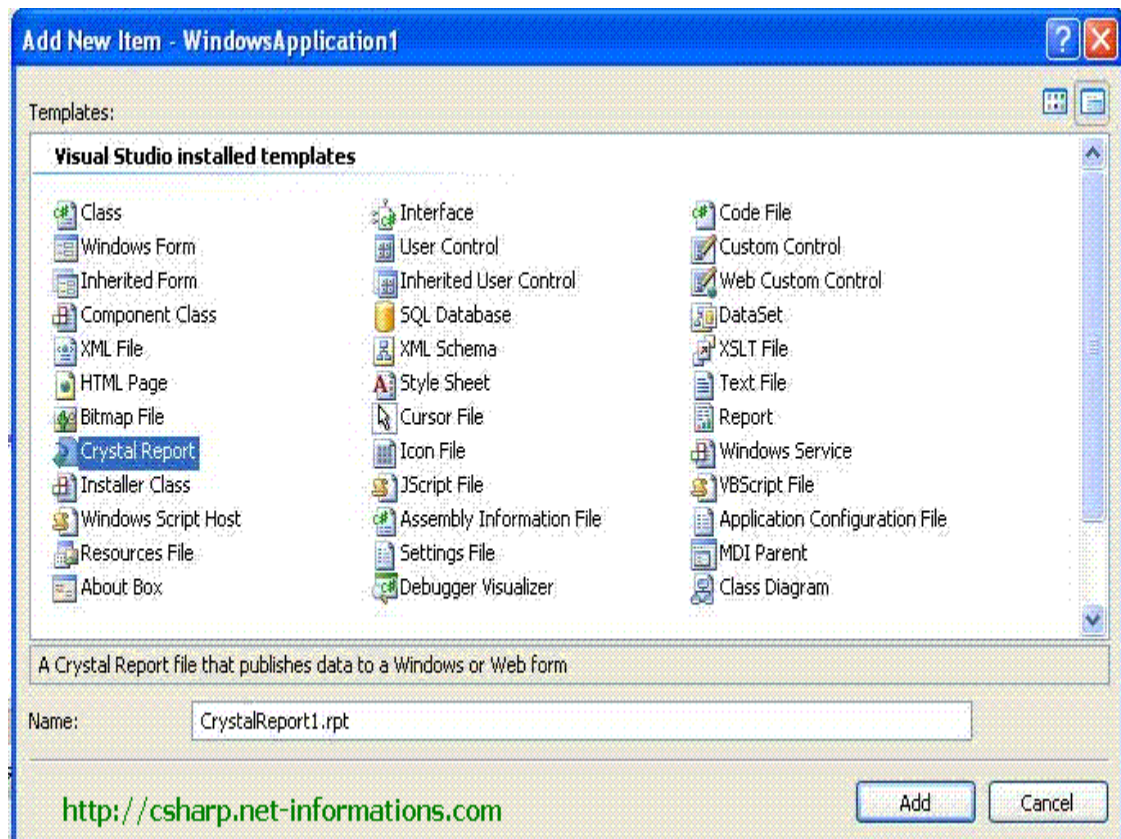
Chapter-5:- User Controls(Components),Crystal Report, Set up Projects

➤ **Creating Crystal Reports:-**

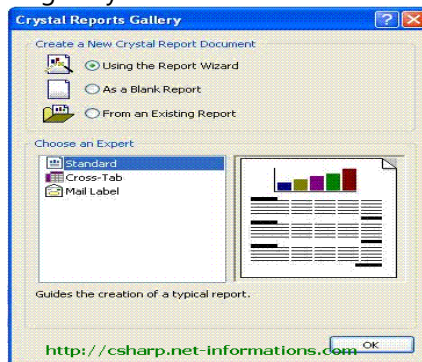
- ❖ **Crystal Report** is a Reporting Application that can generate reports from various Data Sources like Databases , XML files etc.
- ❖ The Visual Studio.NET Integrated Development Environment comes with Crystal Reports tools.
- ❖ The Crystal Reports makes it easy to create simple reports, and also has comprehensive tools that you need to produce complex or specialized reports in csharp and other programming languages.
- ❖ Crystal Reports is compatible with most popular development environments like **C# , VB.NET** etc.
- ❖ You can use the Crystal Reports Designer in Visual Studio .NET to create a new report or modify an existing report.
- ❖ Open Visual Studio .NET and select a new CSharp Windows project.



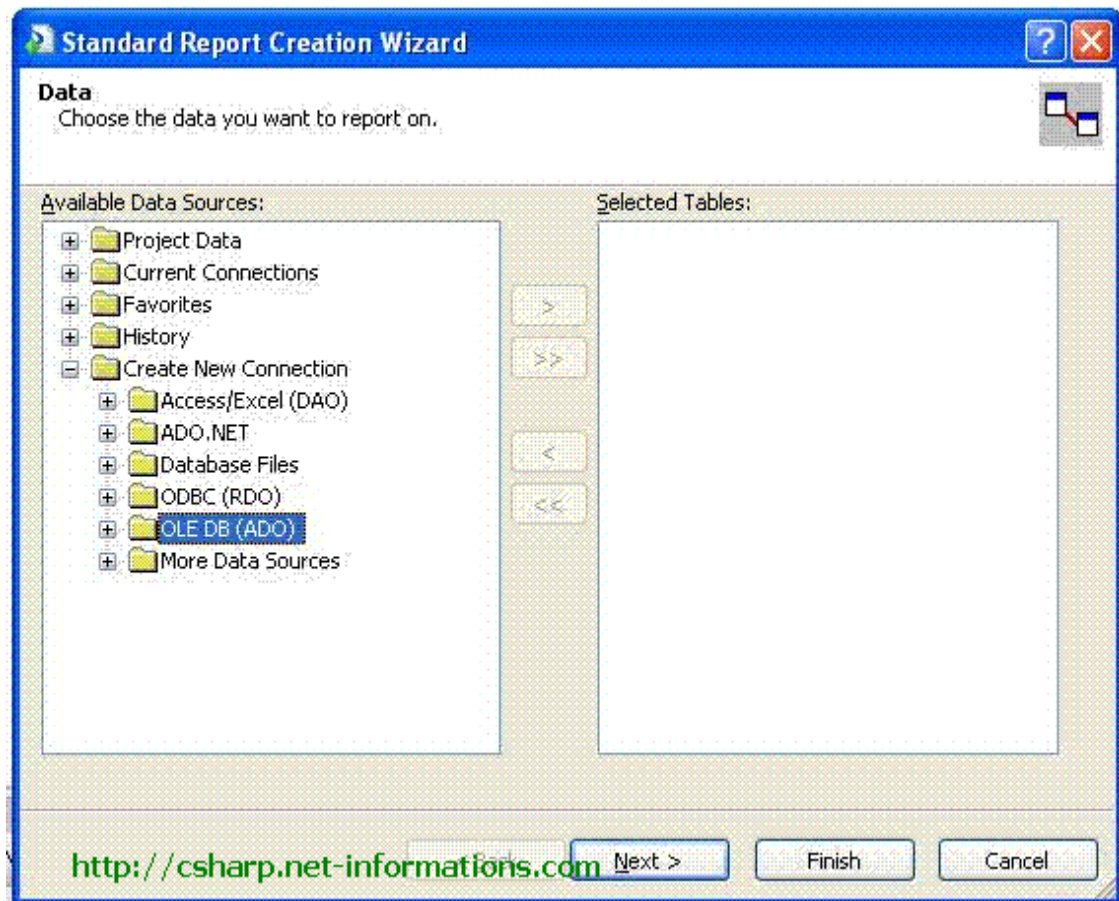
- Now you will get the default Form1.cs.
- From the main menu in Visual Studio C# project select PROJECT-->Add New Item . Then Add New Item dialogue will appear and select Crystal Reports from the dialogue box.



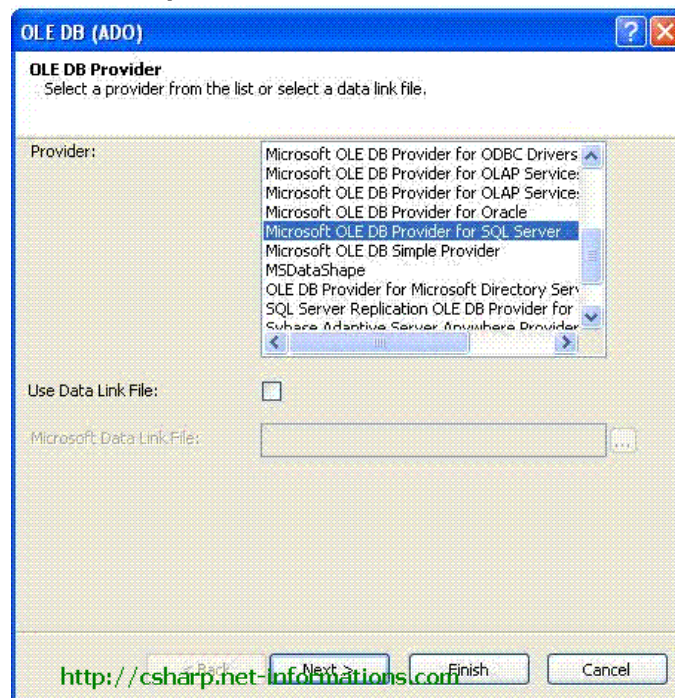
- Select Report type from Crystal Reports gallery.



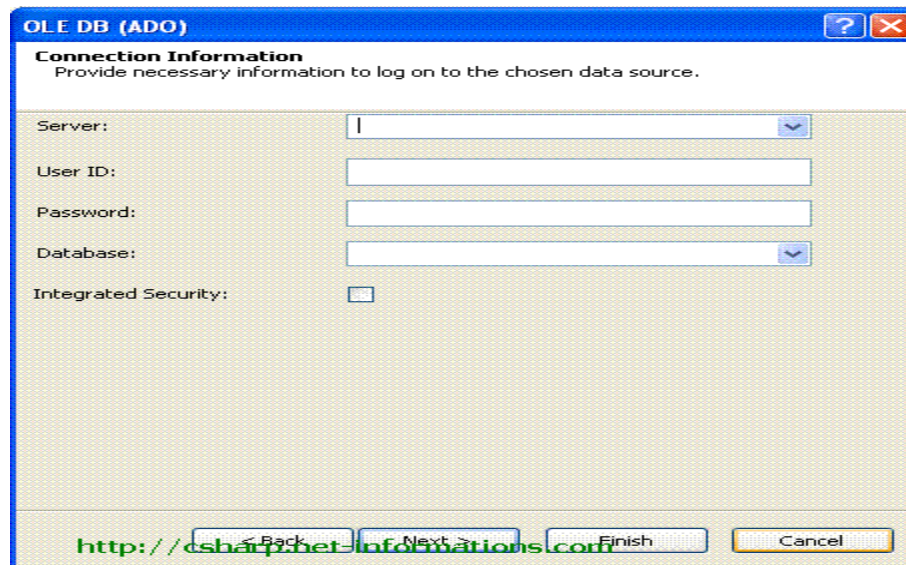
- Accept the default settings and click OK.
- Next step is to select the appropriate connection to your database (here crstalddb). Here we are going to select **OleDb Connection** for SQL Server to connect Crystal Reports in C#.
- Select OLE DB (ADO) from Create New Connection .



- Select Microsoft OLE DB Provider for SQL Server .



- The next screen is the SQL Server authentication screen for connecting to the database - crystalDB. Select your Sql Server name , enter userid , password and select your Database Name .



OLE DB (ADO)

Connection Information
Provide necessary information to log on to the chosen data source.

Server:

User ID:

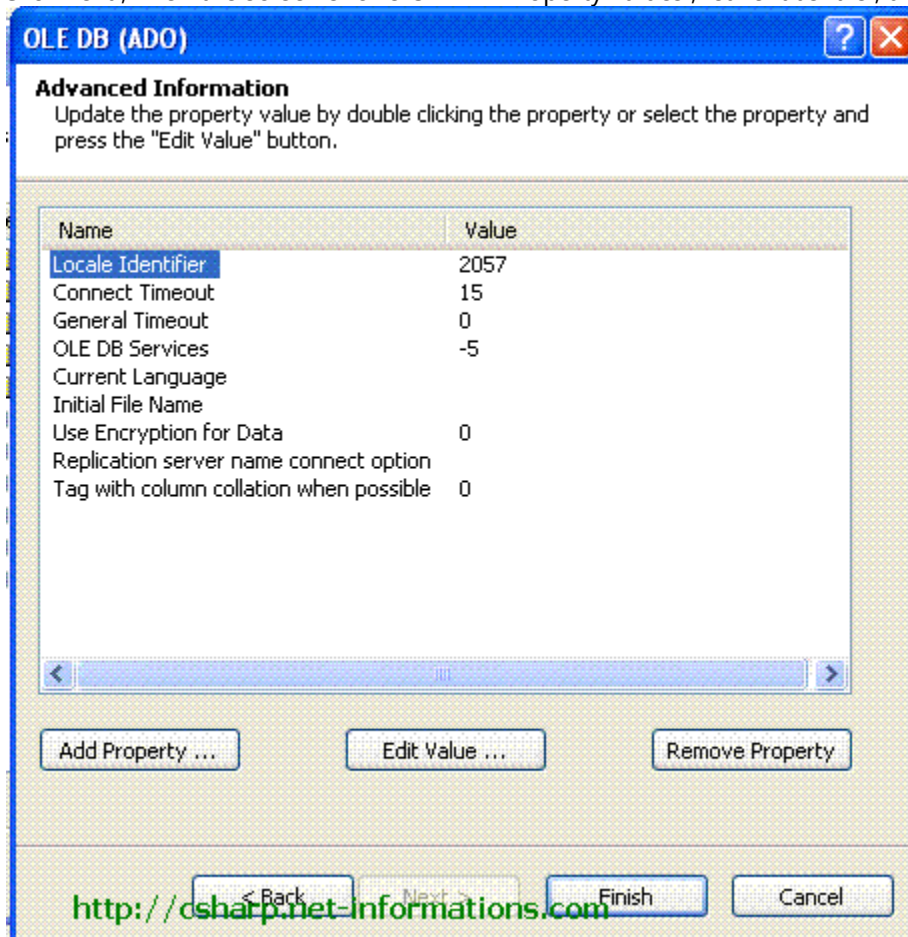
Password:

Database:

Integrated Security: ☐

<http://csharp.net-informations.com>

- Click next , Then the screen shows OLE DB Property values , leave it as it is , and then click finish button.



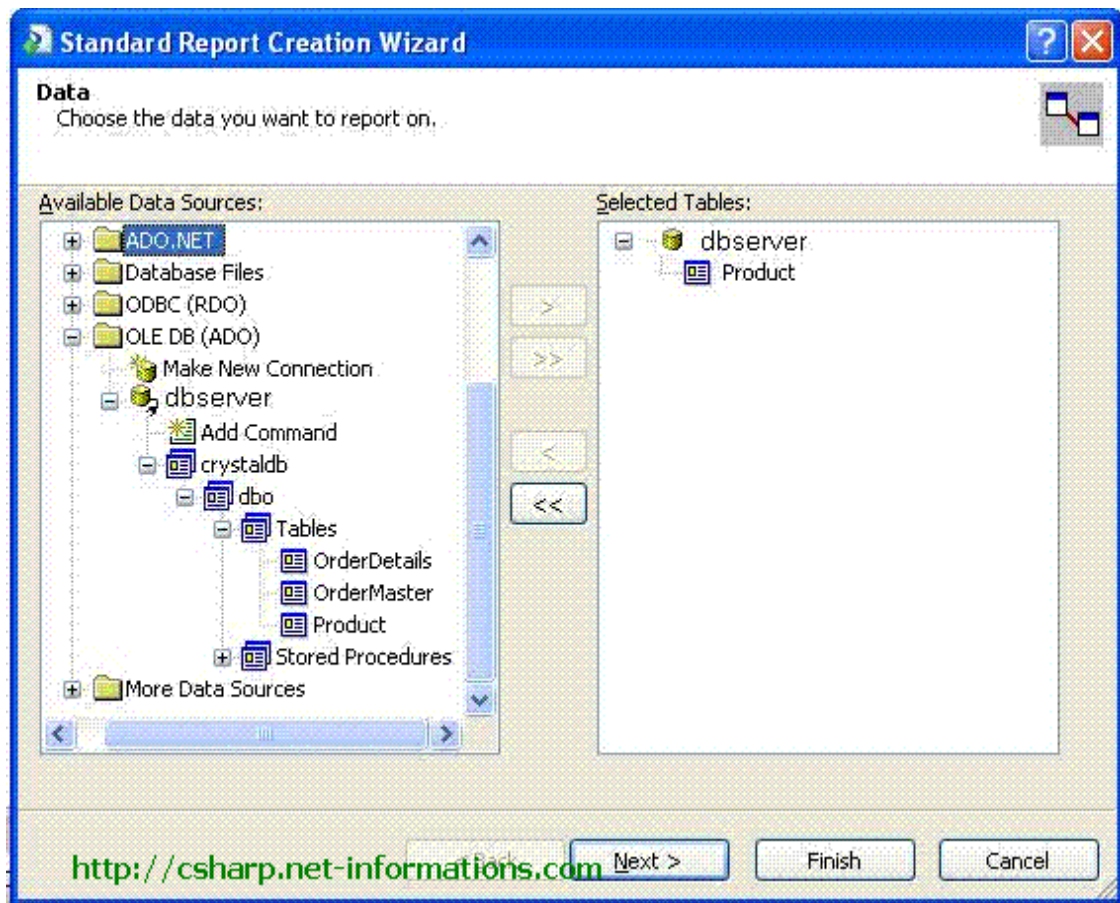
OLE DB (ADO)

Advanced Information
Update the property value by double clicking the property or select the property and press the "Edit Value" button.

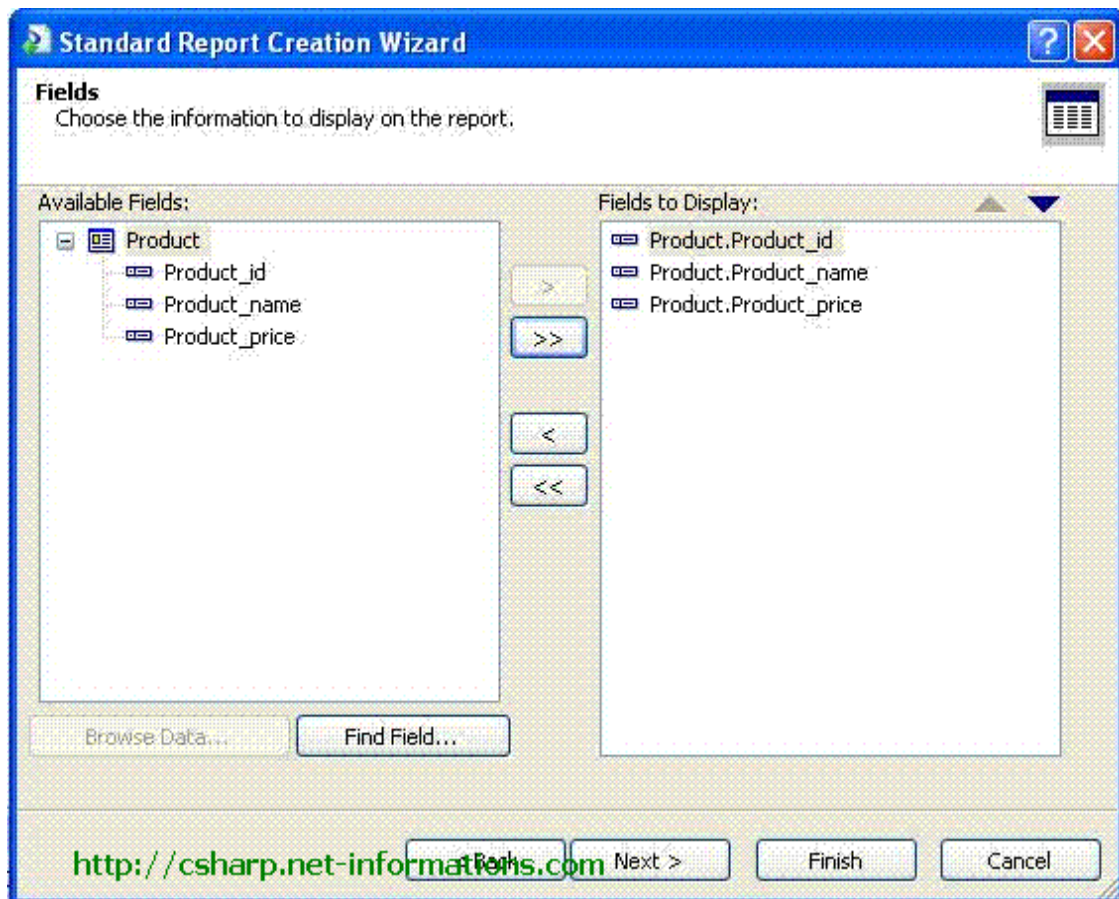
Name	Value
Locale Identifier	2057
Connect Timeout	15
General Timeout	0
OLE DB Services	-5
Current Language	
Initial File Name	
Use Encryption for Data	0
Replication server name connect option	
Tag with column collation when possible	0

<http://csharp.net-informations.com>

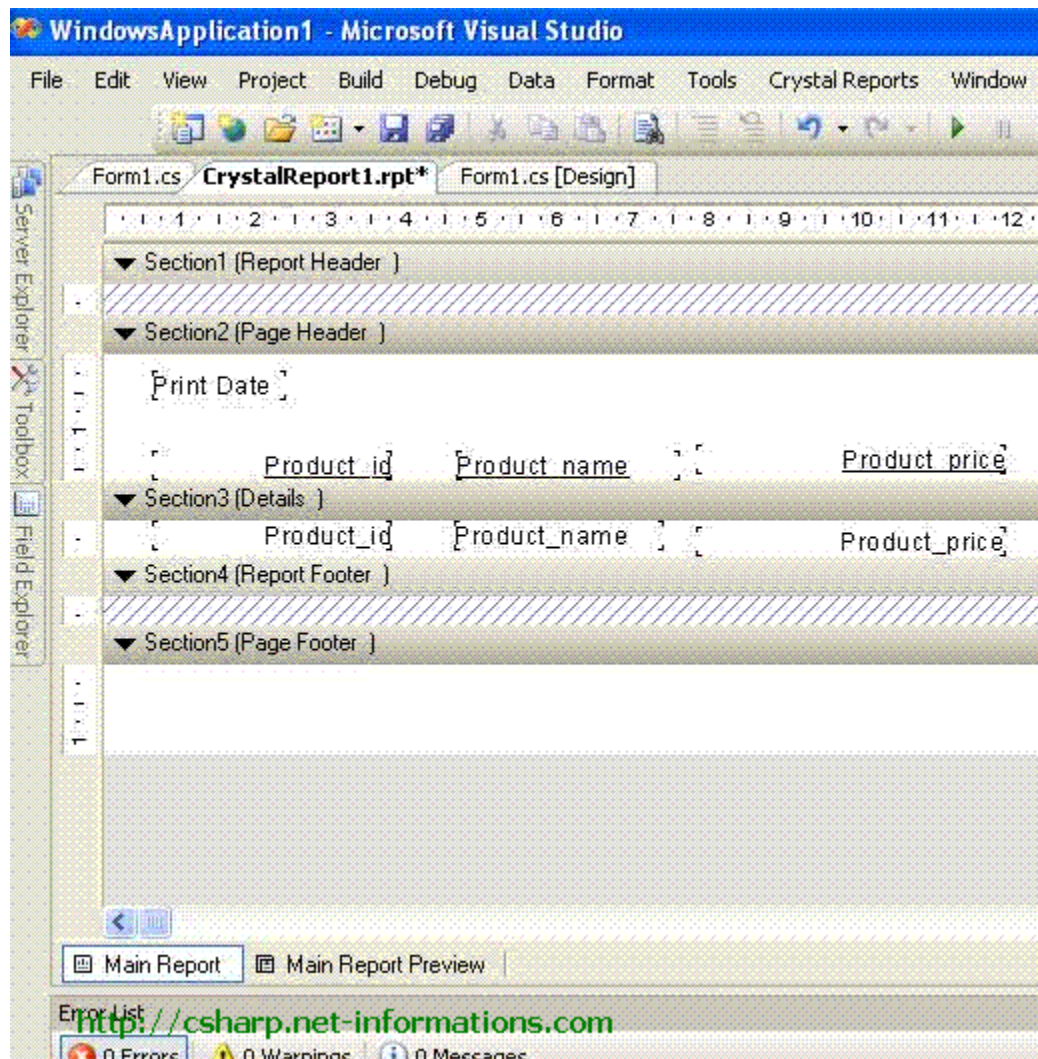
- After you click the finish button , the next window you will get your Server name under OLEDB Connection, from there selected database name (Crystalldb) and click the tables , then you can see all your tables from your database.
- From the tables list double click the Product table then you can see the Product table will come in the right side list.



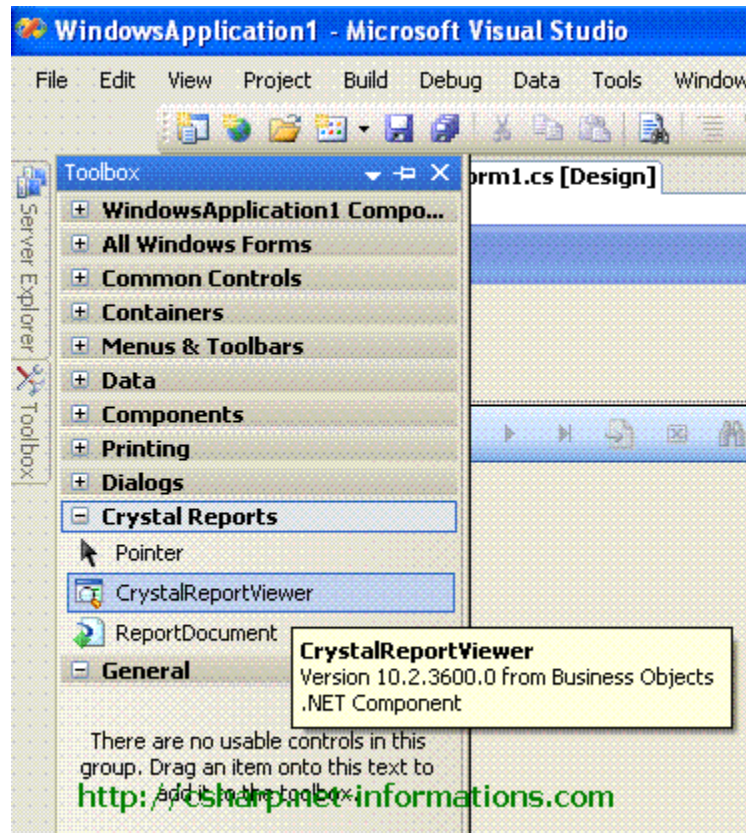
- Click Next Button
- Select all fields from Product table to the right side list .



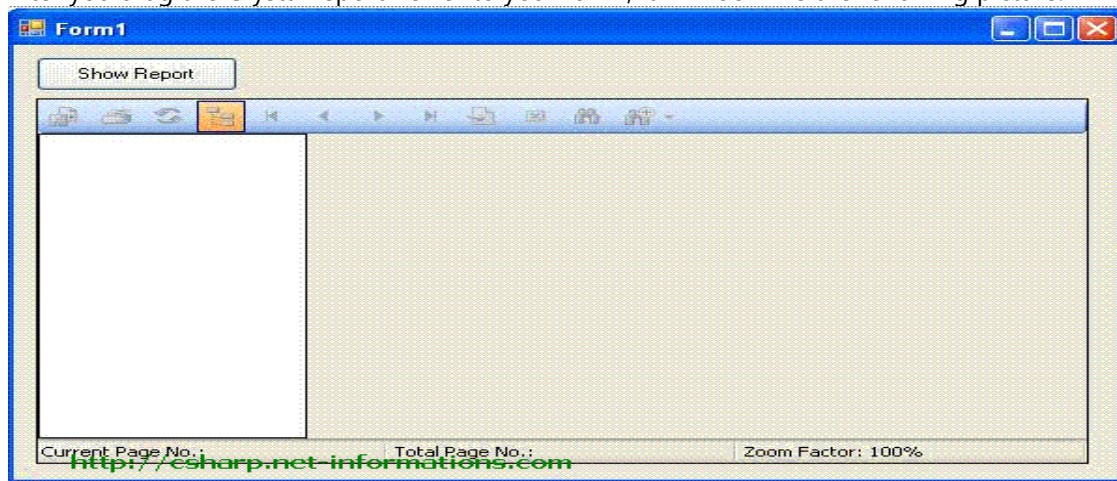
- Click Finish Button.
- Then you can see the Crystal Reports designer window in your C# project.
- In the Crystal Reports designer window you can see the selected fields from Product table.
- You can arrange the field Objects and design of the screen according your requirements. After that your screen is look like the following picture.



- Now the designing part is over and the next step is to call the Crystal Reports in your C# application and view it through Crystal Reports Viewer control in C#.
- Select the default form (Form1.cs) you created in C# and drag a button and a CrystalReportViewer control to your form .

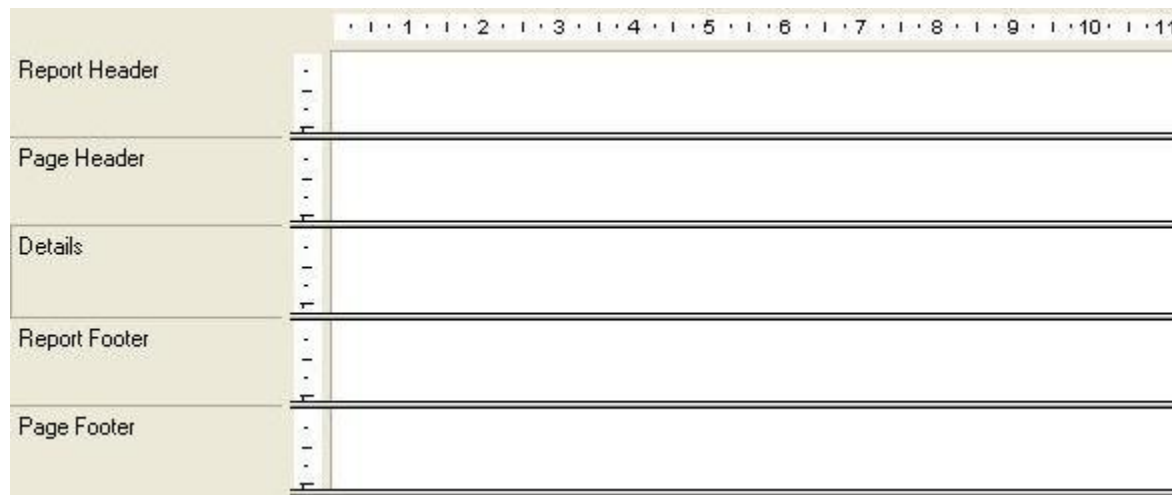


- After you drag the CrystalReportViewer to your form, it will look like the following picture.



➤ **Report Sections:-**

- ❖ Sections are the design areas which you use to build your report.
- ❖ Crystal Reports by default provides five main sections:
 - Report Header – fields placed in this section are printed once, at the beginning of the report
 - Page Header - fields placed in this section are printed at the beginning of each new page
 - Details – fields in this section are printed with each new record
 - Report Footer - fields placed in this section are printed once, at the end of the report
 - Page Footer - fields placed in this section are printed at the bottom of each new page



More sections will appear if you add groups or simply insert new one.

Working with sections

To insert a new section you need to:

- **Click Section Expert button in the top menu. Section Expert contains a list of all sections in the report**
- **Select section and click insert. New section will appear after the section you have selected. If there are more than one section of a kind, there appear with lettered a, b, c and so on**

To delete a section:

- **Open Section expert**
- **Select section you want to delete**
- **Click Delete. Note that you can only delete section if there are lettered. In mean that you cannot delete sections originally provided by Crystal Reports**

To move a section (change sections' order):

- **Open Section Expert**
- **Select section you want to move**
- **Use up and down arrows to change order of the sections. Although the alphabetic order of sections is the same, their data and display order is changed**

To merge sections you need to:

- **Open Section Expert**
- **Move sections so the sections you want to merge fallow each other**
- **Select the top section**
- **Click merge. Section will be merged with section that is next on the list**

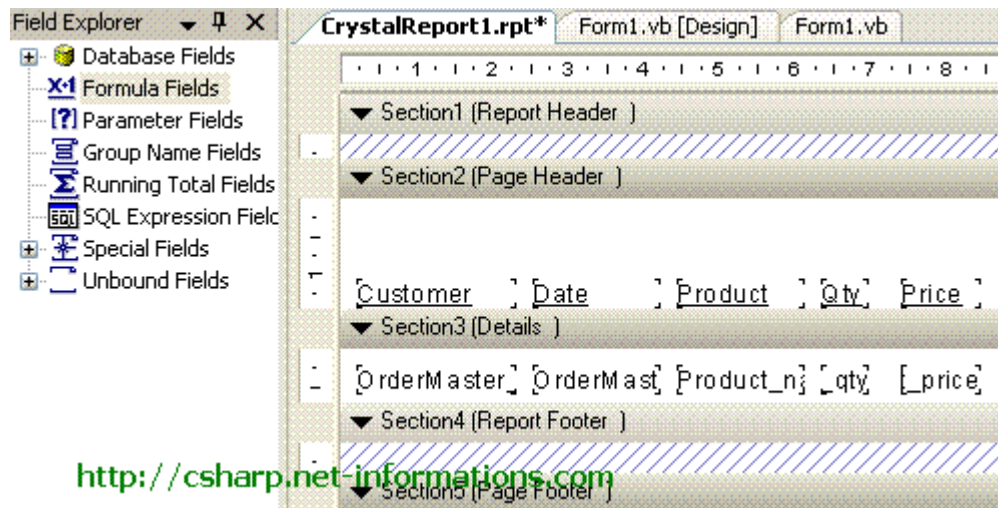
To split sections:

- **Click on the on the boundary of the section you want to split. Horizontal line that splits section will appear**
- **Drag-and-drop it to the place where you want to split section**

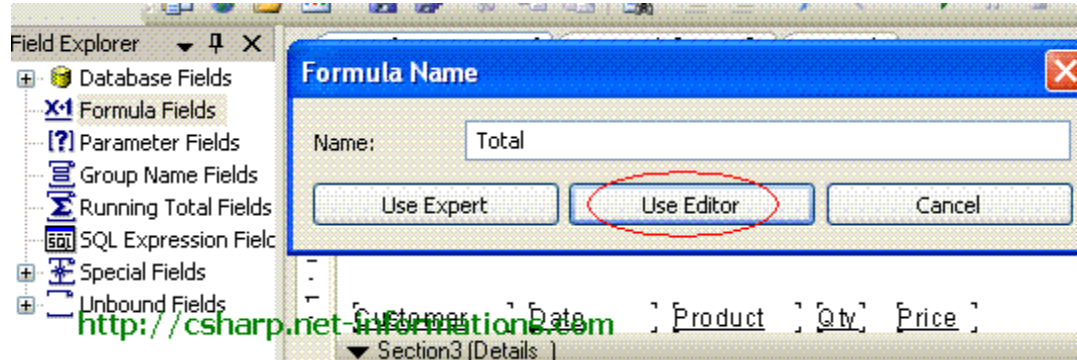
➤ Formula, Special Field and Summary in Report:-

Formula Field:

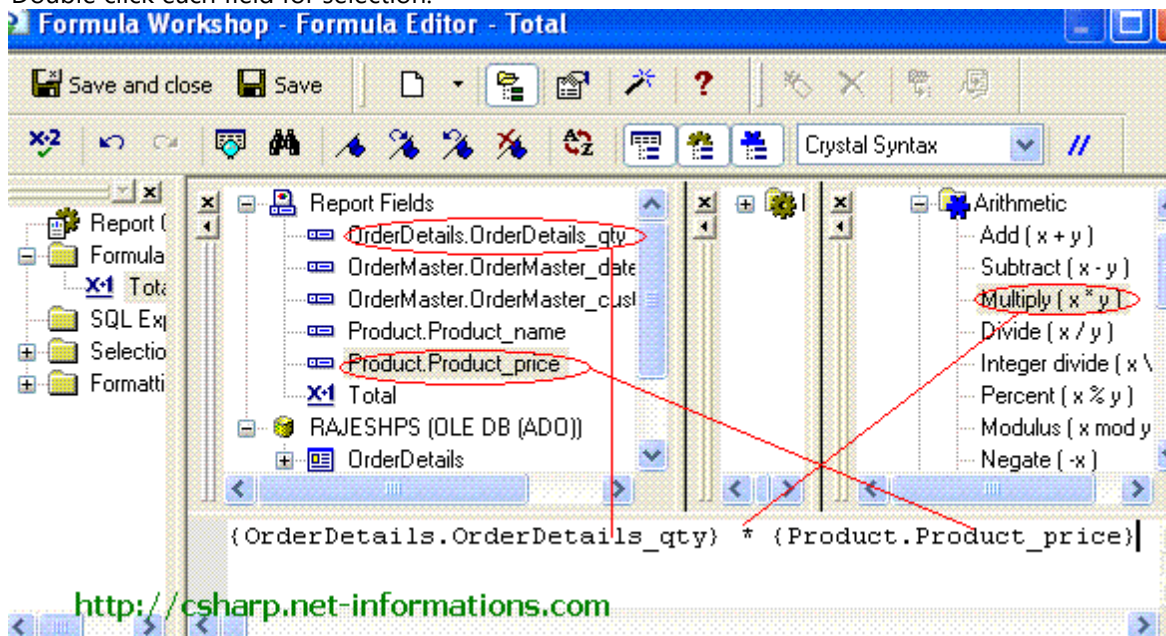
- ❖ The following C# - Crystal Reports section describes how to add a formula field in the Crystal Reports .



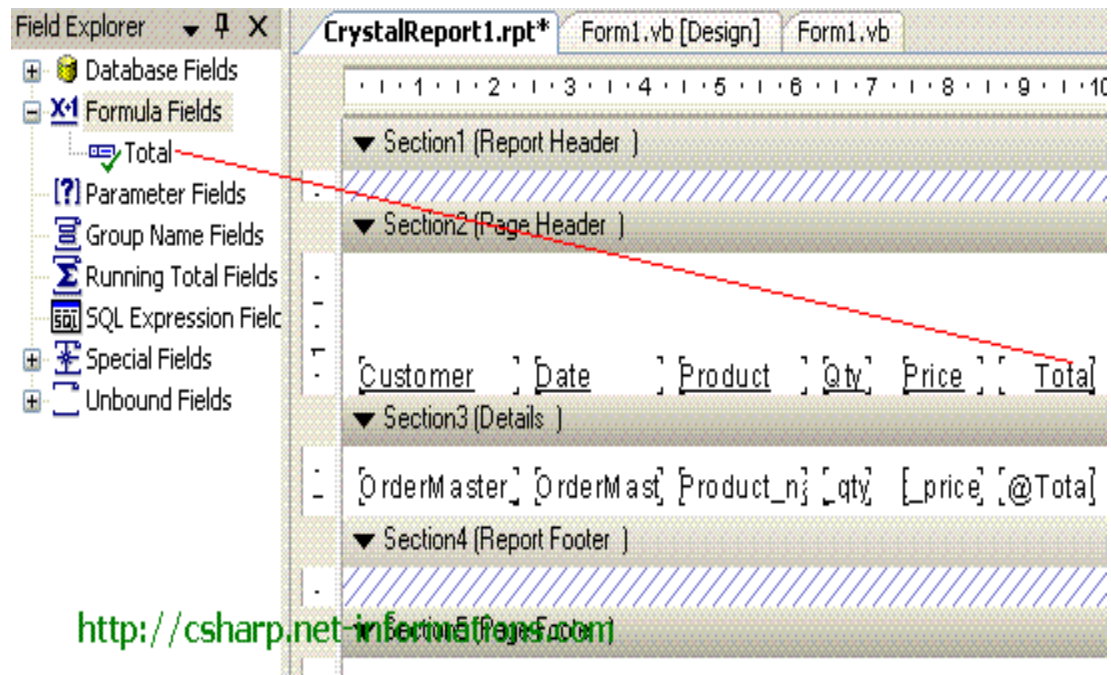
- ❖ Next step is to create a Formula Field for showing the result of **Qty X Price**.
- ❖ Right Click the Formula Field in the Field Explorer and click New. Then you will get an Input Message Box, type Total in textbox and click Use Editor.



- ❖ Now you can see the Formula Editor screen. Here you can enter which formula you want. Here we want the result of Qty X Price. For that we select OrderDetails.Qty, the multiply operator (*) and Product.Price. Double click each field for selection.



- ❖ Now you can see Total Under the Formula Field. Drag the field in to the Crystal Reports where you want to display Total.



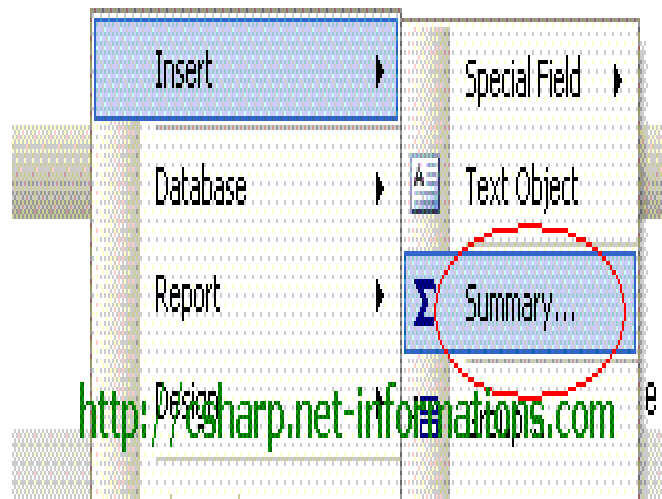
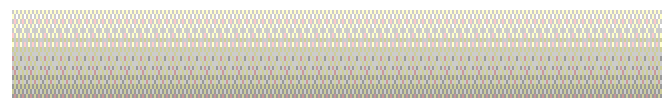
<http://csharp.net-informations.com>

Summary Field:

- ❖ The following C# - Crystal Reports section describes how to add a summary field in the Crystal Reports.
- ❖ In the Crystal Reports designer view window, right click on the Report Footer , just below the Total field and select **Insert -> Summary** .

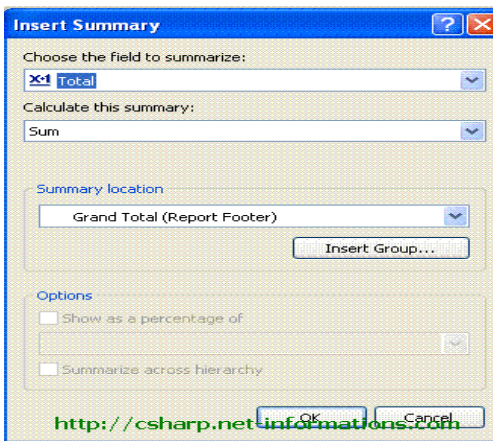


ice' [@Tota]

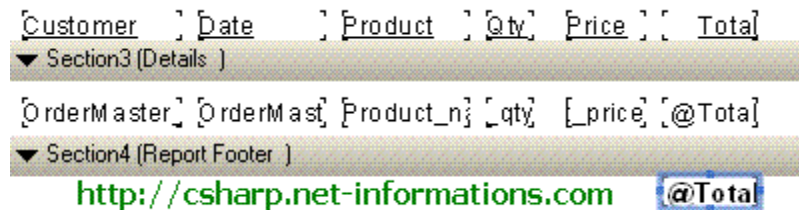


<http://csharp.net-informations.com>

- ❖ Then you will get a screen , select the Total from the combo box and select Sum from next Combo Box , and summary location Grand Total (Report Footer) . Click Ok button



- ❖ Now you can see @Total is just below the Total field in the report Footer.



➤ **Creating Setup Projects**

- ❖ Setup projects are used to create Windows Installer (.msi) files, which are used to distribute your application for installation on another computer or Web server. There are two types of setup projects. Standard setup projects create installers that install Windows applications on a target computer; Web setup projects create installers that install Web applications on a Web server.

Note:

The dialog boxes and menu commands you see might differ from those described in Help depending on active settings or edition. To change your settings, choose Import and Export Settings on the Tools menu. For more information, see [Working with Settings](#).

To create a new setup project

1. On the File menu, point to Add, then click New Project.
2. In the resulting Add New Project dialog box, in the Project Types pane, open the Other Project Types node, open Setup and Deployment Projects, and select Visual Studio Installer.
3. In the Templates pane, choose Setup Project for a standard setup, or Web Setup Project for a Web application.

To add an existing setup project to a solution

1. On the File menu, point to Add, then click Existing Project.
2. In the resulting Add Existing Project dialog box, browse to the location of the setup project and click Open.

➤ **File System Editor:**

- ❖ The **File System** editor presents a graphic representation of the files, folders, and shortcuts your product installation creates or modifies on the target machine. Use this editor to:
 - Add and delete files, folders, and shortcuts in the installer project.
 - Control where to install folders and files on the target machine.
 - Establish and modify the file structure of installer components.
- ❖ You can open this editor by either:

- Double-clicking **File System** in the **Target Machine** node in the **Project Explorer** window.
 - or–
 - Selecting the **File System** option on the **Project** menu.
- ❖ As you work in the **File System** editor, you can add, modify, or delete folders, files, and shortcuts with:
- The context menu.
- Note** : You can display the context menu by right-clicking in the **File System** editor. Context menu activated options differ depending on what is selected when you right-click.
- Options on the **Actions** menu.
- Note** **Actions** menu active options differ depending on your current file or folder selection.
- The DELETE key (for item removal).

Options

Left pane of File System editor

The left pane of the **File System** editor shows a folder view of the target machine. You can add and delete folders your product installs by setting them up here. You can also establish contents of target machine folders.

Microsoft® Visual Studio® Installer provides these initial folders for you to work with:

- **Application Folder**. Sets the root directory for your application on the target machine.
- **User's Desktop**. Place items (such as files or shortcuts) you want to install to the user's desktop in this folder.
- **User's Start Menu**. Place items (such as files or shortcuts) you want to install to the user's Start menu in this folder.

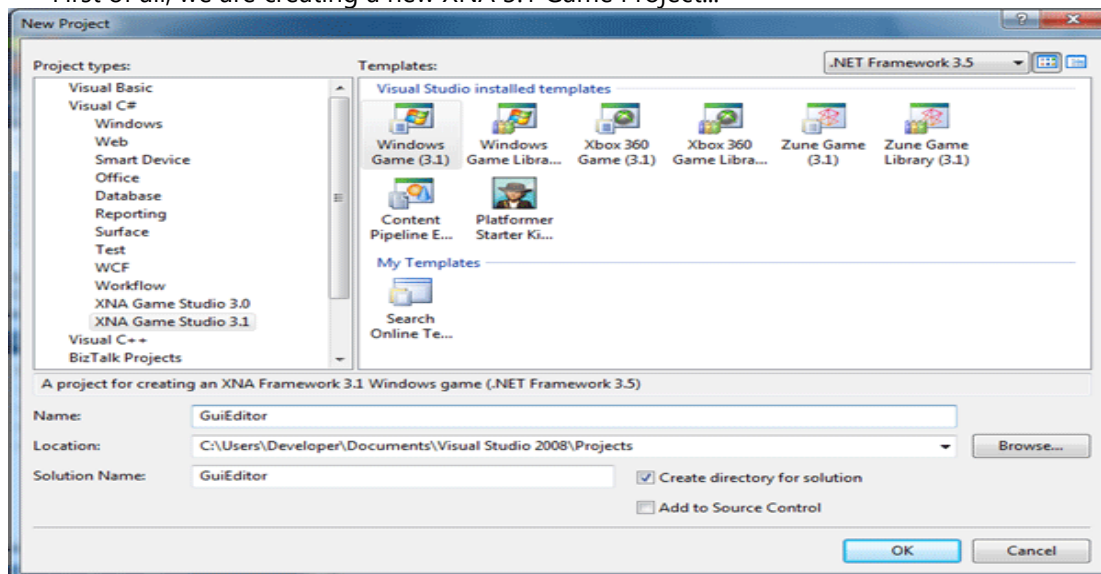
Note If the installer project contains a Microsoft® Visual Basic® project with dependencies that can't be mapped to merge modules, a fourth folder labeled **Windows System Folder** is displayed in the left pane of the **File System** editor.

➤ User Interface Editor:

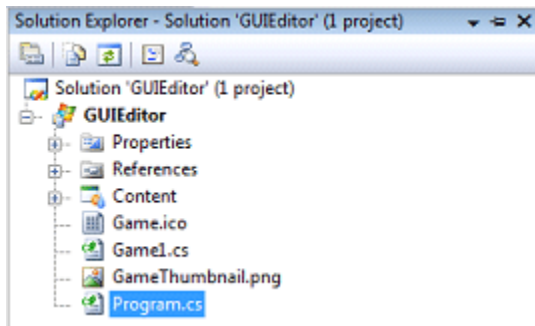
- ❖ "GUI Editor" or "GUI Builder" is a Software Development Tool. It has a kind of WYSIWYG structure and helps the user to build a structure without coding or less coding. Without GUI's we had to write code for everything.

STEP BY STEP BUILDING A "GUI EDITOR"

First of all, we are creating a new XNA 3.1 Game Project...



Here it is..

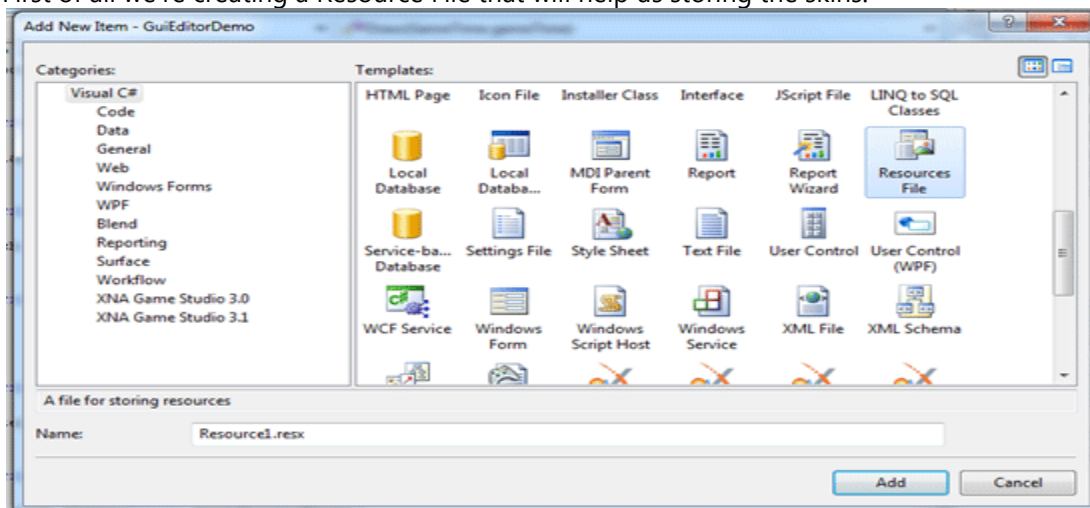


1. Add a resource file that includes Skin images for controls.
2. A Structure that will help us export the control List as an XML File.
3. A "Screens" Folder that will help us to read this controls.
4. For Communication of Windows & XNA creating a class where we will declare "public static" variables
5. A "Properties Panel" where we can change the properties of selected controls.
6. A "Toolbox Panel" where we can add controls.
7. Our Custom Controls

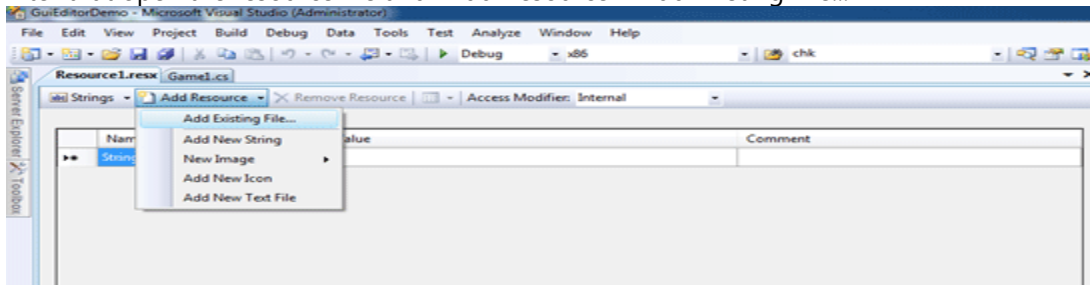
1. Add a resource file that includes Skin images for controls

We are adding 3-4 skin images for our Button control...

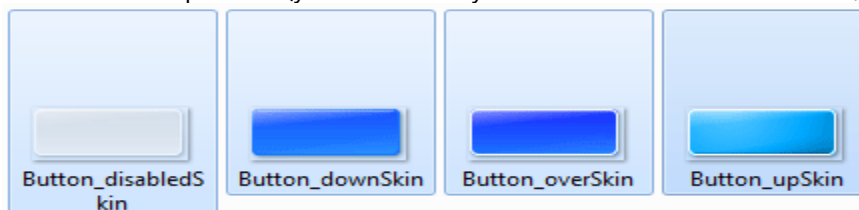
First of all we're creating a Resource File that will help us storing the skins:



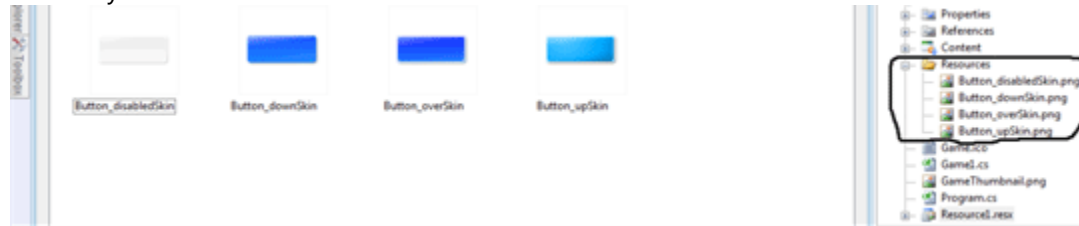
After that open the resource file and "Add Resource->Add Existing File..."



Then add 4 sample skins(you can create your own skins named 'disabled','down','over' and 'up'):

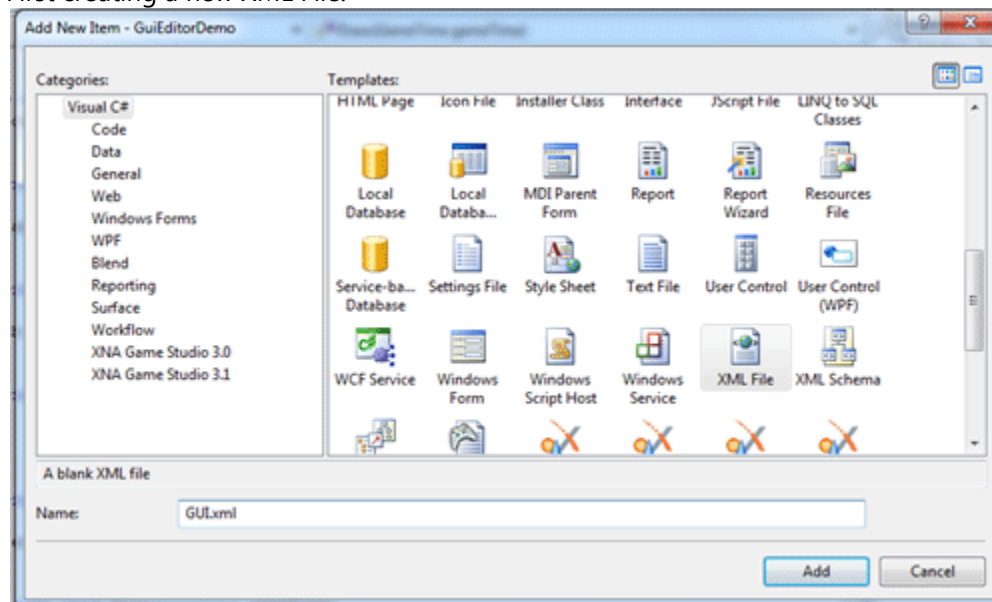


Here they are added at Resources...



2. A Structure that will help us export the control List as an XML File

First creating a new XML File:



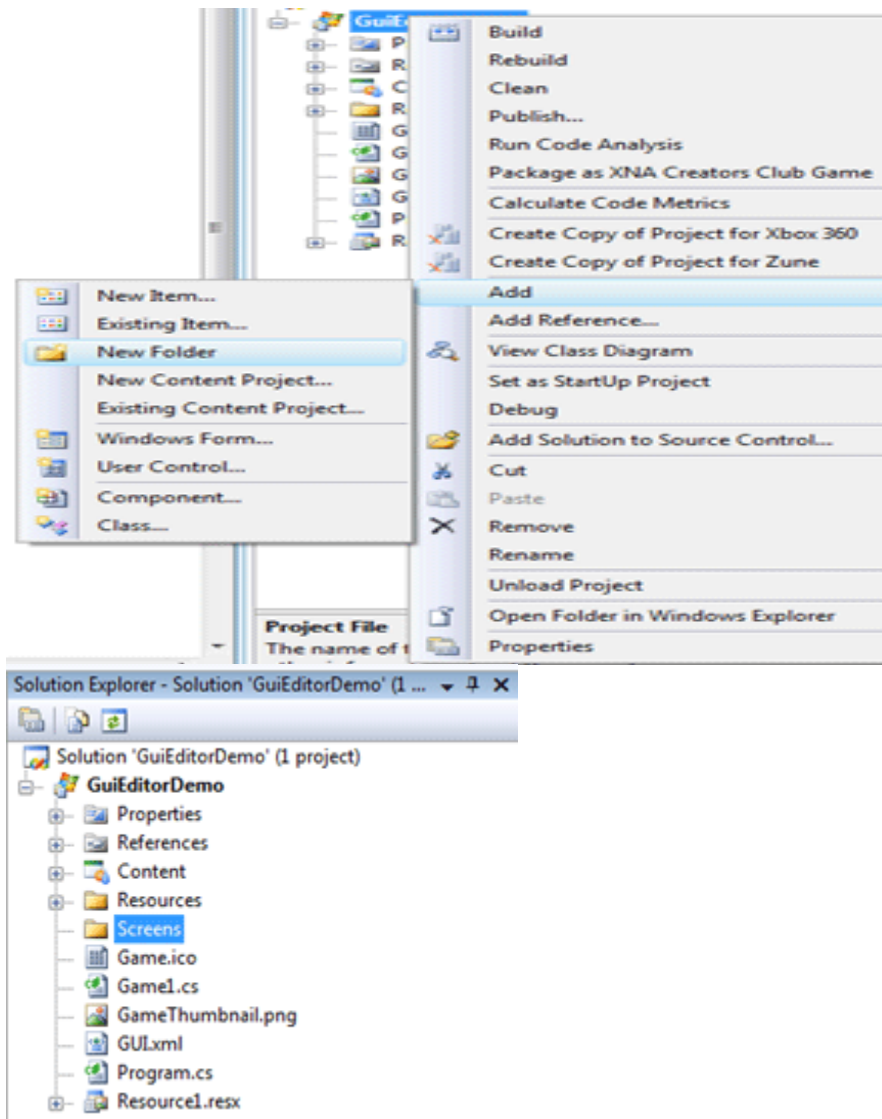
Then make it similar to the codes below:

```
<?xml version="1.0" encoding="utf-8" ?>
<GUI>
  <Control Name="">
    <Type></Type>
    <AllowDrop></AllowDrop>
    <Enabled></Enabled>
    <ForeColor></ForeColor>
    <LocationX></LocationX>
    <LocationY></LocationY>
    <SizeW></SizeW>
    <SizeH></SizeH>
    <Text></Text>
  </Control>
</GUI>
```

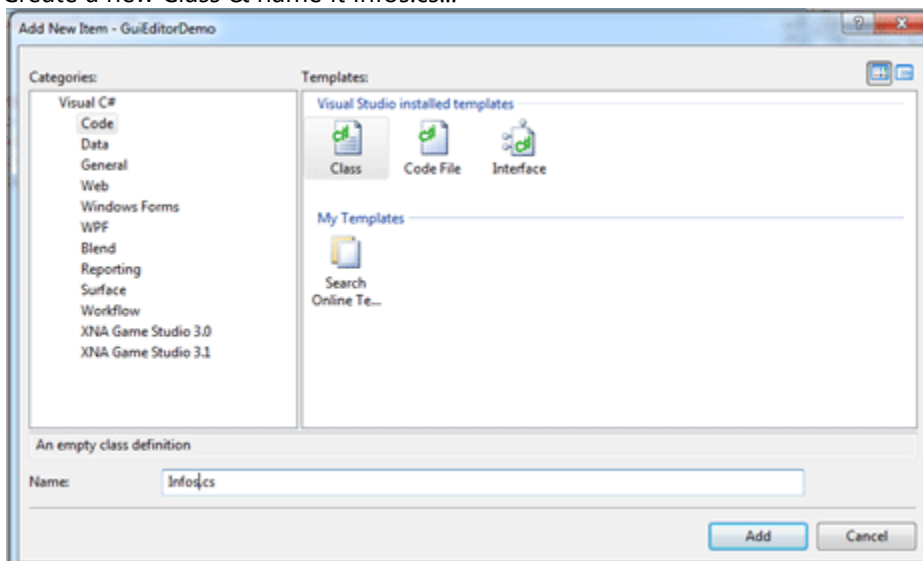
We are taking advantage of XML for the structure we will be building on GUI Editor & as you can see we added elements as they are already properties of the controls. Actually it would be a big mistake not to use XML in this kind of applications.

3. A "Screens" Folder that will help us to read this controls.

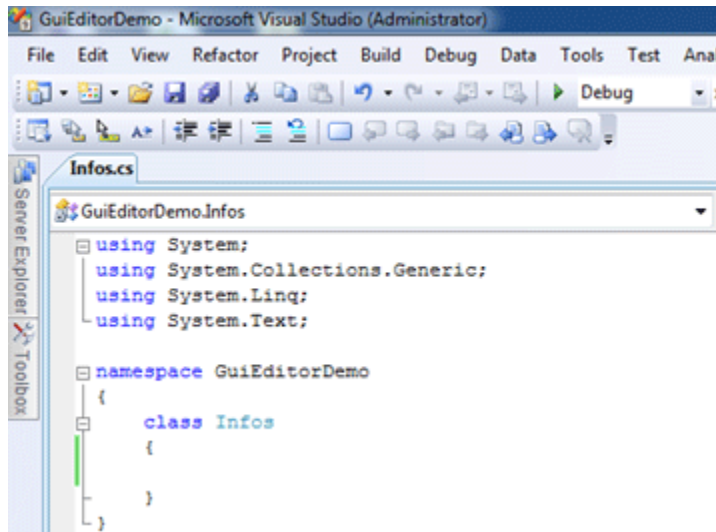
Just create a new Folder in the project and call it Screens.



4. For Communication of Windows & XNA, creating a class where we will declare "public static" variables
 Create a new Class & name it Infos.cs...



It seems just like this:



In this point we aren't writing any code.

5. A "Properties Panel" where we can change the properties of selected controls

We need to add a Properties Panel. First we need to create a Windows Forms:

