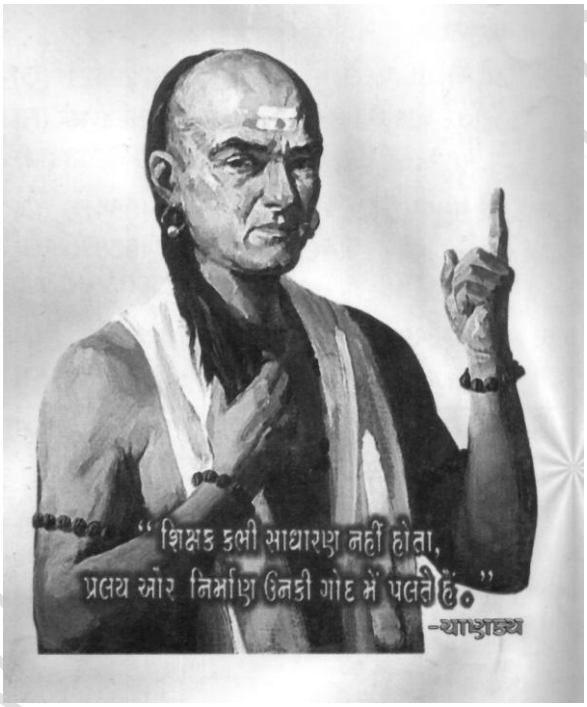
Geetanjali Charitable Trust Sanchalit

GEETANJALI COLLEGE OF MSC(IT) & C.A

INDIAN REDCROSS BUILDING, SUCHAK ROAD,
OPP. SHASTRI MAIDAN, RAJKOT. PH. : 2464447, 2464435

PROGRAMMING WITH C#
BCA SEM - IV



Geetanjali

"Where Pursuit For Progress is Endless"

INDEX

1.	**. NET Framework And Visual Studio IDE, Language Basics**	Introduction to .NET Framework Features / Advantages CLR CTS CLS BCL / FCL / Namespaces Assembly and MetaData JIT and types Managed Code and Unmanaged Code Introduction to .NET Framework and IDE versions Different components (windows) of IDE Types of Projects in IDE (Console, Windows, Web, Setup, etc.) Data Types (Value Type & Reference Type) Boxing and UnBoxing Operators (Arithmetic, Relational, Bitwise, etc.) Arrays (One Dimensional, Rectangular, Jagged) Decisions (If types and switch case) Loops (for, while, do..while, foreach)	
3.	**Class and Inheritance, Property, Indexer, Pointers, Delegates, Event, Collection**	Concept of Class, Object Encapsulation, Inheritance Polymorphism Creating Class and Objects Methods with “ref” and “out” parameters Static and Non-Static Members Constructors Overloading Constructor,	

		Boxing & Unboxing Method and Operator Inheritance Sealed Class & Abstract Class Overriding Methods Overloading Methods, Overloading Operators Interface inheritance Creating and using Property Creating and using Indexer Creating and using Pointers (unsafe concept) Creating and using Delegates (Single / Multicasting) Creating and using Events with Event Delegate Collections ArrayList HashTable Stack Queue SortedList	
5.	**Windows Programming**	Creating windows Application MessageBox class with all types of Show() method Basic Introduction to Form and properties Concept of adding various Events with event parameters Different Windows Controls:- 1) Button 2) Label 3)TextBox 4)RadioButton 5) CheckBox 6)ComboBox 7)ListBox 8)PictureBox 9)ScrollBar 10)TreeView 11)Menu (MenuStrip, ContextMenuStrip) 12)ToopStrip 13)Timer 14)Panel and GroupBox 15)Dialog Boxes (ColorDialog, FontDialog, SaveFileDialog and	

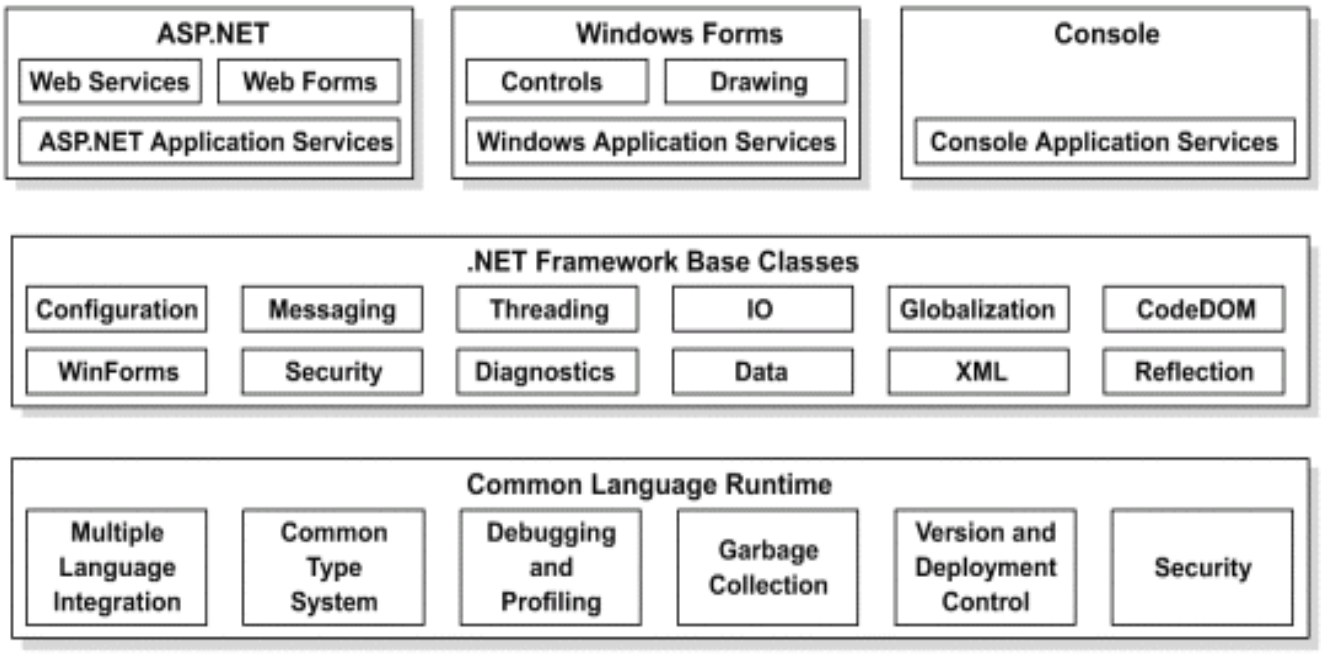
		OpenFileDialog,Print dialogbox) 16)MDI Concept with MDI Notepad Concept of Inheriting Form	
6.	**User Controls (Components)**	Creating User Control with 1) Property 2) Method 3) Event Using User Control in Windows Projects as component	
7.	**Database Programming with ADO.NET**	Concept of Connected and Disconnected Architecture Data Providers in ADO.NET Connection Object Connected Architecture 1)Command 2)DataReader Disconnected Architecture 1)DataAdapter 2)DataSet 3)DataTable 4)DataRow 5)DataColumn 6)DataRelation 7)DataView Data Binding GridView Programming	
8.	**Crystal Reports**	Creating Crystal Reports Types of Reports Report Sections:- Formula, Special Fiend and Summary in Report	
9.	**Setup Project**	Types of Setup Projects Creating Setup Project 1)File System Editor 2)User Interface Editor 3)Launch Conditions Editor	

CH : 1 Introduction

- **Introduction of .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.**



- **What are the features of .NET Platform / .NET Framework?**

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.

- **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.**

- **Multi-Device Development**

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

- **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 processes. **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.

- **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.

- **Easy Deployment**

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

- **Distributed Architecture**

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

- **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.

- **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.

- **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.

- **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.

➢ **What is the common language runtime (CLR)?**

- 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

- **SUMMARY:-** Heart or backbone of .Net
  Execution Engine
  Runtime Engine
  Converting IL to Native code
  Managing memory

- **What is the common type system (CTS)?**
  - 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 form 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

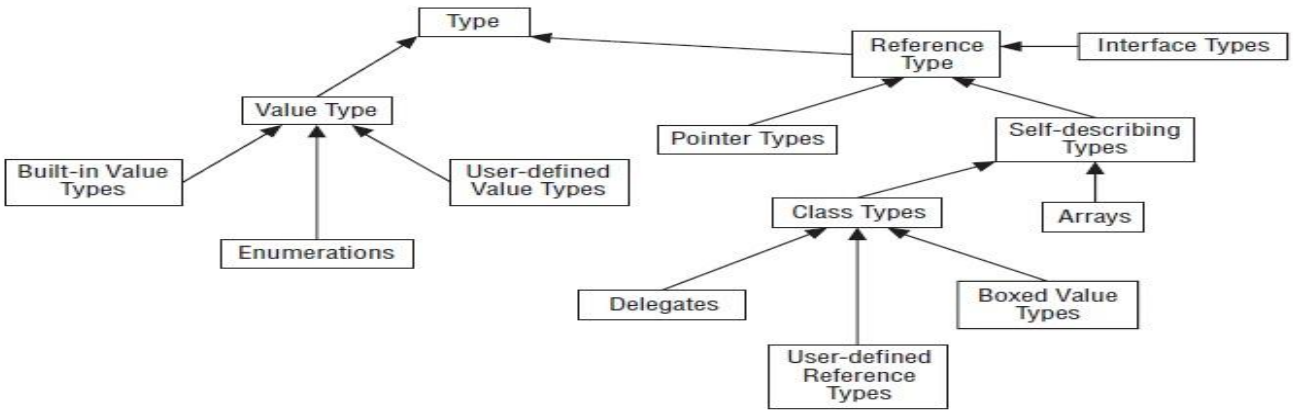
Value types

- Store directly data on stack. In built data type.

  Example: dim a as integer

Reference types

- Store a reference to the value's memory address, and are allocated on the heap.
- Example: Hello a = new Hello();



- **SUMMARY:-**

  CTS **(Common Type System)** = CLS (**Common Language Specification)**

  CTS rules = CLS rules

  Cross language integration

  CTS support 2 types

  1. Value type(Store data in stack)
  2. Reference type(Store reference to the value's memory address)

- **What is the Common Language Specification (CLS)?**
  - 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**.
  - **SUMMARY:-** Include basic language features.
    Subset of CTS (common type system)
- **What is an 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.

- **What are private assemblies and shared assemblies?**
  - 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.**

➢ **SUMMARY:-** Primary building block
Logical DLL.
Collection of functionality.
Private assemblies
Used only by a single application
Stored in application's install directory
Shared assemblies
Referenced by more than one application.

➢ **What is metadata?**

- 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
  - Provide security

➢ **SUMMARY:-** Means data about data.
.NET compiler generate the metadata
Store Metadata in the assembly.
.NET program compile into PE (PORTABLE EXECUTABLE) Files.

Section of PE contains Metadata.

- **What is .NET Framework Class Library? (FCL)**
  - 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**.
- **What is 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.
  - **At the root of the hierarchy is the System namespace.**
  - The notion of a namespace plays a fundamental role in the .NET Framework.
  - For example, imagine that in a Reliance Industry there is an Executive named Ram Vyas, a General Manager named Ram Vyas, and a Tester named Ram Vyas.
  - In this case, the name Ram Vyas is ambiguous. (Salary of GM 1,00,000, Executive 50,000 , Tester 10,000).
  - When the paymaster stands and calls out the names of people to receive their salary, the GM Ram Vyas won't be happy if he rushes to the table when the paymaster calls out his name and the envelope contains the Tester Ram Vyas's salary only 10000.
  - To resolve the naming ambiguity, the Reliance Industry can simply define three namespaces: Executive, General Manager and Tester, Now the three individuals can be unambiguously

referred to by their fully qualified names:

- Executive. Ram Vyas
- GeneralManager. Ram Vyas
- Tester. Ram Vyas

Above three posts Executive, GeneralManger and Tester are like namespace. They are used for identifying Ram Vyas differently.

Same like above Namespaces provide scope: Two classes have the same name but they should reside in different namespace.

- **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`	Types that allow reading and 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`

- **SUMMARY:-**

FCL = collection of namespaces.
Logical naming scheme.
Logical container or partition.
It's like Drives of our computer.
C:, D:, E: and F:.

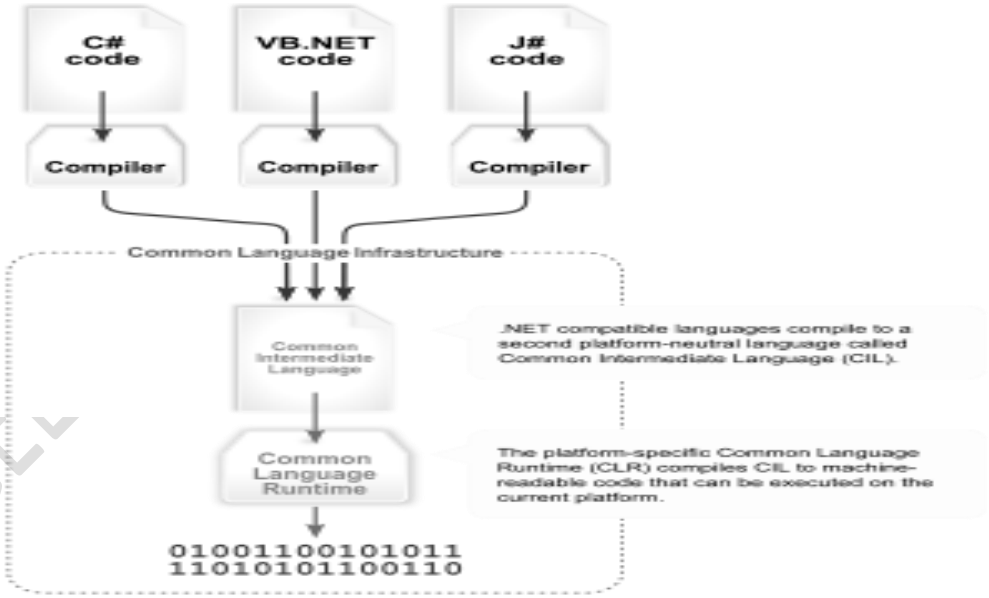
➢ **What is Garbage Collection (GC) ?**

- Garbage Collection is a **mechanism that allows the computer to detect when an object is no longer needed**.
- It then **automatically free up the memory** used by that object.
- One of the advantages of CLR is automatic memory management that uses 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 :

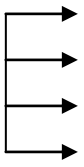
After converting program to MSIL. It is converted to Native Code (Binary Code) so that it can be executed on machine.

Step-4 :

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

- **SUMMARY:-** Automatically free up the memory.
  CLR use garbage collection mechanism.
  Manages the allocation and release of memory.
  Don't need to write code for memory management.

Program Execution of C#.NET

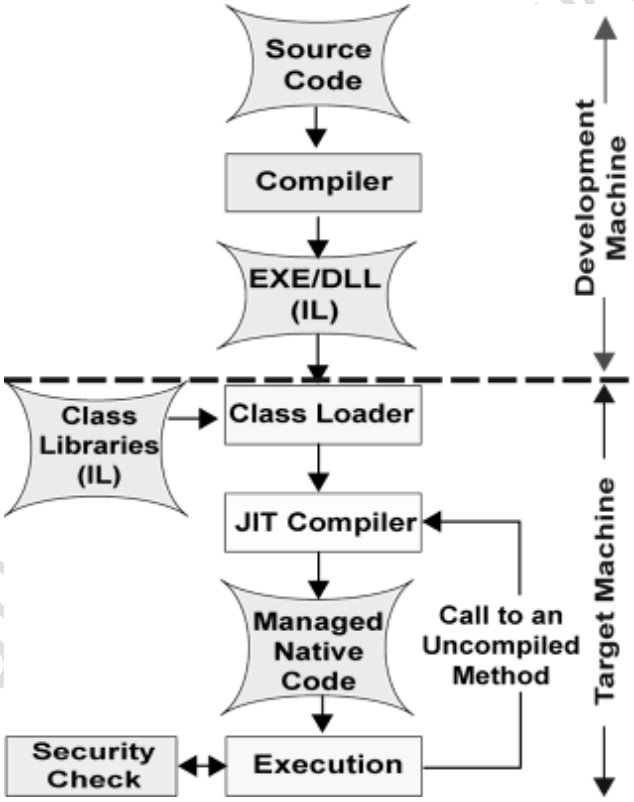


Select Compiler
Compile Code to MSIL
Convert MSIL to Native Code
Execute Code

- **What is the 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**
- **SUMMARY:-** CPU-independent
  Instructions for loading
  Combined with metadata and the common type system
  MSIL allows for true cross-language integration.
  MSIL = CIL (Common Intermediate Language)
  MSIL = IL(Intermediate Language)

- **What is 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 compiled. If changes are made then possible that it will again generates MSIL to Native.



  - Firstly any program compile by its own compiler then it will **convert into MSIL** then with the **help of JIT**; MSIL compile 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.

- **SUMMARY:-** just-in-time.

  Smart compiler.
  Does not compile whole program each time
  Compiles only that portion of the program
  Compile Code to MSIL(USING JIT)
  Convert MSIL to Native Code
  Types of JIT
  Pre-JIT - Compiles complete program into native code
  Econo-JIT - Compiles only those methods that are called at runtime
  Normal-JIT - It's like Econo-JIT.

- **What is Managed Code and 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 coping 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.

➢ **SUMMARY:-**

Managed Code

- C#.NET, J# and vb.net Compilers create.
- Targets the CLR (Common Language Runtime),
- Compiles IL (Intermediate Code)
- Runs in the CLR.
- Always compiled to IL.
- Provides platform independence.
- Can access both managed data and unmanaged data.

UnManaged Code

- Does not target the CLR (Common Language Runtime)
- What you use to make before C#.NET 2003 was released.
- Directly executed by the OS.
- Targets the processor architecture.
- Always dependent on the computer architecture.

- **.NET Framework Versions till now**

Version	Version Number	Release Date	Visual Studio	Default in Windows
1.0	1.0.3705.0	2002-02-13	Visual Studio .NET	
1.1	1.1.4322.573	2003-04-24	Visual Studio .NET 2003	Windows Server 2003
2.0	2.0.50727.42	2005-11-07	Visual Studio 2005	
3.0	3.0.4506.30	2006-11-06		Windows Vista, Windows Server 2008
3.5	3.5.21022.8	2007-11-19	Visual Studio 2008	Windows 7, Windows Server 2008 R2
4.0 Beta 2		2009-10-19	Visual Studio 2010	

- **Features of .Net :**
  - OOP Model
  - Data Security
  - Cross Structured Exception Handling
  - Cross Language Inter Operability
  - Platform Independence
  - Easier Versioning
  - Garbage Collector
  - X-Copy Deployment
  - 40 + Languages Supported

- **.Net Platform :**

Visual Studio

.Net Enterprise Server	.Net Framework	Building Blocks

O.S.

- **.Net IDE.**
  - The Visual Studio .NET IDE contains several features that enable more efficient development of projects.

- **Title Bar:**

  It shows the name of your current application or project and 3-buttons :
  - Minimize Button
  - Restore Button
  - Close Button .

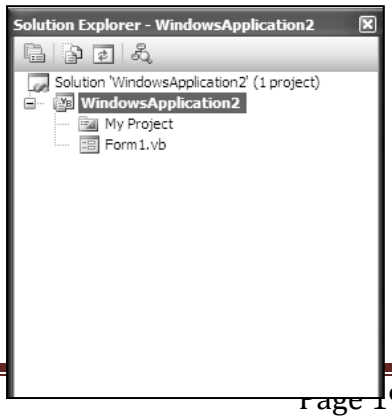
- **MenuBar:**

  It displays the list of Menus available in .Net, Such as
  - File Menu
  - Edit Menu
  - View Menu
  - Tools Menu
  - Test Menu
  - Window Menu
  - Help Menu .

- **ToolBar:**

  It shows the shortcuts for functions available within the Menus, along with related small icons.

- **Solution Explorer:** This is used to manage and browse the files in a Solution.
  - A solution is a set of files and other resourses that are used to build an application.
  - The files in a solution explorer are arranged in

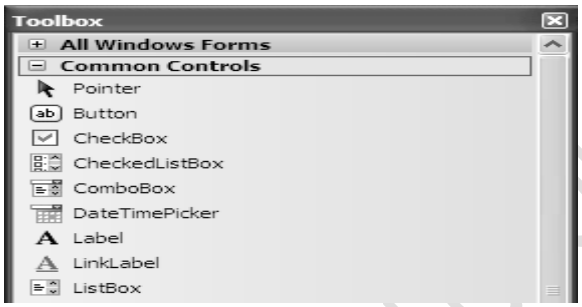


hierarchical order.

- All C# code files have a .cs extension.
- Solution Explorer displays your project hierarchy,including all project references; project items such as forms, classes, modules, and so on; and any subfolders that contain project items.

- **Server Explorer :**It is used to manage database connections.

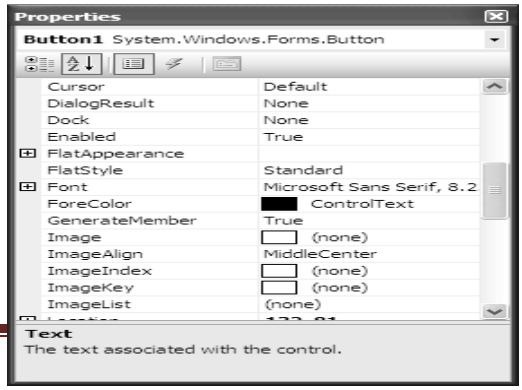
- It is used to browse running windows Services and use them as data source.
  - Server Explorer also has the ability to manage and use specific aspects of a server.
  - It is used for
    - Managing Data Connections
    - Viewing and Managing Servers
    - Using Drag-and-Drop Techniques



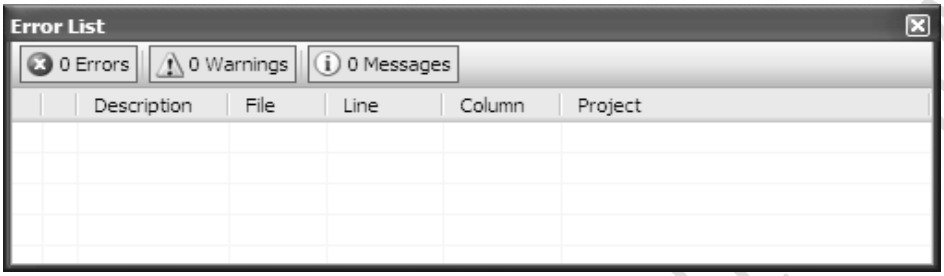
- **Tool Box :**
  - It shows all the user interface controls available in .net .
  - The Toolbox contains tools appropriate for the item that you currently have selected and for the project type that you are working on.
  - The Toolbox contains tools appropriate for a Form Designer.
  - These include Windows Forms controls and components, plus tools in the other Toolbox tabs: Crystal Reports, Data, and Components (plus the General tab is scrolled off the bottom of the Toolbox). You can add other customized tabs to the Toolbox to hold your favorite controls and components.

- **Properties Window:**

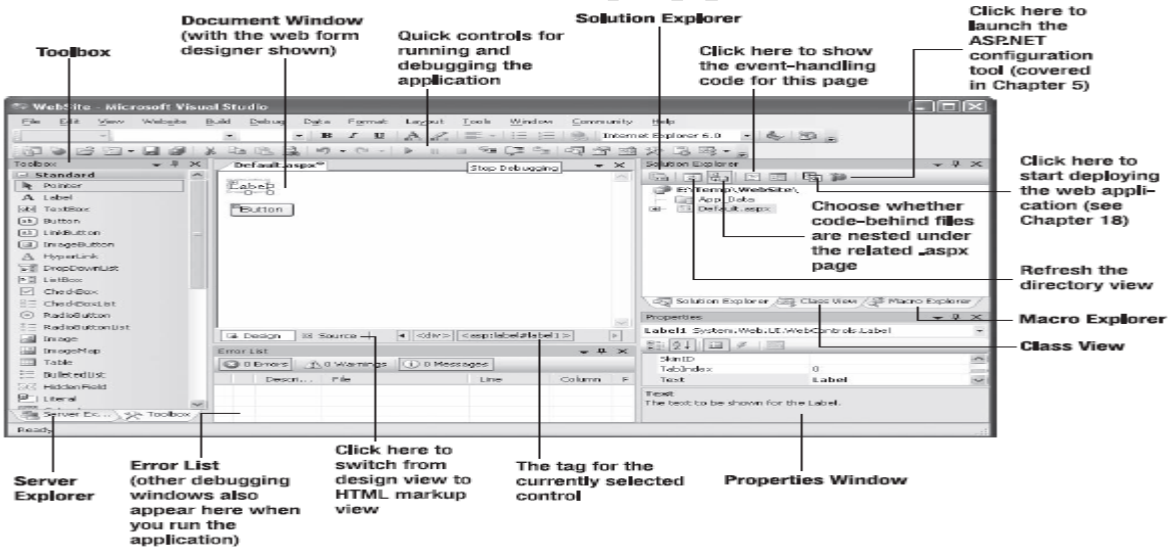
  It is used to edit properties in a GUI.
  - It lists all the available properties for all objects including class, forms, webpages and other items.



- Displays information about selected items , such as user interface components.
  - The Properties window lets you change an object's properties at design time. When you select an object in a Form Designer or in the Solution Explorer, the Properties window displays that object's properties. To change a property's value, simply click the property and enter the new value.
- **Error List Window :**shows the list of errors , if something missing or wrong in your application.



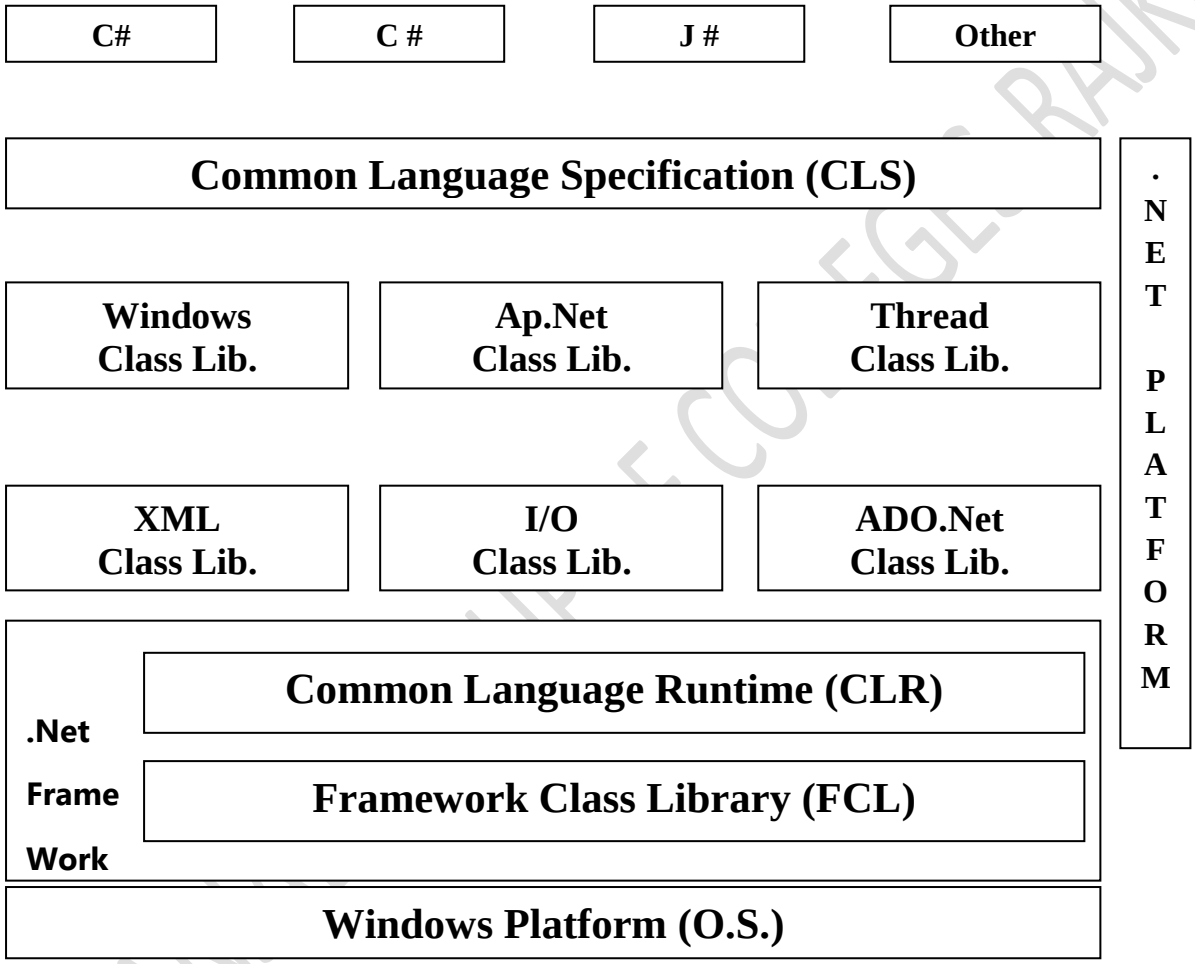
- **Code Editor Window :**used to write the code in your application.
  - It supports syntax highlighting and code completion for variables , functions, methods . loops , etc.



- **SUMMARY:-** The Visual Studio .NET IDE contains several features that enable more efficient development of projects.
  - **Title Bar:** Minimize Button, Restore Button, Close Button .
  - **MenuBar:** File Menu, Edit Menu, View Menu, Tools Menu, Test Menu, WindowMenu, Help Menu .
  - **ToolBar:**

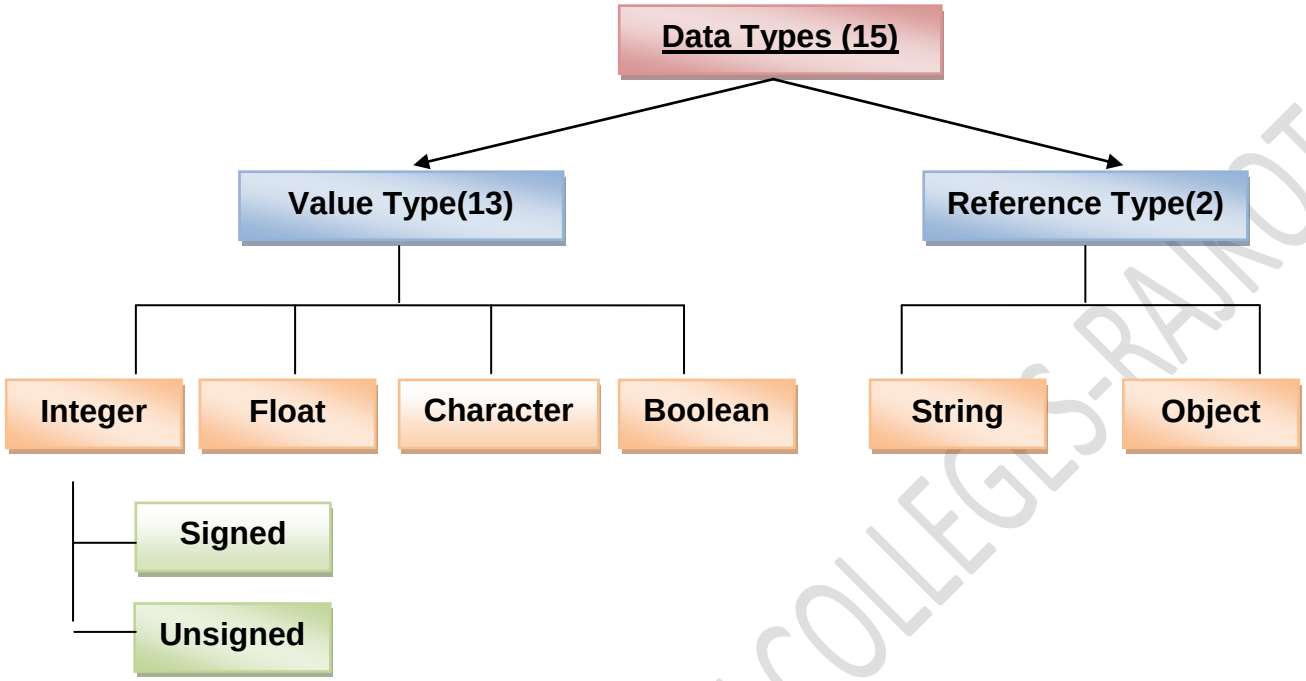
- **Solution Explorer:** This is used to manage and browse the files in a Solution.
- **Server Explorer :-** It is used to manage database connections.
- **Tool Box :** It shows all the user interface controls
- **Properties Window:-** It is used to edit properties in a GUI.
- **Error List Window :-** Shows the list of errors , if something missing or wrong in your application.
- **Code Editor Window :-** Used to write the code in your application.

➢ **Architecture of .Net :**



CH : 2 C # Basics

- **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	-	-

- **SUMMARY :-** DATA TYPES
  - INTEGER
    - SIGNED INTEGER
    - UNSIGNED INTEGER
  - FLOAT
  - CHARACTER
  - BOOLEAN
  - STRING
  - OBJECT

SIGNED INTEGER: - SBYTE, SHORT, INT, LONG (S2IL)

UNSIGNED INTEGER: - BYTE, USHORT, UINT, ULONG (BU3)
FLOATING POINT: - FLOAT, DOUBLE, DECIMAL (FD2)
CHARACTER: - CHAR
BOOLEAN: - BOOL
STRING: - STRING
OBJECT: - OBJECT

- **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)

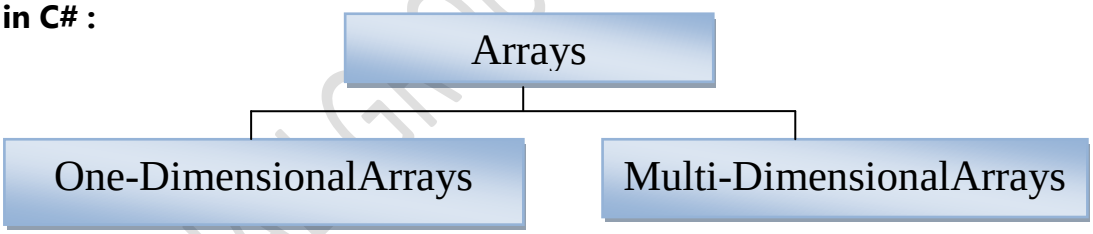
➢ **SUMMARY:-**

ARITHMETIC: - DEVISION, MULTIPLICATION, ADDITION, SUBTRACTION, MODULES, INCRIMENT, DECRIMENT (DEMASMINDE)
(/, *, +, -, %, ++, --)

RELATIONAL & LOGICAL: - EQUALTO, NOT EQUALTO, LESSTHAN, GREATERTHAN, GREATERTHAN OR EQUALTO, LESSTHAN OR EQUALTO (==, !=, <, >, >=, <=)

BITWISE: - AND, OR, NOT, XOR, S-LEFT, S-RIGHT (&, |, ~, ^, <<, >>)

➢ **Array in C# :**



- 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] ;
- Example : conceptually, the sample one-dimensional array looks like this :

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

2

- To
- initialize array element :

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] ;
```

  - This example will make a table of varing row size as shown below :

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

  - **SUMMARY:-** Collection of variables of the same type
    Array can have one or more dimensions.
    Grouping together related variables.
    It organizes data in such a way that it can be easily manipulated.

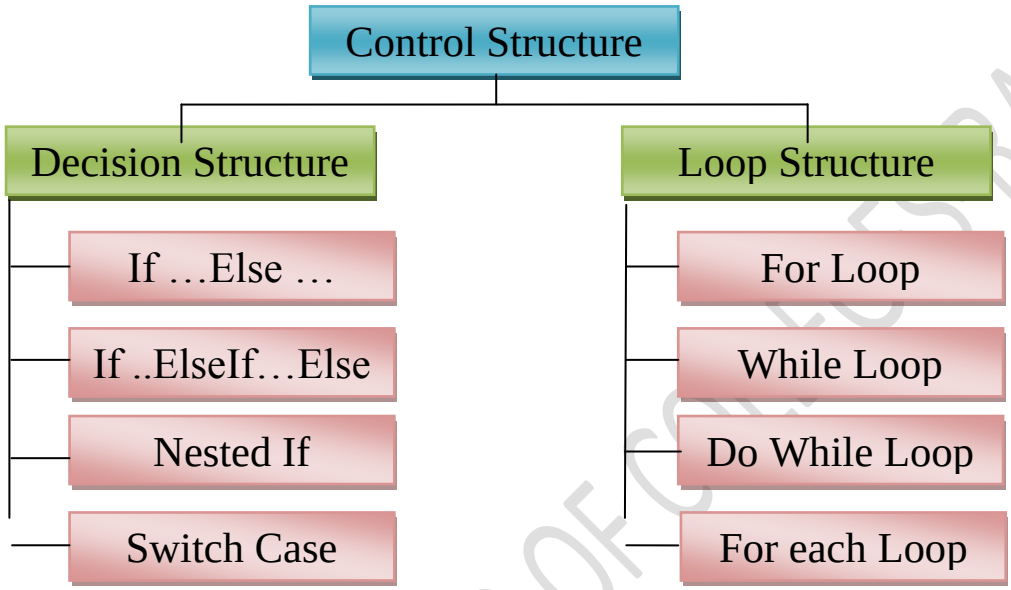
One-dimensional Array : List of related variables.

Multi-dimensional Array: More than one row and column

- TYPES:
  - Rectangular Array
  - Jagged Array

Jagged Array : "Array of array" in which the length of each array can differ.

➢ **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 if-else-if Ladder :**
  - This type of conditional structure is **used to check multiple conditions.**
  - 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;
}
else
{
   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)
        {
                statement;
        }
        else
        {statement;    }
} else
        {statement;}
```

➢ **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 ;
  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**.

LOOPS:-

- **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) ;
```

➢ **The for each loop :**

- for each loop is used to **cycle through the elements of a collection.**
- 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 "type" and "loopvariable" specifies the datatype and the name of loop variable.
- 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 are no more elements to obtain.
- Thus, "foreach" loop cycles through the collection one element at a time, from start to finish.

➢ **SUMMARY:-**

➢ **The if statement :**

- Checks the condition
- True – TARGET STATEMENT WILL EXECUTE.

- False – ELSE PART WILL EXECUTE.
- Either "if" or "else" will be executed.
- "else" clause is optional.

➢ **The if-else-if Ladder :**

- Used to check multiple conditions.
- Evaluated from the top to downward.
- If none of the conditions is true, then the final "else" clause will be executed.
- The final "else" is like a default condition.

➢ **Nested ifs :**

- Target of another if or else.
- "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 switch statement :**

- Provides for a multiway branch.
- Select among several alternatives.
- Must be of an integer, char, byte, or string type.
- The "default case" is optional.
- Executed until the break is encountered.

➢ **The for loop :**

- Used to repeatedly execute a sequence of code.
- "Initialization" = sets a loop control variable.
- The "condition" = Boolean expression
  - If test is "true", the for loop continues to iterate.
  - If it is "false", the loop terminates.
- The "increment" = how the loop control variable is changed each time the loop iterates.

➢ **The while loop :**

- Used to repeatedly execute statements.
- Single statement or a block of statements.
- Condition that controls the loop and may be valid Boolean Expression.
- The statement is executed while the condition is "true".
- Condition = "false" , program control passes to the line immediately following the loop.

➢ **The do-while loop :**

- Checks its condition at the bottom of the loop.
- Will always execute at least once.
- Executes as long as the conditional expression is true.
- The general form or syntax is as follows :

- **The for each loop :**
  - Used to cycle through the elements of a collection.
  - A collection is a group of objects.
  - The loop ends when there are no more elements to obtain.

CH : 3 Class & Inheritance

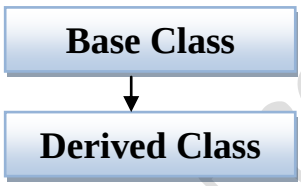
- **Class :**
  - Classes are the **basic ingredients of Object-Oriented languages**.
  - It is **used to define custom data and methods**.
  - Classes can be **declared by using the "class" keyword** followed by the name of class and the brackets surrounding the body of the class.
  - **Supports member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

➢ **Inheritance in C# :**

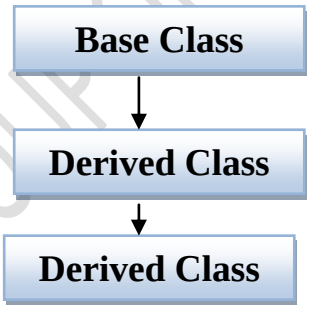
- Inheritance is the **process by which one object can acquire the properties of another object**.
- Inheritance is one of the principle of object-oriented programming supports the concept of hierarchical(top-down) classification.
- Using inheritance the **object can inherits its general attributes from 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 :**

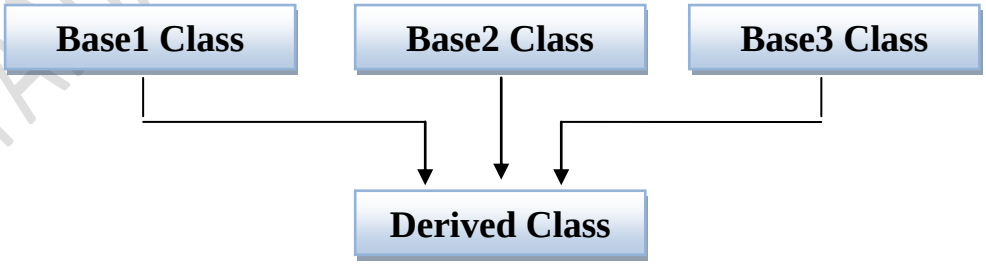
(1) Single Inheritance :



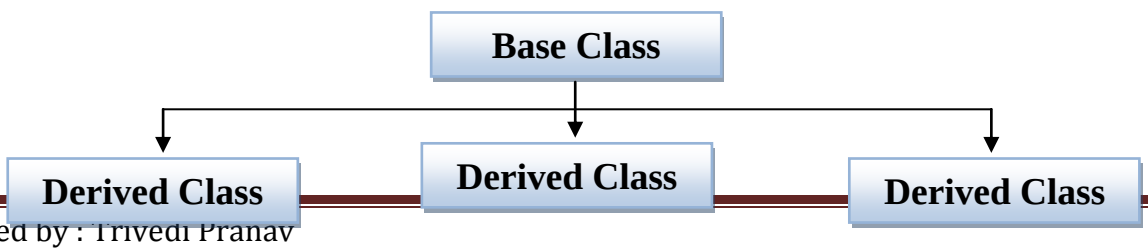
(2) Multi-Level Inheritance :



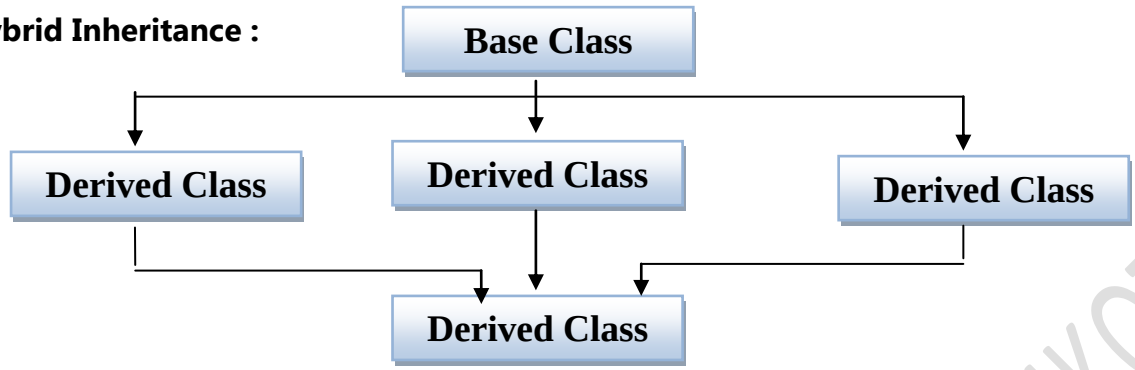
(3) Multiple Inheritance :



(4) Hierarchical Inheritance :



(5) Hybrid Inheritance :



- Multiple inheritances are **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;
}
```

  - **SUMMARY:-** One object can acquire the properties of another object.
    One of the principle of object-oriented programming
    Object can inherits its general attributes from its parent.
    Reusing something that is already exists.

Types of Inheritance :

Single Inheritance :
Multi-Level Inheritance :
Multiple Inheritance :
Hierarchical Inheritance :
Hybrid Inheritance :

Multiple inheritances is not supported directly by c#.

- **Encapsulation**
  - Encapsulation is defined 'as the process of enclosing one or more items within a physical or logical package'.
  - Encapsulation, in object oriented programming methodology, prevents access to implementation details.
  - Abstraction and encapsulation are related features in object oriented programming. **Abstraction allows making relevant information visible and encapsulation enables a programmer to *implement the desired level of abstraction*.**
  - Encapsulation is implemented by using **access specifiers**. An **access specifier** defines the scope and visibility of a class member. C# supports the following access specifiers:
    - Public
    - Private
    - Protected
    - Internal
    - Protected internal
- **Polymorphism**
  - The word polymorphism means having many forms. In object-oriented programming paradigm, polymorphism is often expressed as 'one interface, multiple functions'.
  - Polymorphism can be static or dynamic. In static polymorphism **the response to a function is determined at the compile time**. In dynamic polymorphism , **it is decided at run-time.**
- **Static Polymorphism**
  - The mechanism of linking a function with an object during compile time is called early binding. It is also called static binding. C# provides two techniques to implement static polymorphism. These are:
    - Function overloading
    - Operator overloading
- **Function Overloading**
  - You can have multiple definitions for the same function name in the same scope. The definition of the function must differ from each other by the types and/or the number of

arguments in the argument list. You cannot overload function declarations that differ only by return type.

- Following is the example where same function **print()** is being used to print different data types:

```
using System;
namespace PolymorphismApplication
{
  class Printdata
  {
    void print(int i)
    {
      Console.WriteLine("Printing int: {0}", i );
    }
    void print(double f)
    {
      Console.WriteLine("Printing float: {0}" , f);
    }
    void print(string s)
    {
      Console.WriteLine("Printing string: {0}", s);
    }
    static void Main(string[] args)
    {
      Printdata p = new Printdata();
      // Call print to print integer
      p.print(5);
      // Call print to print float
      p.print(500.263);
      // Call print to print string
      p.print("Hello C++");
      Console.ReadKey();
    }
  }
}
```

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

```
Printing int: 5
Printing float: 500.263
Printing string: Hello C++
```

- **Dynamic Polymorphism**

- C# allows you to create abstract classes that are used to provide partial class implementation of an interface. Implementation is completed when a derived class inherits from it. Abstract classes **contain abstract methods, which are implemented by the derived class**. The derived classes have more specialized functionality.
- Please note the following rules about abstract classes:
  - You cannot create an instance of an abstract class
  - You cannot declare an abstract method outside an abstract class
  - When a class is declared **sealed**, it cannot be inherited, abstract classes cannot be declared sealed.
- The following program demonstrates an abstract class:

```
using System;
namespace PolymorphismApplication
{
  abstract class Shape
  {
    public abstract int area();
  }
  class Rectangle:  Shape
  {
    private int length;
    private int width;
    public Rectangle( int a=0, int b=0)
    {
      length = a;
      width = b;
    }
    public override int area ()
    {
      Console.WriteLine("Rectangle class area :");
      return (width * length);
    }
  }
  class RectangleTester
  {
    static void Main(string[] args)
    {
      Rectangle r = new Rectangle(10, 7);
      double a = r.area();
      Console.WriteLine("Area: {0}",a);
```

```
            Console.ReadKey();
        }}
    }
```

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

```
Rectangle class area :
Area: 70
```

- When you have a function defined in a class that you want to be implemented in an inherited class(es), you use **virtual** functions. The virtual functions could be implemented differently in different inherited class and the call to these functions will be decided at runtime.
- Dynamic polymorphism is implemented by **abstract classes** and **virtual functions**.
- The following program demonstrates this:

```
using System;
namespace PolymorphismApplication
{
  class Shape
  {
    protected int width, height;
    public Shape( int a=0, int b=0)
    {
      width = a;
      height = b;
    }
    public virtual int area()
    {
      Console.WriteLine("Parent class area :");
      return 0;
    }
  }
  class Rectangle: Shape
  {
    public Rectangle( int a=0, int b=0): base(a, b)
    {
    }
    public override int area ()
    {
      Console.WriteLine("Rectangle class area :");
      return (width * height);
    }
  }
```

```
class Triangle: Shape
{
   public Triangle(int a = 0, int b = 0): base(a, b)
   {
   }
   public override int area()
   {
      Console.WriteLine("Triangle class area :");
      return (width * height / 2);
   }
}
class Caller
{
   public void CallArea(Shape sh)
   {
      int a;
      a = sh.area();
      Console.WriteLine("Area: {0}", a);
   }
}
class Tester
{
                static void Main(string[] args)
                {
                        Caller c = new Caller();
                    Rectangle r = new Rectangle(10, 7);
                    Triangle t = new Triangle(10, 5);
                    c.CallArea(r);
                    c.CallArea(t);
                    Console.ReadKey();
            }
        }
}
```

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

Rectangle class area:

Area: 70

Triangle class area:

Area: 25

- **Creating Class and Objects Methods with "ref" and "out" parameters**
  - **ref**

- The **ref** keyword can be **used to pass arguments by reference**. That means the parameter can be changed in the method and the changed value will be retained when control passes back to the calling method. Ref parameters need to be initialized before passing.
- The ref keyword causes **arguments to be passed by reference.**
- The effect is that any changes made to the parameter in the method will be reflected in that variable when control passes back to the calling method.

Example

```
public class RefMethodParameterDemo
{
        public static void RefParamMethod1(ref int num1, ref int num2)
        {
                num1 = 20;
               num2 = 40;
        }
        static void Main()
        {
        //Need to be initialized before passing as argument
                int n1 = 0;
                int n2 = 0;
                //Both the method definition and the calling method
                //must explicitly use the ref keyword.
                RefParamMethod1(ref n1, ref n2);
                Console.WriteLine("n1 = " + n1);
                Console.WriteLine("n2 = " + n2);
                Console.Read();
    }
  }
```

o/p

```
n1 = 20
n2 = 40
```

- **out**

- The **out** keyword can be **used to pass arguments by reference.**
- This is similar to the ref keyword, except that out doesn't require the variable be initialized before being passed.
- A method **without parameter** is useful **when you want a method to return multiple values.**

- The method definition must explicitly use the out keyword
- The method is required to assign a value before the method returns.
- EXAMPLE:-

```
class OutMethodParameterDemo
{
        public static void OutParamMethod1(out int num1, out string str1)
      {
                num1 = 20;
                str1 = "Hi I am out";
      }
      static void Main()
      {
                //No need to initialize the variable before passing
                int n1;
               string s1;
/*Both the method definition and the calling method must explicitly use the out keyword The out
keyword causes arguments to be passed by reference.*/
                OutMethodParameterDemo.OutParamMethod1(out n1, out s1);
                Console.WriteLine(n1);
                Console.WriteLine(s1);
                Console.Read();
      }
}
```

o/p

20
Hi I am out

- **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.

- **SUMMARY:-**

Used when you want to prevent the concept of “Inheritance”.
Declaration with “sealed” keyword.
Sealed class is totally opposite of “Abstract” class.
Illegal to declare a class as both “sealed” and “abstract”.
Used on virtual methods to prevent further overrides.

- **Method overriding**
  - Method overriding in C# is a feature **like the virtual function in C++.**
  - Method overriding is a feature that allows you to invoke functions (that have the same signatures) that belong to different classes in the same hierarchy of inheritance using the base class reference.
  - C# makes use of two keywords: **virtual and overrides to accomplish Method overriding**. Let's understand this through small examples.

P1.cs

```
class BC
{
        public void Display()
        {
                System.Console.WriteLine("BC::Display");
        }
}
class DC : BC
{
        new public void Display()
        {
                System.Console.WriteLine("DC::Display");
        }
}
class Demo
{
        public static void Main()
        {
                BC b;
                b = new BC();
                b.Display();
        }
}
```

Output

```
BC::Display
```

  - The above program compiles and runs successfully to give the desired output.
  - It consists of a base class BC and a derived class DC.
  - Class BC consists of function Display(). Class DC hides the function Display() it inherited from the base class BC by providing its on implementatin of Display(). Class Demo consists of entrypoint function Main().

- Inside Main() we first create a reference b of type BC. Then we create an object of type BC and assign its reference to reference variable b.
- Using the reference variable b we invoke the function Display(). As expected, Display() of class BC is executed because the reference variable b refers to the object of class BC.

➢ **CONSTRUCTOR:**

o A constructor is a **method in the class which gets executed when its object is created**.

o Usually, we put the initialization code in the constructor.

o Writing a constructor in the class is damn simple, have a look at the following sample:

```
public class mySampleClass
{
        public mySampleClass()
        {
                // This is the constructor method.
        }
        // rest of the class members goes here.
}
```

o When the object of this class is instantiated, this constructor will be executed. Something like this:

```
mySampleClass obj = new mySampleClass()
// At this time the code in the constructor will be executed
```

➢ **Constructor Overloading**

o C# supports overloading of constructors, that means, we can have **constructors with different sets of parameters.** So, our class can be like this:

```
public class mySampleClass
{
        public mySampleClass()
        {
                // This is the no parameter constructor method.
                // First Constructor
        }
        public mySampleClass(int Age)
        {
                // This is the constructor with one parameter.
                // Second Constructor
        }
        public mySampleClass(int Age, string Name)
        {
```

```
        // This is the constructor with two parameters.
        // Third Constructor
    }
    // rest of the class members goes here.
}
```

- Well, note here that call to the constructor now depends on the way you instantiate the object. For example:

```
mySampleClass obj = new mySampleClass()
// At this time the code of no parameter
// constructor (First Constructor)will be executed
mySampleClass obj = new mySampleClass(12)
// At this time the code of one parameter
// constructor(Second Constructor)will be
// executed.
```

- The call to the constructors is completely governed by the rules of overloading here.

- **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
 {
        Static void Main()
        {
                int X ;
                object Obj ;
                X = 10 ;
                Obj = x ; // box int x  into an object
                int Y = (int) obj ; //unbox obj to int by casting
        }
 }
```

SUMMARY:-

- **Boxing :**
  - Object refers to a value type.
  - Casting of value type to object type is called "boxing".
  - No explicit type casting is required.
  - Value of value type to be stored in object type.
  - Value type is boxed inside the object.
- **UnBoxing :**
  - Reverse process of "boxing".
  - Casting of object type to value type.
  - Performed using explicit cast from object type to value type.
  - Necessary to unbox object type to only that value type from which it was boxed.

➢ **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
        }
}
```

- **SUMMARY:-**

  Concept of polymorphism.
  Differ in their type and number of parameters.
  Method is called whose return type and parameter list is matched.

- **Overloading Operators :**
  - Operator overloading is **closely related to method overloading**.
  - Operator overloading is a **concept of polymorphism**, used to **expand the usage of operator**.
  - To overload an operator the **"operator" keyword is used** to define an "operator method" which defines the 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 :

    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
  }
```

- 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 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".

➢ **SUMMARY:**

Closely related to method overloading.
Concept of polymorphism.
Used to expand the usage of operator.
"Operator" keyword is used.
None of its original meaning is lost.
There are two forms of operator methods :
For "unary" operators.
For "binary" operators.

CH: 4 Properties, Indexers, Pointers, Delegates, Events & Collections

- **Properties 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 assessors are **used to get and set the value of variable**.
  - Assessors 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
{
        get
        {
                // get accessor code
        }
        set
        {
                // set accessor code
        }
}
```

- **SUMMARY:-**
- **Possible**

  One type of class member.

  Along with get and set accessors.

  Assignments like a normal variable

  Get and set are automatically invoked.

  - **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 returen 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
        {
            // get accessor code
        }
        set
        {
            // set accessor code
        }
}
```

- **SUMMARY:-**
  - 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.

- **Interface :**
  - Interface is syntactically **similar to an "abstract" classes** and **methods in which no method can include a body**.
  - Interface specifies what must be done but not how.
  - An interface might not seem as all-powerful as a class, but interface can be used where a class can't.
  - A class can **inherit from only one other class and can implement multiple interfaces.**
  - Interface can be declared as :

```
interface name
{
    ret-type method-name1(parameter-list);
    ret-type method-name2(parameter-list);
}
```

  - Once interface is defined, **any number of classes can implement it**.
  - To implement an interface, a class must provide bodies for the method described by the interface.
  - Each class is **free to determine the details of its own implementation.**
  - Thus, two classes might implement the same interface in different ways.
  - So, **interface is the concept of "polymorphism".**
  - The general form of a class that implements an interface :

```
class class-name : interface-name
  {
   // class-body
  }
```

- **Interface inheritance :**
  - **One interface can inherit another.**
  - Syntax is the same as for inheriting classes.
  - C# **does not support multiple inheritances** of classes as c++ and other object-oriented languages do.
  - The capacity of multiple inheritances was intentionally left out because of the trouble caused by its complex implementation.

- C# **provides the functionality and benefit of multiple inheritance by enabling multiple interfaces to be implemented.**
- When a class implements an interface that inherits another interface, it must provide implementations for all the members defined within the interface inheritance chain.

```
public interface IA
{
        void Meth1( );      void Meth2( );
}
public interface IB : IA
{
        void Meth3( );
}
class MyClass : IB
{
        // Class Body
}
```

- When interface inherits another, **it is possible to declare a member in the derived interface that hides one defined by the base interface.**
- This happens when a member in a derived interface has the same declaration as one in the base interface.
- In this case, the base interface name is hidden.

➢ **SUMMARY:-**

- Similar to an "abstract" classes
- Methods in which no method can include a body.
- Interface can be used where a class can't.
- Any number of classes can implement in interface.
- Interface is the concept of "polymorphism".

➢ **Interface inheritance :**

- One interface can inherit another.
- Syntax is the same as for inheriting classes.
- When interface inherits another, it is possible to declare a member in the derived interface that hides one defined by the base interface.

- **Pointer :**
  - Pointer is a **variable that contains the address of other variable**.
  - Pointers are **like a reference** in c#.
  - The main difference between "pointer" and "reference" is that **a pointer can point anywhere in memory**, where a **reference always refers to an object of its type**.
  - In c# **managed code prevents the use of pointer.**
  - So, all pointer operations **must be marked as "unsafe".**
  - To compile this unmanaged code the "unsafe compiler option" must be used.
- **Pointer Declaration :**
  - Pointers can be declared like other variable, additionally use a **"*"** operator.
  - Syntax to declare a pointer variable :

```
Syntax :
        type*  ptr-name ;

Example :
        int* ip;
        float* fp;
```

  - When a pointer points to a variable the value of that variable can be obtained or changed through the pointer.
  - In above example, "ip" can be used to point to an integer datatype and "fp" can be used to point to float datatype.
- **Pointer Initialization :**
  - Pointer can be **initialed to an address of other variable of the type** which is declared at the time of pointer declaration.
  - Pointers can be initialize like other variable, additionally use a **"&"** operator.
  - Syntax to initialize pointer variable :

```
Syntax :
        ptr-name = &variable-name ;
 Example :
        int num = 10 ;
        int* ip;
        ip = &num;
```

  - The "&" operator is **used to return the memory address** of operand.

- The "*" operator is **used to declare pointer variable** and refers to the value of variable pointed by the pointer.

➢ **Pointer to Array :**

- Pointer to Array is a variable that contain the address of an array.

```
Syntax :
        ptr-name = &array-name[0] ;
 Example :
        int[ ] num = new int[10] ;
        fixed(int* ip=&num[0]);
```

- The name of an array without any index generates a pointer to the start of the array.

```
Syntax :
        ptr-name = array-name ;
 Example :
        int[ ] num = new int[10] ;
        fixed(int* ip=num);
```

- The "fixed" modifier is often used when working with pointers.
- It prevents a managed variable from being moved by the garbage collector.

➢ **Pointer to Structure :**

- Pointer to structure **can be used to point an object of structure datatype as long as the structure does not contains any reference type.**
- When you access a memory of structure through a pointer you must use the **arrow operator "→" instead of dot operator " . "**
- The syntax and example is shown below.

```
   Syntax :
        struct-type* ptr-name ; //declaration
        ptr-name = &obj-of-structure ; //initialization
Example :
struct Mystruct
{
      public  int a;
      public void display()
      {
         Console.WriteLine(a);
      }}
class Program
  {
```

```
unsafe static void Main(string[] args)
{
    Mystruct obj = new Mystruct();
    Mystruct* ptr;
    ptr = &obj;
    ptr->a = 10;
    ptr->display();
    Console.ReadKey();
} }
```

- Using "**unsafe**" : C# allows you to write "unsafe" code.
- It is the **code does not execute under the full management of the Common Language Runtime (CLR).**
- This is used because managed code prevents the use of pointers.
- Because the pointer can point anywhere in memory, it is possible to misuse a pointer.
- This is why c# does not support pointers when creating managed code.
- So, to compile unmanaged code, you must use the unsafe compiler option.

- **SUMMARY:-**
  - Contains the address of other variable.
  - Like a reference in c#.
  - A pointer can point anywhere in memory
  - Reference always refers to an object of its type.
  - Managed code prevents the use of pointer.
  - **Pointer Declaration :**
    - Pointers can be declared like other variable, additionally use a **"*"** operator.
  - **Pointer Initialization :**
    - Initialed to an address of other variable of the type
    - Use a "&" operator.
    - The "&" operator is used to return the memory address of operand.
    - The "*" operator is used to declare pointer variable.

- **Delegate :**
  - Delegate is an **object that can refer to a method.**
  - Delegate **holds the address of method so method can be called** through a delegate.

- Delegate in c# is **similar to a "function pointer" in C and C++**, the difference between them is that **delegate is "type safe".**
- Technically, a delegate is a reference type used to encapsulate a method with a specific signature and return type.

➢ There are four steps to use delegate :

➢ **Step 1 : Declaration of Delegate :**

- A delegate is created with the keyword "delegate", followed by a return type and the signature of the methods that can be delegated to it.
- The general form of the delegate declaration is shown below :
  - **delegate ret-type name(parameter-List) ;**
- Here, "ret-type" is the type of value returned by the methods that the delegate will be calling.
- The name of delegate is specified by the "name".
- The parameters required by the methods called through the delegate are specified in the "parameter-list".
- Once delegate is created, a delegate instance can refer to and call methods whose return type and parameter list match those specified by the delegate declaration.

➢ **Step 2 : Creating Delegate Method :**

- To initialize the delegate object with the address of any method, first the method should be created which is agree with a delegate declaration.
- The general form is shown below :

```
return-type method-name(parameter-list)
{
        // body
}
```

- The return type and parameter list must be match with delegate declaration.
- Once, the method is created you can initialize the delegate object with the address of this method.

- **Step 3 : Delegate Instantiation :**
- The object of the delegate has been created to store the address of method.
- This process is known as delegate instantiation.

- The general form is shown below :

  delegate-name obj-name = new delegate-name(method-name) ;

- Passing any method name without a pair of round brackets "( )" means assigning the address of method.
- In delegate instantiation, the method name which passed as argument to the delegate object must match its return-type and parameter-list with delegate.
- **Step 4 : Invoking (calling) Delegate :**
- Invoking delegate means calling a method by using the object of delegate.
- The general form is shown below :

  del-obj ( ) ;

  OR

  del-obj.Invoke ( ) ;

1) **Use of Delegate :**

a. In situation where you need to pass the method around the other method.

- Starting of Threads.
- Event Handling.
- Generic Class Library, etc..

➢ **Delegate Multicasting :**

- One of the most exciting features of a delegate is **its support for multicasting.**
- Multicasting is the **ability to create an invocation list or a chain of methods that will be automatically called when a delegate is invoked.**
- To add a method to the chain the **" += "** operator is used.
- To remove a method from chain the **" -= "** operator is used.

To add a method :-
Syntax :

```
delegate-name obj=new delegate-name(method1-name);
obj += new delegate-name(method2-name);
obj += new delegate-name(method3-name);
……
```

To remove a method :-
Syntax :

```
obj-= new delegate-name(method1-name);
obj -= new delegate-name(method2-name);
```

.....

- If the delegate returns a value then the value return by the last method becomes the return value of entire delegate.
- Thus, a delegate that makes use of multicasting will often have a void return type.
- Delegate multicasting is a powerful mechanism because they allow to define set of methods that can be executed as a unit.
- Delegate chins have special value to events.

➢ **Delegate Multicasting :**

- To add a method to the chain the " += " operator is used.
- To remove a method from chain the " -= " operator is used.

➢ **SUMMARY:-**

- Object that can refer to a method.
- Holds the address of method
- Similar to a "function pointer" in C and C++,
- Delegate is "type safe".

➢ There are four steps to use delegate :

- Step 1 : Declaration of Delegate :
- Step 2 : Creating Delegate Method :
- Step 3 : Delegate Instantiation :
- Step 4 : Invoking (calling) Delegate :

➢ **Events :**

- Event is a method that is **automatically executed when some action has occurred on GUI**.
- For example, user clicking a button control on GUI.
- Events can not only used on GUI but it **can be used in file-handling and database interaction.**
- When the event occurs, **all registered handlers are called**.
- Event handlers are **represented by delegates**.
- The object that raises the event is called "event source" and the object that responds to the event is called "event receiver".
- Events are members of a class and are declared using the "event" keyword.

- The general form to declare event is shown below :

  event event-delegate event-name ;

- Here, event-delegate is the name of the delegate used to support the event, and the event-name is the name of the specific event object being declared.
- Steps to create events are :
- **Step 1 : Create a delegate**
- **Step 2 : Create a class that defines the following**
  - An event that is created from the delegate.
  - Method that call the event.
- **Step 3 : Create another class that connects method to the event.**
  - This class must contain the following :
  - A constructor that takes a parameter of class type created in step 2.
  - The definition for the methods associated with the event.
- **Step 4 : Invoke the event**
- **Step 5 : Create an object of first class type**
- **Step 6 : Create an object of second class type**

➢ **Creating Event :**

- Events are the member of class and declared by using "event" keyword.
- Syntax :

  event event-delegate event-name ;

- The "delegate-name" is the name of delegate used to support the event.
- The "event-name" is the name of event object being declared.
- Example :

```
delegate void Mydel( ) ;
public event Mydel someEvent ;
```

➢ **Firing Event :**

- Firing an event's means calling the event handler for specific event when some action has occurred.
- Example :

```
public void onsomeEvent( )
{
        If(someEvent != null)
```

```
                    someEvent( );
                }
```

- **Chaining Event :**
  - Like a delegate, events can be multicast.
  - This enables multiple objects to respond to an event notification.
  - Chaining Event / Event Multicast: If multiple objects are handling the same event then it is called event chaining.
  - Syntax :

```
Obj-of-class.event-name += Handler-method1-name ;
Obj-of-class.event-name += Handler-method2-name ;
```

- **SUMMARY:-**
  - Automatically executed when some action has occurred on GUI.
  - Can be used in file-handling and database interaction.
  - When the event occurs, all registered handlers are called.
  - Event handlers are represented by delegates.
- **Creating Event :**
  - Events are the member of class and declared by using "event" keyword.
- **Firing Event :**
  - Firing an event's means calling the event handler for specific event when some action has occurred.
- **Chaining Event :**
  - Like a delegate, events can be multicast.
  - This enables multiple objects to respond to an event notification.
  - Chaining Event / Event Multicast: If multiple objects are handling the same event then it is called event chaining.
- **Collections :**
  - Collection is a **group of objects**.
  - The .Net Framework contains a large number of interfaces and classes that define and implement various types of collections.
  - The principal **benefit of collection is that they standardize the way group of objects are handled by programs**.

- All collections are design around a set of clearly define interface.
- Several built-in interfaces such as **ArrayList**, **HashTable**, **Stack** and **Queue**.
- Those are all include in **System.Collections** namespace.
- A collection class refers to a class that represents a collection of similar objects.
- For example, Mybook collection class that manage a collection of objects of the book class.
- The Mybook collection class provides a member of methods to modify the collection such as Add (), Remove (), Clear (), etc…
- The .Net Framework supports four general type of collections :

❖ **Non-Generic :**

- **ArrayList**
- **HashTable**
- **Stack**
- **Queue**
- **ShortedList**

➢ **ArrayList :**

- The ArrayList class **supports dynamic array** which can grow or sort as needed.
- In C#, standard arrays are of fixed length, which cannot be changed during program execution.
- It means that how many elements an array will hold must be known in advance.
- In some situations in which it is not known until the runtime how large an array will be, the ArrayList is used.
- An ArrayList is a **variable-length array of object references that can dynamically increase or decrease in size**.
- An ArrayList is **created with an initial size**.
- When this size is exceeded, the **collection is automatically enlarged**.
- When objects are removed, the array can be shrunk.
- An ArrayList has the following constructor :

public ArrayList( )
public ArrayList(ICollection c)
public ArrayList(int capacity)

- The first constructor builds an empty ArrayList with an initial capacity of zero.
- The second constructor builds an ArrayList that is initialized with the elements specified by c and has an initial capacity equal to the number of elements.
- The third constructor builds an ArrayList that has the specified initial capacity.
- The capacity is the size of array which grows automatically as elements are added to an ArrayList.
- ArrayList defines several methods described as follows :

Name	Description
Add(value) :	Used to add element in array.
CopyTo(Array A) :	Used to copy the ArrayList value into specific array variable. Array variable must be one dimensional.
IndexOf(value) :	Returns the index of the first occurrence of value, if not found it will return -1.
LastIndexOf(value) :	Returns the last index of the last occurrence of value, if not found it will return -1.
Reverse( ) :	Reverses the content of collection in ArrayList.
Remove(value) :	Removes the element specified by value in argument from ArrayList.
RemoveAt(int index) :	Removes the value located at specified index.
Sort( ) :	Sorts the collection in to ascending order.

- ArrayList defines several properties described as follows :

Name	Description
Capacity :	Used to get or set the capacity of ArrayList items.
Count :	Used to get the no. of items inside ArrayList. Counting of items.
Item :	Used to get particular item by giving Index value of item.

- Example of ArrayList is given below :

```
using System ;
using System.Collections ;
class Demo
{
        public static void Main( )
        {
```

```
                ArrayList A=new ArrayList( ) ;
                A.Add("A");
                A.Add("S");
                A.Add("P");
                Console.WriteLine("Number of Elements :" + A.Count) ;
                A.Remove("A") ;
                foreach (char ch in A)
                    Console.Write(ch + "  ") ;
                Console.ReadKey( ) ;
            }
        }
```

- **HashTable :**
  - HashTable stores a **Key Value pair type collection of data**.
  - We can **retrieve items from hashTable to provide the key . Both key and value are Objects.**
  - HashTable stores information using a mechanism called "hashing".
  - In "hashing", **the informational content of a key is used to determine a unique value, called "hash code".**
  - The "hash code" is used then as index at which the data associated with the key is stored in the table.
  - HashTable defines the following constructor :

public HashTable( )
public HashTable(int capacity)

  - The first form constructs a default HashTable.
  - The second form initializes the capacity of HashTable to capacity.
  - The methods that defines by HashTable is shown below :

Name	Description
Add( ) :	Used To add a pair of value in HashTable
Remove( ) :	Removes the specified key and corresponding value in HashTable.
ContainsKey(object o) :	Check if a specified key exist or not in HashTable.
ContainsValue(object o) :	Check the specified Value exist in HashTable

  - The HashTable supports for following properties.

Name	Description
Count :	Used To get the no. of items inside Hashtable. Counting of items.
Keys :	Gives collection of all the keyvalues given for HashTable items.
Values :	Gives collection of all the values given for HashTable Items.

- **Example :**

```
using System ;
using System.Collections ;
class Demo
{
        public static void Main( )
        {
            HashTable Ht=new HashTable( ) ;
            Ht.Add("R","Rajkot");
            Ht.Add("M","Mumbai");
            Ht.Add("D","Delhi");
            Ht.Remove("M");
            Icollection c = Ht.keys;
            foreach(string s in c)
                Console.WriteLine(s+" "+Ht[s]) ;
            Console.ReadKey( ) ;
        }
}
```

➢ **Stack :**

- Stack follows the **push-pop operations,** that is we can **Push Items into Stack and Pop it later** also it follows the **Last In First Out (LIFO) system.** That is we can push the items into a stack and get it in reverse order. Stack returns the last item first.
- The real life example is **Stack of Plats on the table.**
- The first plat put down is the last one to be picked up.
- The Stack defines following constructors:

  public Stack( )
  public Stack(int capacity)
- Stack defines several methods described as follows :

Name	Description
Clear( ) :	Clears all elements in Stack and set count to '0'.
Contains(value) :	Returns true if value is in a Stack, otherwise it returns false.

Peek( ) :	Return the object at the front of the invoking Stack but does not remove original value.
Push(object) :	Adds the element in Stack on the top of Stack.
Pop() :	Returns the value from the top of Stack and remove it from Stack.
Name	**Description**
Count( ) :	Used to get or set the total number of elements in Stack.

- **Example :**

```
using System ;
using System.Collections ;
class Demo
{
        public static void Main( )
        {
            Stack St=new Stack( ) ;
            St.Push("Rajkot");
            St.Push("Mumbai");
            St.Push("Delhi");
            St.Push("Pune");

            for (int i=0; i<St.Count; i++)
                Console.WriteLine(St.Pop()) ;

            Console.ReadKey( ) ;
        }
}
```

- **Queue :**
  - It is also an another type of **data structure which is FIFO(first in first out) type**.
  - The **first item put in a queue is the first item retrieved.**
  - The collection class that supports a queue is called **"Queue".**
  - Queue is a **dynamic collection that grows as needed to accommodate the elements it must store.**
  - Queue works like First In First Out method and the item added first in the Queue is first get out from Queue. We can Enqueue (add) items in Queue and we can Dequeue (remove from Queue) or we can Peek (that is get the reference of first item added in Queue) the item from Queue.
  - Queue has several constructors as shown below:

public Queue( )
public Queue(ICollection c)
public Queue(int capacity , float growfact)

- The third constructor always specifies a growth factor in "growfact" which must be between 1.0 and 10.0, by default it is 2.0.
- Queue defines several properties described as follows :

Name	Description
Count( ) :	Used to get or set the total number of elements in Queue.

- Example of Queue is given below :

```
using System ;
using System.Collections ;
class Demo
{
        public static void Main( )
        {
            Queue Q=new Queue( ) ;
            Q.Enqueue(18);
            Q.Enqueue(73);
            Q.Enqueue(35);
            Q.Enqueue(50);
            foreach (int i in Q )
                Console.WriteLine(i) ;
            Q.Dequeue( ); // remove element 18
          Console.WriteLine("Now top element is : "+Q.Peek( ));
            Console.ReadKey( ) ;
        }
}
```

➢ **SortedList :**

- SortedList creates a collection **that stores a key/value pairs in sorted order, based on the value of the keys.**
- SortedList has several constructors as shown below:

public SortedList ( )
public SortedList (int capacity)

- The first constructor builds an empty collection with an initial capacity of zero.
- The second constructor builds an empty SortedList that has the initial capacity specified by "capacity".

- The capacity of SortedList grows automatically as needed when elements are added to the list.
- When the current capacity is exceeded, the capacity is increased.
- SortedList defines several methods described as follows :

Name	Description
ContainsKey(key) :	Returns true if key is in a SortedList otherwise it returns false.
ContainsValue(value) :	Returns true if value is in a SortedList, otherwise it returns false.
GetKey(int index) :	Returns the value of the key located at index specified in argument.
IndexOfKey(key) :	Returns the index of the key specified by key, if not found then returns -1.
IndexOfValue(value) :	Returns the index of the value specified by value, if not found then returns -1.
GetByIndex(int index) :	Returns the value located at index specified by index.

- Example of SortedList is given below :

```
using System ;
using System.Collections ;
class Demo
{
        public static void Main( )
        {
            SortedList S=new SortedList( ) ;
            S.Add("c","pointer");
            S.Add("c++","class");
            S.Add("java","Applet");
            S["J2EE"]="JDBC";
            for (int i=0;i<S.count;i++)
                Console.WriteLine(S.GetByIndex(i)) ;
            Console.ReadKey( ) ;
        }
}
```

- **SUMMARY:-**
  - Collection is a group of objects.
  - Those are all include in System.Collections namespace.

❖ **Collection:**

- **ArrayList**
- **HashTable**
- **Stack**
- **Queue**
- **ShortedList**

➢ **ArrayList :**
- Supports dynamic array which can grow or sort as needed.
- It can dynamically increase or decrease in size.
- An ArrayList is created with an initial size.
- When objects are removed, the array can be shrunk.

➢ **HashTable :**
- Stores a Key Value pair type collection of data.
- We can retrieve items from hashTable to provide the key.
- It use hash code.

➢ **Stack :**
- Follows the push-pop operations.
- Use Last In First Out (LIFO) system.
- Push the items into a stack and get it in reverse order.
- Stack returns the last item first.

➢ **Queue :**

- Type of data structure
- Use FIFO(first in first out) type.
- The first item put in a queue is the first item retrieved.

➢ **SortedList :**

- Creates a collection that stores a key/value pairs in sorted order
- Based on the value of the keys.
- The capacity of SortedList grows automatically.

CH : 5 Windows Forms & Control Programming

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.

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 how a 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 using 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.
Enter	Occurs when the control receives focus.
Leave	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.

- **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**

  - Default Button can have following values:

1. **DefaultDesktopOnly**	3. **RTLReading**
2. **RightAlign**	4. **ServiceNotification**

➢ Basic GUI Controls :

➢ BUTTON

- Button class in Windows Forms represents a Button control.
- **A Button control is a child control placed on a Form and used to process click event and can be clicked by a mouse click or by pressing ENTER or ESC keys.**

➢ Creating a Button

- To create a Button control, you simply drag and drop a Button control from Toolbox to Form in Visual Studio.
- After you drag and drop a Button on a Form, the Button looks like Figure 1. Once a Button is on the Form, you can move it around and resize it using mouse.

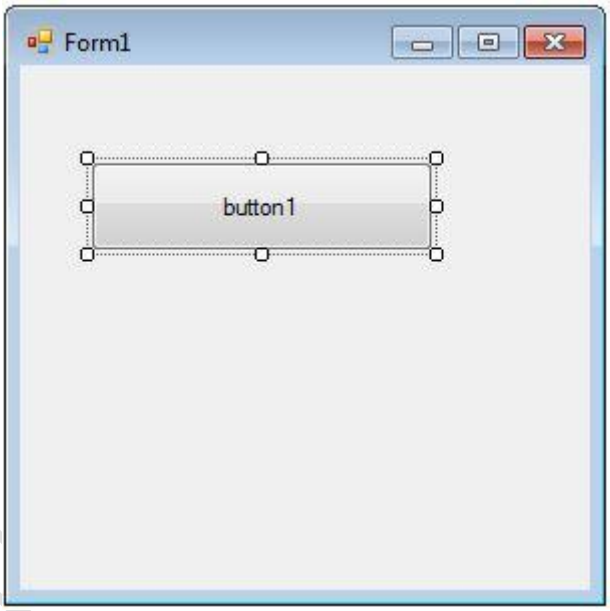


Figure 1

➢ Setting Button Properties

- After you place a Button control on a Form, the next step is to set button properties.
- The easiest way to set a Button control properties is by using the Properties Window. You can open Properties window by pressing F4 or right click on a control and select Properties menu item. The Properties window looks like Figure 2.

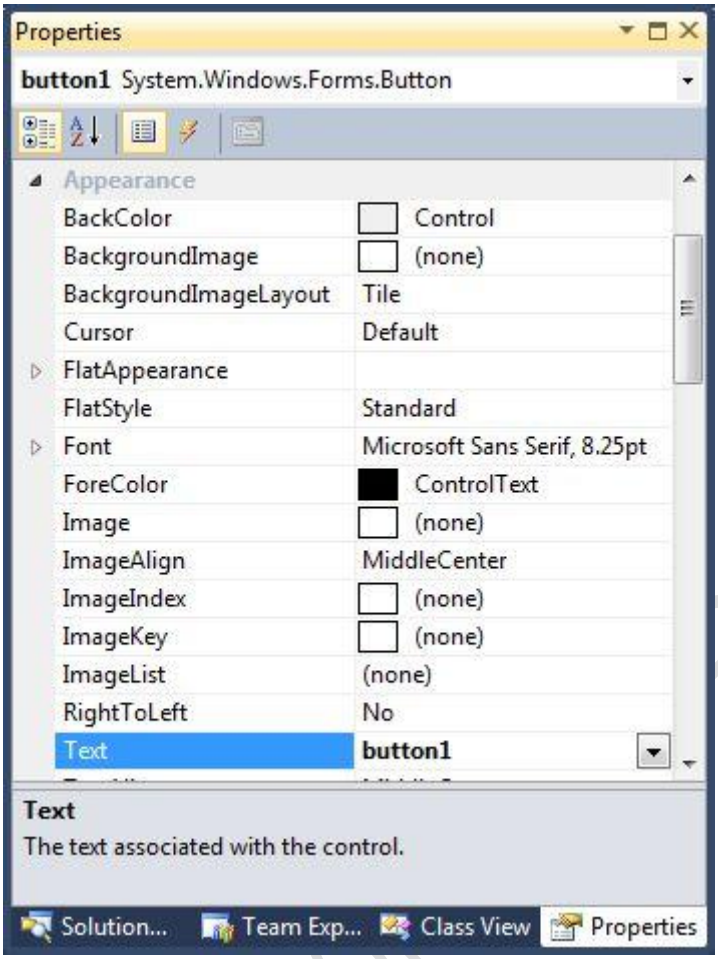


Figure 2

- **Background and Foreground**
  - BackColor and ForeColor properties are used to set background and foreground color of a Button respectively. If you click on these properties in Properties window, the Color Dialog pops up.
  - Alternatively, you can set background and foreground colors at run-time. The following code snippet sets BackColor and ForeColor properties.

```
// Set background and foreground
dynamicButton.BackColor = Color.Red;
dynamicButton.ForeColor = Color.Blue;
```

- **AutoEllipsis**

- An ellipsis character (...) is used to give an impression that a control has more characters but it could not fit in the current width of the control. Figure 3 shows an example of an ellipsis character.

Figure 3

- If AutoEllipsis property is true, it adds ellipsis character to a control if text in control does not fit. You may have to set AutoSize to false to see the ellipses character.

Image in Button

- The Image property of a Button control is used to set a button background as an image.
- The Image property needs an Image object. The Image class has a static method called FromFile that takes an image file name with full path and creates an Image object.
- You can also align image and text. The ImageAlign and TextAlign properties of Button are used for this purpose.
- The following code snippet sets an image as a button background.

```
// Assign an image to the button.
dynamicButton.Image = Image.FromFile(@"C:\Images\Dock.jpg");
// Align the image and text on the button.
dynamicButton.ImageAlign = ContentAlignment.MiddleRight;
dynamicButton.TextAlign = ContentAlignment.MiddleLeft;
// Give the button a flat appearance.
dynamicButton.FlatStyle = FlatStyle.Flat;
```

➢ **Text and Font**

- The Text property of Button represents the contents of a Button. The TextAlign property if used to align text within a Button that is of type ContentAlignment enumeration.
- The Font property is used to set font of a Button.
- The following code snippet sets Text and Font properties of a Button control.

```
dynamicButton.Text = "I am Dynamic Button";
dynamicButton.TextAlign = ContentAlignment.MiddleLeft;
dynamicButton.Font = new Font("Georgia", 16);
```

- **Button States**
  - Button control has five states - Normal, Flat, Inactive, Pushed, and All. ButtonState enumeration represents a button state.
  - Unfortunately, Windows Forms does not have a straight-forward way to set a button control state but there is a work around.
  - Windows Forms has a ControlPaint class with some static methods that can be used to draw various controls at runtime. The DrawButton method is used to draw a Button control and the last parameter of this method is ButtonState enumeration.
  - The following code snippet sets the button state of button1 using ControlPaint class.
  - ControlPaint.DrawButton(System.Drawing.Graphics.FromHwnd(button1.Handle), 0, 0, button1.Width, button1.Height, ButtonState.Pushed);
- **Adding Button Click Event Hander**
  - A Button control is used to process the button click event. We can attach a button click event handler at run-time by setting its Click event to an EventHandler obect. The EventHandler takes a parameter of an event handler. The Click event is attached in the following code snippet.
- **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 entire 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 :**

Name	Description
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.

➢ **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.

➢ **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. 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.

- **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-down. 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 can be 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 items are 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 it 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 is not 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.

- **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.

➢ **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.

➢ **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 returns -1.

- **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.

➢ **MENUSTRIP**
- The MenuStrip class is the foundation of menus functionality in Windows Forms.
- If you have worked with menus in .NET 1.0 and 2.0, you must be familiar with the MainMenu control. In .NET 3.5 and 4.0, the MainMenu control is replaced with the MenuStrip control.

➢ **Creating a MenuStrip**
- We can create a MenuStrip control using a Forms designer at design-time or using the MenuStrip class in code at run-time or dynamically.
- To create a MenuStrip control at design-time, you simply drag and drop a MenuStrip control from Toolbox to a Form in Visual Studio.

- After you drag and drop a MenuStrip on a Form, the MenuStrip1 is added to the Form and looks like Figure 1. Once a MenuStrip is on the Form, you can add menu items and set its properties and events.

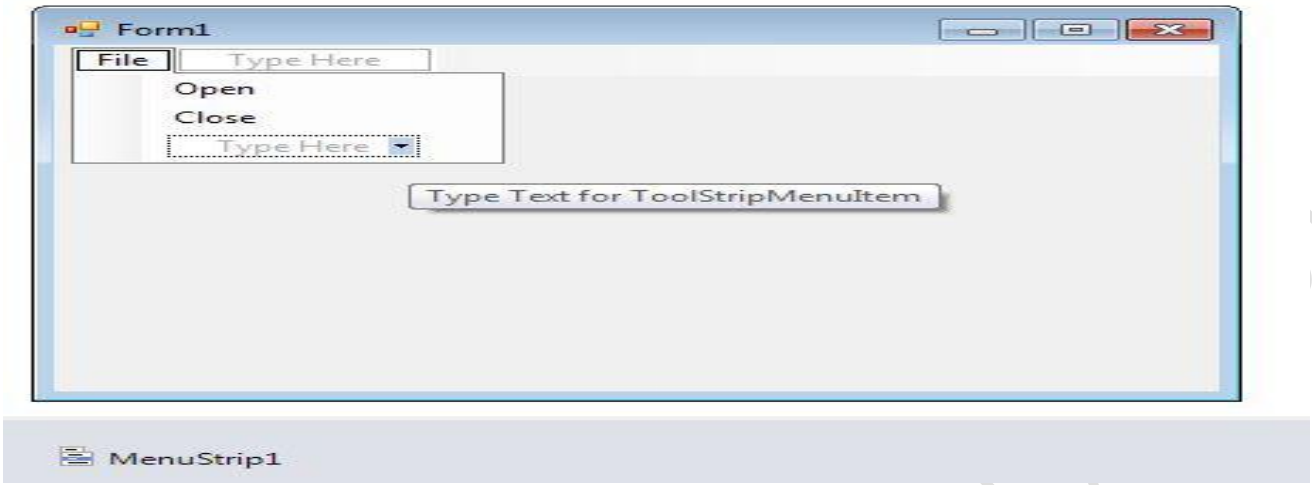


Figure 1

- Creating a MenuStrip control at run-time is merely a work of creating an instance of MenuStrip class, set its properties and adds MenuStrip class to the Form controls.
- First step to create a dynamic MenuStrip is to create an instance of MenuStrip class. The following code snippet creates a MenuStrip control object.

C# Code:

```
MenuStrip MainMenu = new MenuStrip();
```

- In the next step, you may set properties of a MenuStrip control.
- The following code snippet sets background color, foreground color, Text, Name, and Font properties of a MenuStrip.

C# Code:

```
MainMenu.BackColor = Color.OrangeRed;
MainMenu.ForeColor = Color.Black;
MainMenu.Text = "File Menu";
MainMenu.Font = new Font("Georgia", 16);
```

- Once the MenuStrip control is ready with its properties, the next step is to add the MenuStrip to a Form.
- To do so, first we set MainMenuStrip property and then use Form.Controls.Add method that adds MenuStrip control to the Form controls and displays on the Form based on the location and size of the control.
- The following code snippet adds a MenuStrip control to the current Form.

C# Code:

```
this.MainMenuStrip = MainMenu;
Controls.Add(MainMenu);
```

➢ **Setting MenuStrip Properties**

- After you place a MenuStrip control on a Form, the next step is to set properties.

- The easiest way to set properties is from the Properties Window. You can open Properties window by pressing F4 or right click on a control and select Properties menu item. The Properties window looks like Figure 2.

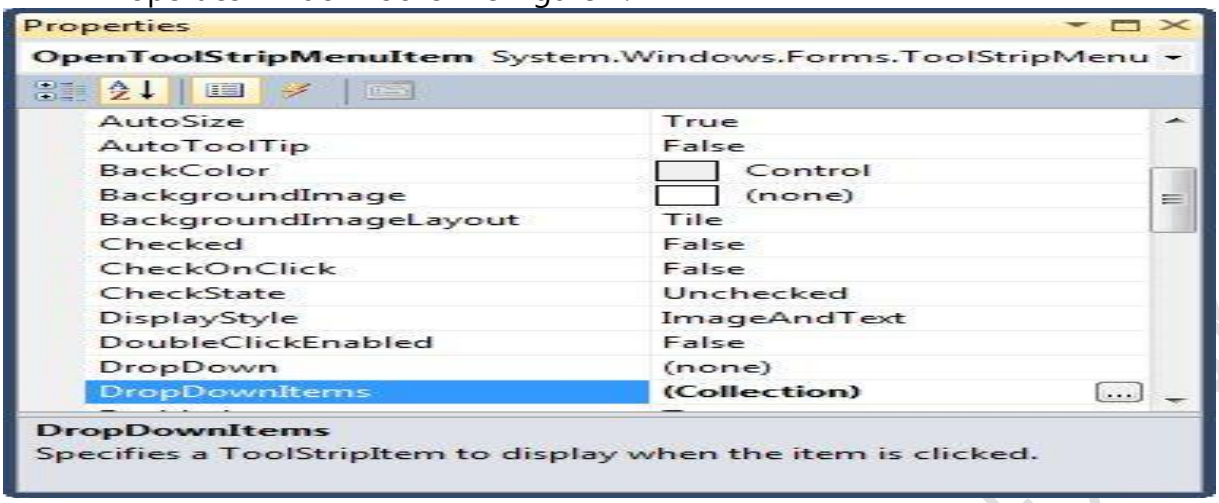


Figure 2

- **Name**
  - Name property represents a unique name of a MenuStrip control. It is used to access the control in the code. The following code snippet sets and gets the name and text of a MenuStrip control.

    C# Code:

    ```
    MainMenu.Name = "MainMenu";
    ```

- **Positioning a MenuStrip**
  - The Dock property is used to set the position of a MenuStrip.
  - It is of type DockStyle that can have values Top, Bottom, Left, Right, and Fill. The following code snippet sets Location, Width, and Height properties of a MenuStrip control.

    C# Code:

    ```
    MainMenu.Dock = DockStyle.Left;
    ```

- **Font**
  - Font property represents the font of text of a MenuStrip control. If you click on the Font property in Properties window, you will see Font name, size and other font options.
  - The following code snippet sets Font property at run-time.

    C# Code:

    ```
    MainMenu.Font = new Font("Georgia", 16);
    ```

- **Background and Foreground**
  - BackColor and ForeColor properties are used to set background and foreground color of a MenuStrip respectively.
  - If you click on these properties in Properties window, the Color Dialog pops up.
  - Alternatively, you can set background and foreground colors at run-time. The following code snippet sets BackColor and ForeColor properties.

    C# Code:

    ```
    MainMenu.BackColor = Color.OrangeRed;
    ```

```
MainMenu.ForeColor = Color.Black;
```

- The new MenuStrip with background and foreground looks like Figure 3.

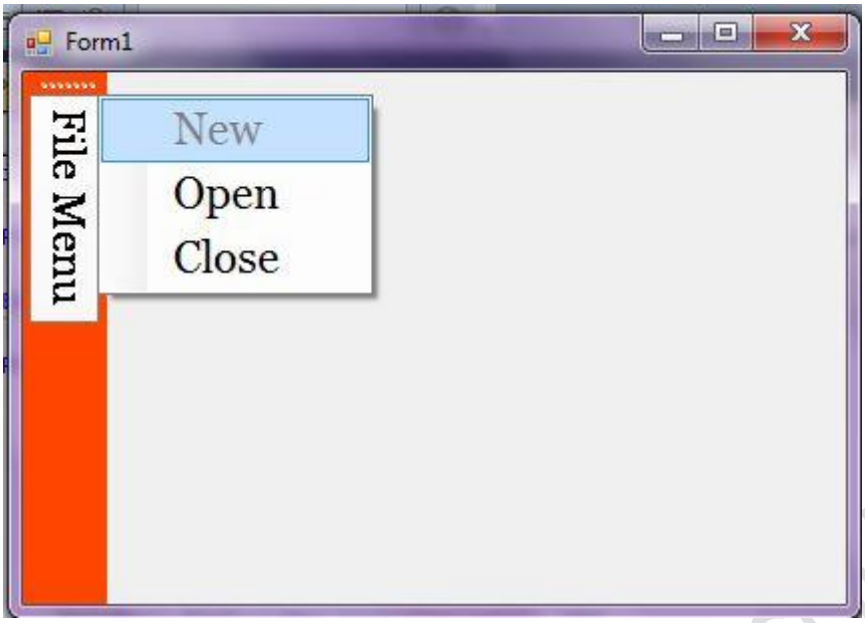


Figure 3

- **MenuStrip Items**
  - A Menu control is nothing without menu items.
  - The Items property is used to add and work with items in a MenuStrip.
  - We can add items to a MenuStrip at design-time from Properties Window by clicking on Items Collection as you can see in Figure 4.

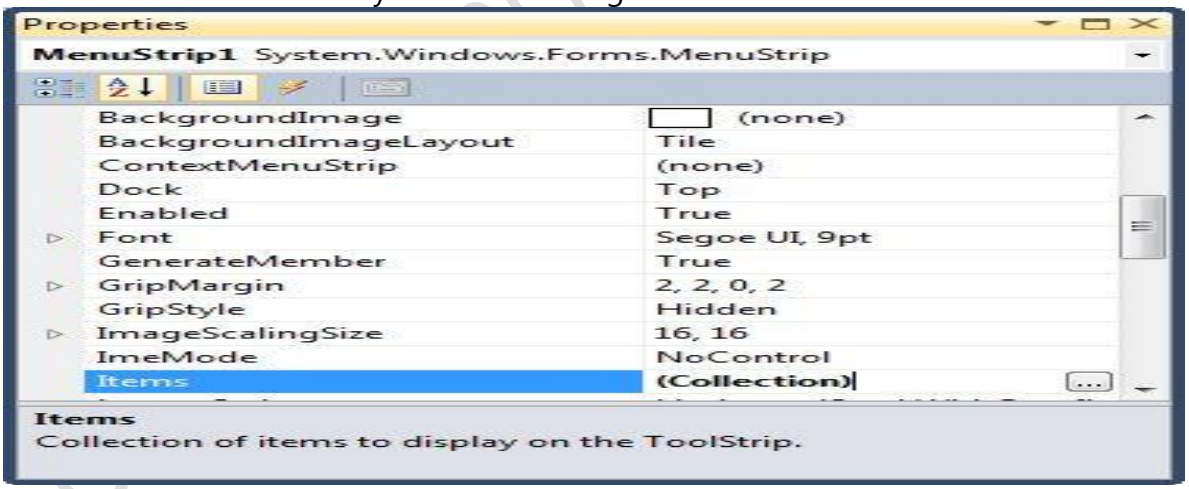


Figure 4

  - When you click on the Collections, the String Collection Editor window will pop up where you can type strings. Each line added to this collection will become a MenuStrip item. I add four items as you can see from Figure 5.

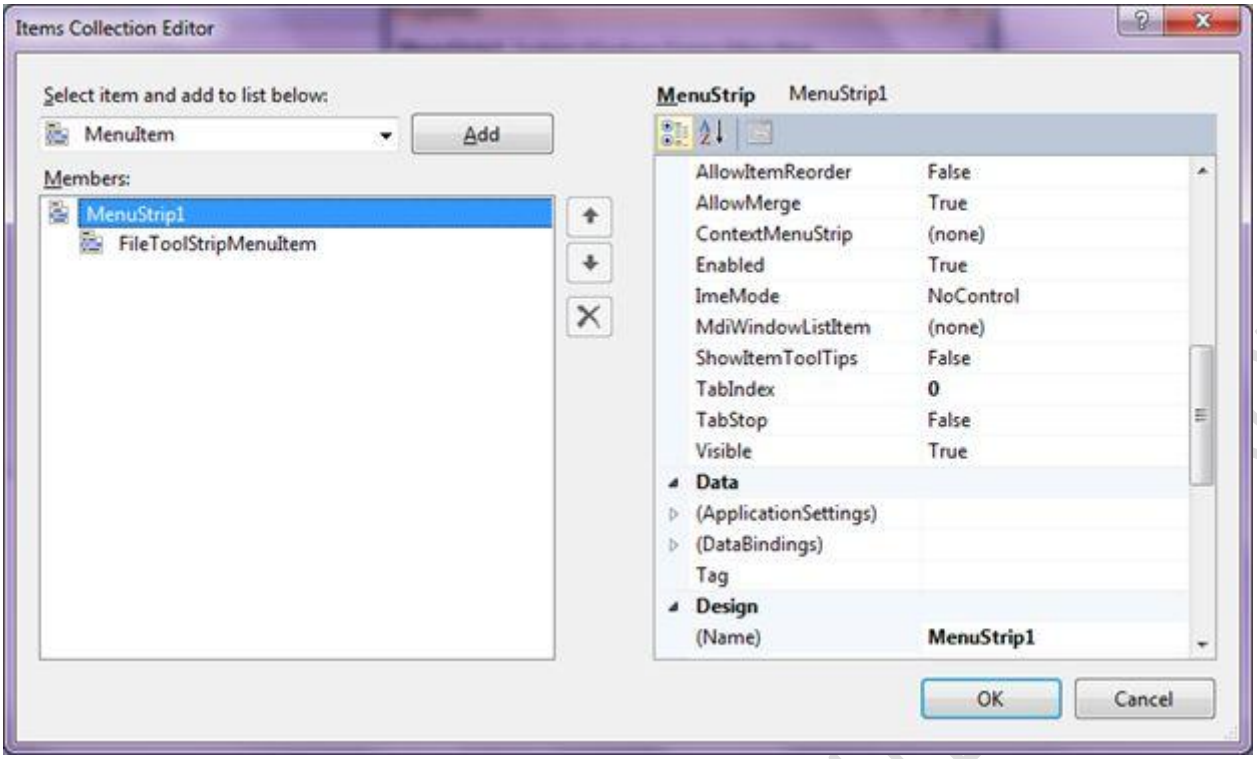


Figure 5

- A ToolStripMenuItem represents a menu items. The following code snippet creates a menu item and sets its properties.

C# Code:

```
// Create a Menu Item
ToolStripMenuItem FileMenu = new ToolStripMenuItem("File");
FileMenu.BackColor = Color.OrangeRed;
FileMenu.ForeColor = Color.Black;
FileMenu.Text = "File Menu";
FileMenu.Font = new Font("Georgia", 16);
FileMenu.TextAlign = ContentAlignment.BottomRight;
FileMenu.ToolTipText = "Click Me";
```

- Once a menu item is created, we can add it to the main menu by using MenuStrip.Items.Add method. The following code snippet adds FileMenu item to the MainMenu.

C# Code:

```
MainMenu.Items.Add(FileMenu);
```

➢ **Adding Menu Item Click Event Handler**

- The main purpose of a menu item is to add a click event handler and write code that we need to execute on the menu item click event handler.
- For example, on File >> New menu item click event handler, we may want to create a new file.
- To add an event handler, you go to Events window and double click on Click and other as you can see in Figure 6.

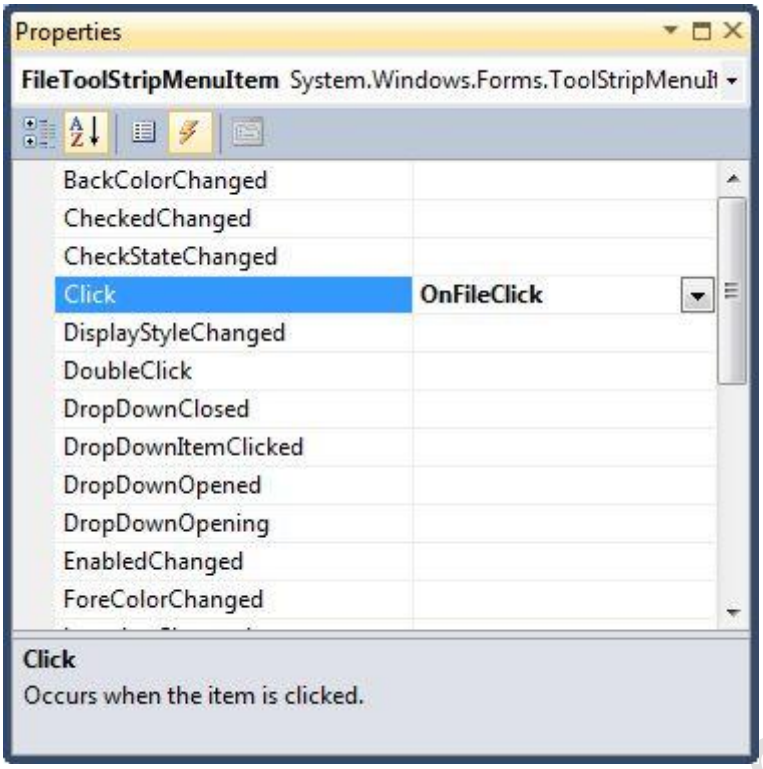


Figure 6

- We can also define and implement an event handler dynamically. The following code snippet defines and implements these events and their respective event handlers.

C# Code:

```
FileMenu.Click += new System.EventHandler(this.FileMenuItemClick);

    private void FileMenuItemClick(object sender, EventArgs e)
    {
        MessageBox.Show("File menu item clicked");
    }
```

Summary

we discussed discuss how to create menus using the MenuStrip control. First we discussed how to create menus at design-time and run-time. After that we saw, how to set menus properties and click event handlers.

- **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.
- 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 images.
ToolStripComboBox	A ComboBox with methods/properties to configure various styles.
ToolStripSeparator	A separator that you can use to visually separate groups of ToolStripItem elements.
ToolStripDropDownButton	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 Form controls.

- All these controls are usual controls which you use in developing Windows Forms. The only difference is they appear under ToolStrip. ToopStrip 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.

➢ **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).

➢ **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 different groups under Border. Although they have BorderStyle property but bydefault BorderStyle is set to None.	Generally GroupBox is used to display group along with Border.

- **Properties of Panel :**

Name	Description
AutoScroll	Gets or sets a value indicating whether the container enables the user to scroll to any 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.

➢ **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
AllowFullOpen	Gets/sets whether the user can use the dialog box to define custom colors.
AnyColor	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.

CustomColors	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 box is opened
ShowHelp	Gets/sets whether a Help button appears in the color dialog box.
SolidColorOnly	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
CheckFileExists	Gets/sets if the dialog box displays a warning if the user specifies a nonexistent file.
CheckPathExists	Gets/sets whether the dialog box displays a warning if the user gives a path that does not exist.
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 as file type" or "Files of type" box.
FilterIndex	Gets/sets the index of the filter selected in the file dialog box.
InitialDirectory	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 by the 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.
FixedPitchOnly	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.
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.

Multiple-Document Interface (MDI) Applications

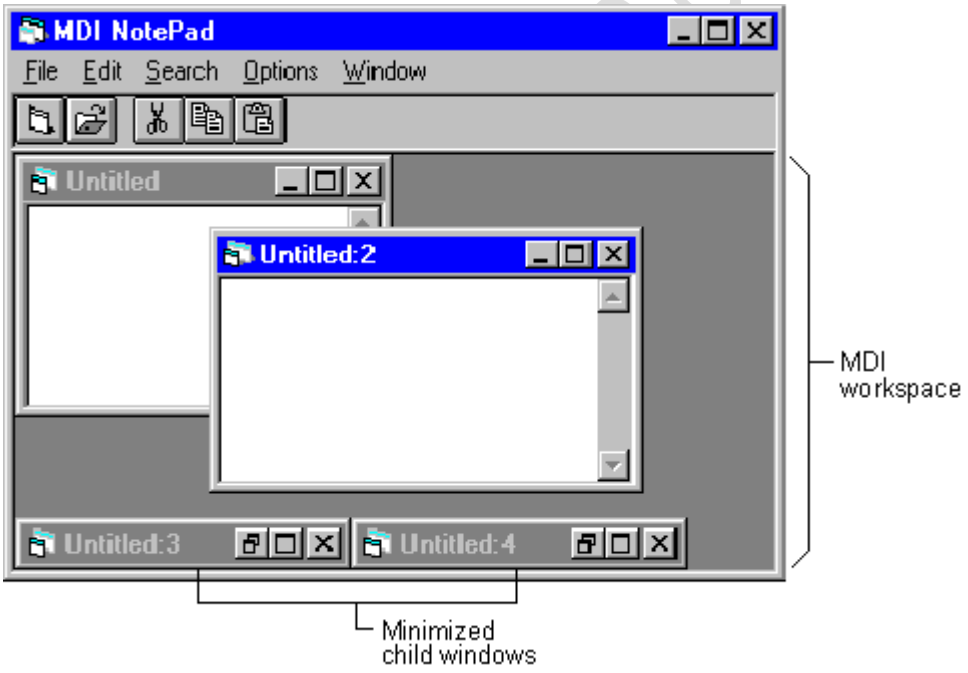
The multiple-document interface (MDI) allows you to create an application that maintains multiple forms within a single container form. Applications such as Microsoft Excel and Microsoft Word for Windows have multiple-document interfaces.

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



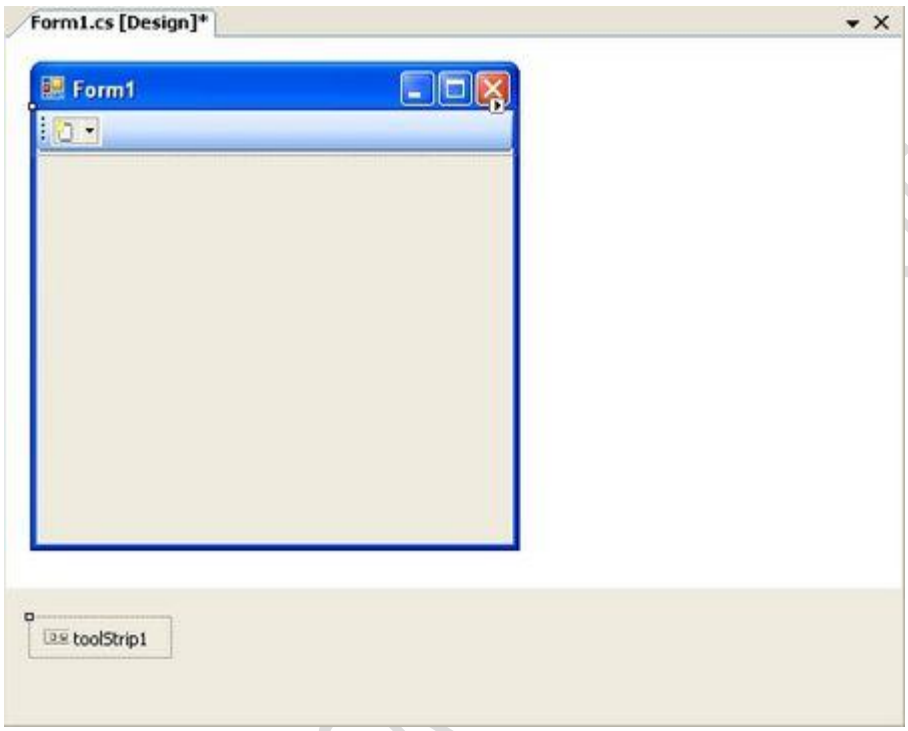
Note Your application can also include standard, non-MDI forms that are not contained in the MDI form. A typical use of a standard form in an MDI application is to display a modal dialog box.

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).

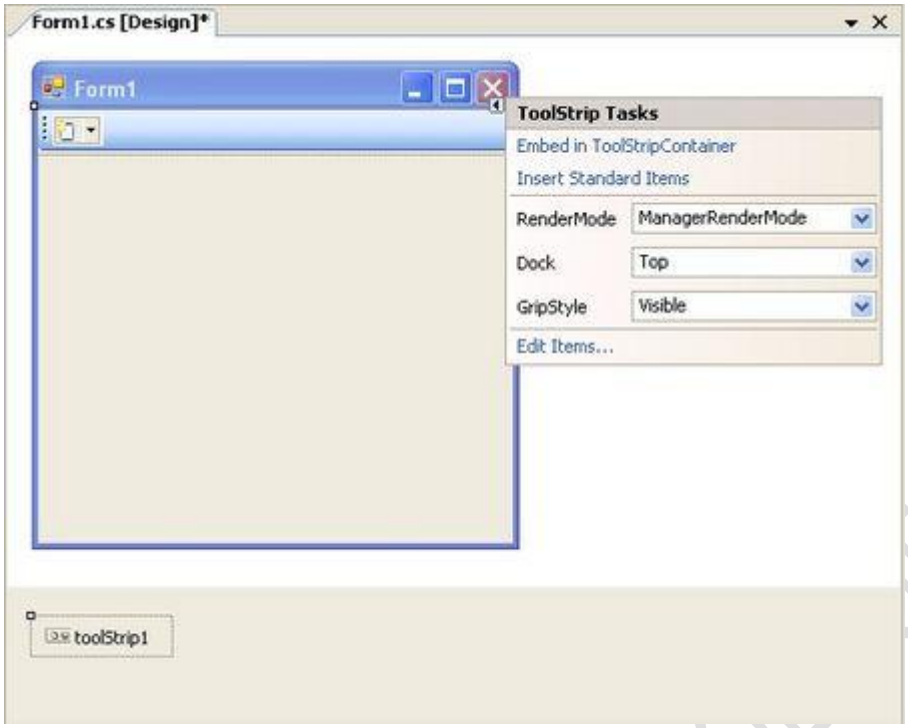
Create and configure menus

To best see the ToolStrip in action, let us create a simple notepad application.

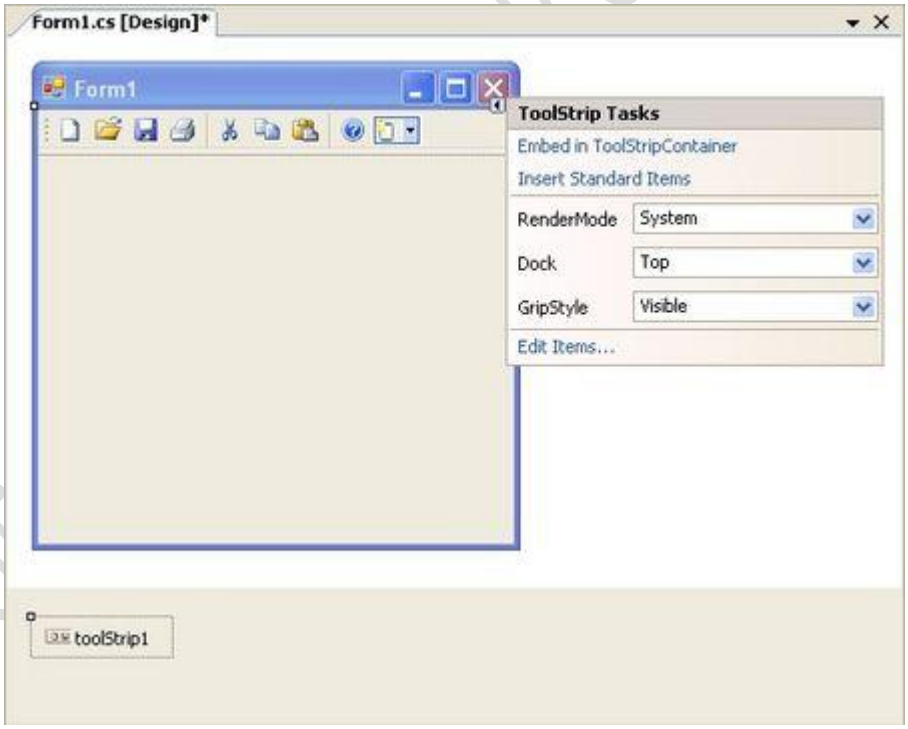
1. Create a new project
2. Add a ToolStrip to a Form



3. Insert Standard Items by clicking on the Smart Tag icon



4. Change the RenderMode to **System**

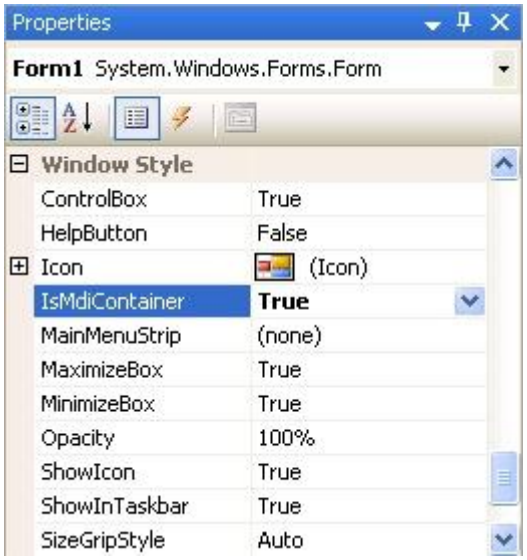


5. Create event handlers for all of the buttons on the ToolStrip by double-clicking on them.

- New
- Open
- Save
- Print
- Cut
- Copy
- Paste

```
        private void newToolStripButton_Click(object sender, EventArgs e)
        {
        }
        private void openToolStripButton_Click(object sender, EventArgs e)
        {
        }
        private void saveToolStripButton_Click(object sender, EventArgs e)
        {}
}
        private void printToolStripButton_Click(object sender, EventArgs e)
        {
        }
        private void cutToolStripButton_Click(object sender, EventArgs e)
        {
        }
        private void copyToolStripButton_Click(object sender, EventArgs e)
        {
        }
        private void pasteToolStripButton_Click(object sender, EventArgs e)
        {
        }
```

6. Change the Form to an MDI Form



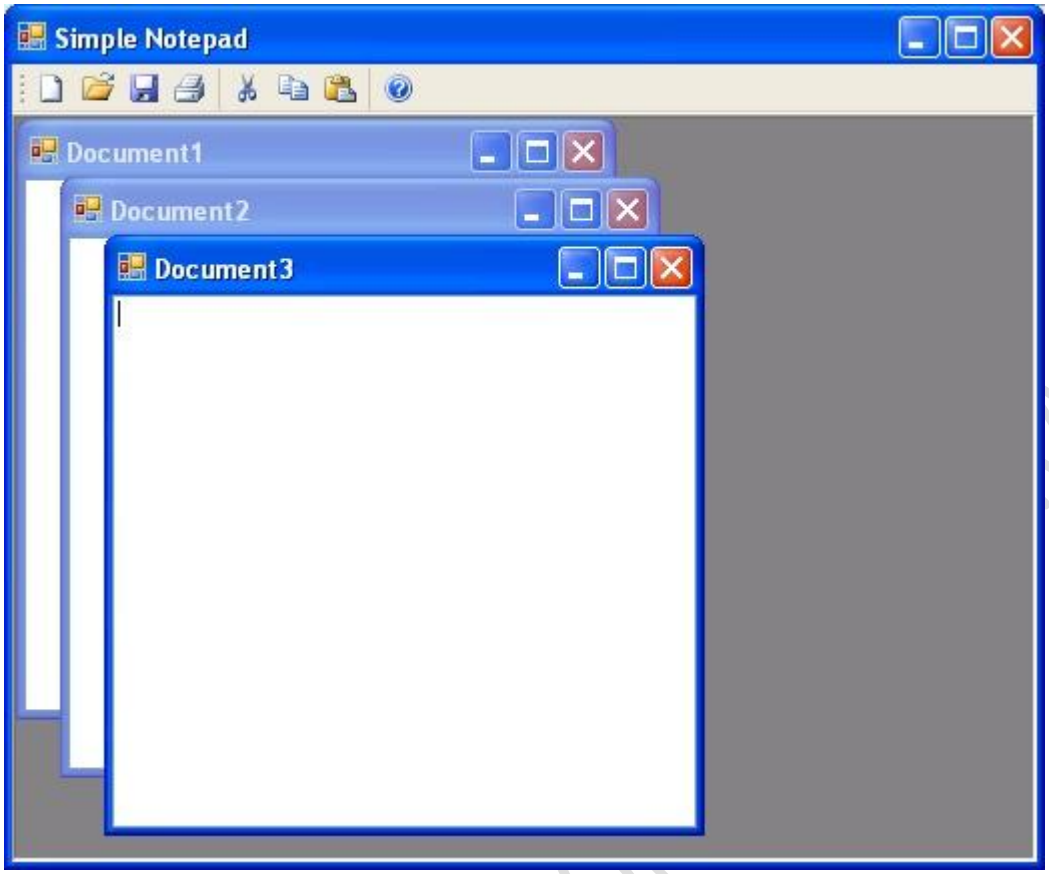
7. Modify the code of the form to the following

```
int count;
Form mdiChild;
TextBox editTextBox;
public Form1()
{
        InitializeComponent();
        count = 1;
}
```

8. Edit the newToolStripButton_Click method with the following code:

```
private void newToolStripButton_Click(object sender, EventArgs e)
{
  mdiChild = new Form();
  mdiChild.Text = "Document" + count.ToString();
  mdiChild.MdiParent = this;
  editTextBox = new TextBox();
  editTextBox.Multiline = true;
  editTextBox.Dock = DockStyle.Fill;
  mdiChild.Controls.Add(editTextBox);
  mdiChild.Show();
  count++;
 }
```

9. Test the new method



10. Edit the cutToolStripButton_Click method with the following code:

```
private void cutToolStripButton_Click(object sender, EventArgs e)
{
  Form activeChildForm = this.ActiveMdiChild;

  if (activeChildForm != null)
  {
    TextBox activeTextBox = activeChildForm.ActiveControl as TextBox;
    if (activeTextBox != null)
    {
      activeTextBox.Cut();
    }
  }
}
```

11. Edit the copyToolStripButton_Click method with the following code:

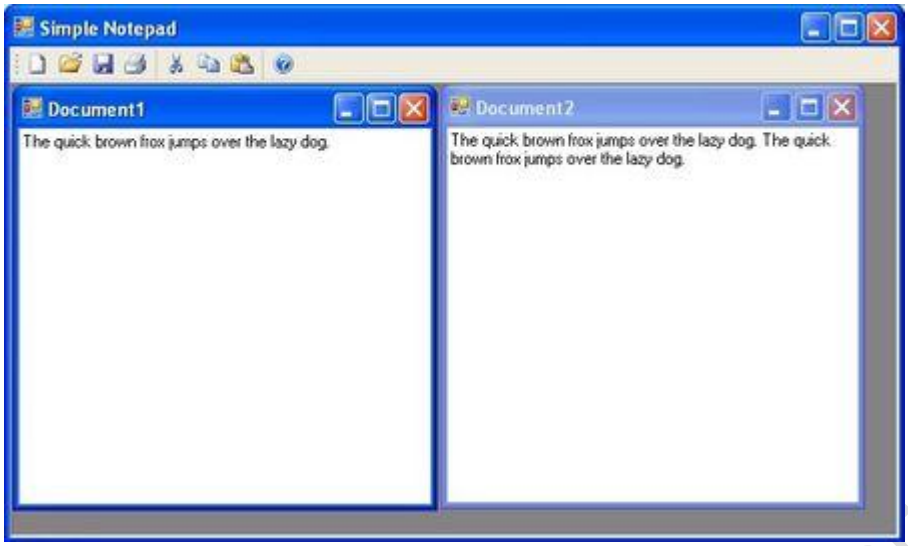
```
private void copyToolStripButton_Click(object sender, EventArgs e)
{
```

```
  Form activeChildForm = this.ActiveMdiChild;

  if (activeChildForm != null)
  {
    TextBox activeTextBox = activeChildForm.ActiveControl as TextBox;
    if (activeTextBox != null)
    {
      activeTextBox.Copy();
    }
  }
}
```

12. Edit the pasteToolStripButton_Click method with the following code:

```
private void pasteToolStripButton_Click(object sender, EventArgs e)
{
  Form activeChildForm = this.ActiveMdiChild;

  if (activeChildForm != null)
  {
    TextBox activeTextBox = activeChildForm.ActiveControl as TextBox;
    if (activeTextBox != null)
    {
      activeTextBox.Paste();
    }
  }
   }
```

13. Test the Cut, Copy and Paste methods



14. Edit the saveToolStripButton_Click method with the following code:

```
private void saveToolStripButton_Click(object sender, EventArgs e)
{
  SaveFileDialog sfd = new SaveFileDialog();

  sfd.Title = "Save a Text File";
  sfd.Filter = "Text File (*.txt)|*.txt|All Files (*.*)|*.*";

  DialogResult dr = sfd.ShowDialog();
  if (dr == DialogResult.OK)
  {
    System.IO.StreamWriter sw = new System.IO.StreamWriter(sfd.FileName);

    Form activeChildForm = this.ActiveMdiChild;

    if (activeChildForm != null)
    {
      TextBox activeTextBox = activeChildForm.ActiveControl as TextBox;

      if (activeTextBox != null)
      {
        sw.Write(activeTextBox.Text);
      }

      sw.Close();
    }
  }
```

```
}
```

15. Edit the openToolStripButton_Click method with the following code:

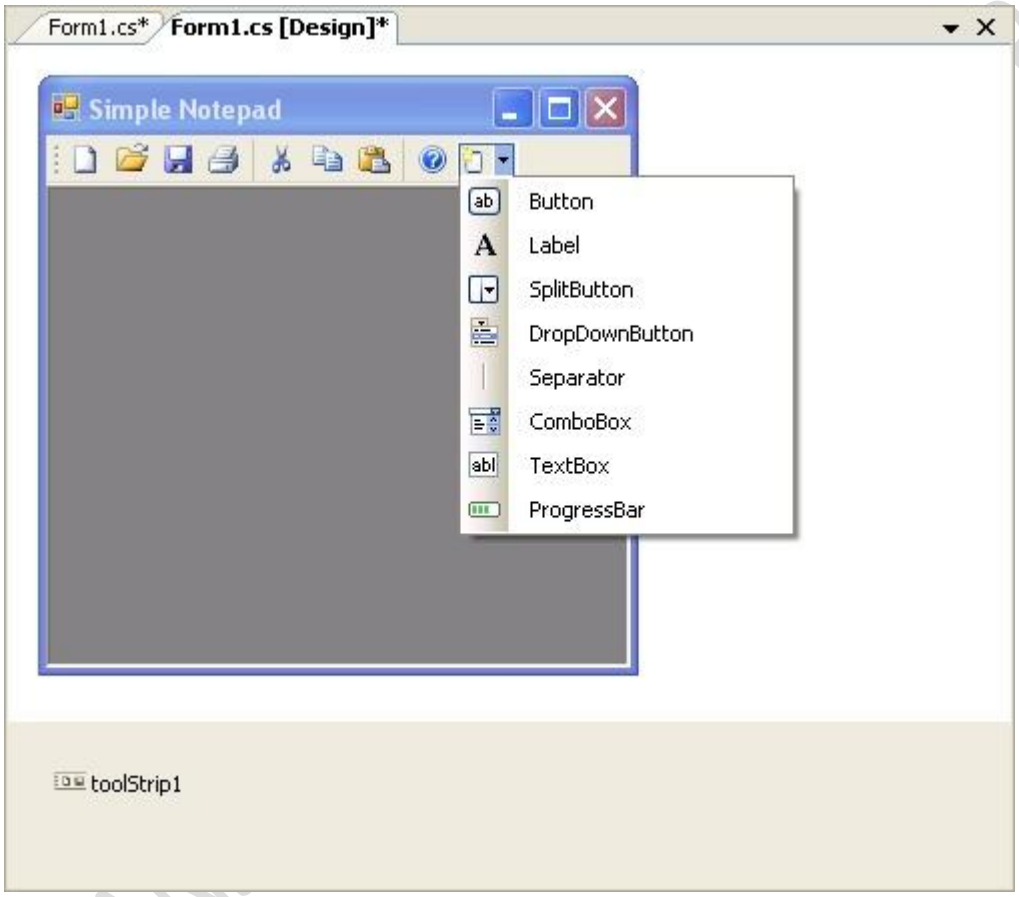
```
private void openToolStripButton_Click(object sender, EventArgs e)
{
  OpenFileDialog ofd = new OpenFileDialog();

  ofd.Title = "Open a Text File";
  ofd.Filter = "Text File (*.txt)|*.txt|All Files (*.*)|*.*";

  DialogResult dr = ofd.ShowDialog();
  if (dr == DialogResult.OK)
  {
    System.IO.StreamReader sr = new System.IO.StreamReader(ofd.FileName);

    Form activeChildForm = this.ActiveMdiChild;

    if (activeChildForm != null)
    {
      TextBox activeTextBox = activeChildForm.ActiveControl as TextBox;

      if (activeTextBox != null)
      {
        activeTextBox.Text = sr.ReadToEnd();
      }

      sr.Close();
    }
  }
}
```

16. Edit the printToolStripButton_Click method with the following code:

```
/// <summary>
    /// This displays the Print dialog only, it does not print the document
    /// </summary>
/// <param name="sender"></param>
    /// <param name="e"></param>
    private void printToolStripButton_Click(object sender, EventArgs e)
    {
      PrintDialog pd = new PrintDialog();
      pd.ShowDialog();
```

}

17. We can add more controls to the ToolStrip such as:
    - Button
    - Label
    - SplitButton
    - DropDownButton
    - Separator
    - ComboBox
    - TextBox
    - ProgressBar



Windows Forms :

- Forms allow us to work visually with controls and other items from the toolbox.
- based on the System.Windows.Forms namespace.
- Forms can be standard windows, multiple document interface (MDI) windows, dialog boxes, or display surfaces for graphical routines.

Controls used in Form :

- 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.

- **MessageBox :**
  - It will **display message or string in another window**.
- **Label :**
  - 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.
  - The TextAlign (formerly Alignment) property allows you to set the alignment of the text within the label.
- **TextBox :**
  - 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.
- **RadioButton :**

- The RadioButton control can display text, an Image, or both.
- To create multiple groups on one form, place each group in its own container, such as a GroupBox or Panel control.
- 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.

➢ **CheckBox :**

- When you select a checkbox, a check appears in it, indicating that the box is indeed selected.
- 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.

➢ **ComboBox :**

- 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.

➢ **ListBox :**

- 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.
- You also can support multiple selections in list boxes.

➢ **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.

➢ **PictureBox :**

- Picture boxes are used to display graphics from a bitmap, icon, JPEG, GIF or other image file type.

➢ **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.

➢ **TreeView :**

- A TreeView control displays a hierarchical list of Node objects, each of which consists of a label and an optional bitmap.
- You use a tree view to display a hierarchy of nodes. Each node is not only displayed visually, but also can have child nodes.
- You can expand and collapse parent nodes by clicking them; when expanded, their children are also visible.

➢ **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.
- 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.
- Toolstrip has got one important property called "Items" which is collection of all the items (toolbar items) which are placed under.

➢ **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.

➢ **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.

➢ **DialogBox:**

➢ **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.

➢ **Open File Dialog Box and Save File Dialog Box**

- 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.

➢ **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.

CH : 6 USER CONTROLS

➢ **Objectives**

- Learn to create a Windows Form User Control
- Create enumerations for ease of use
- Add descriptions to your properties and methods using attributes
- Create error messages and verification that will be useful to the users of your Control

➢ **Overview of User Controls**

- Microsoft® .NET controls are all inherited from a common base class called **UserControl**.

- This class has all the basic functionality for a graphical control that will be used on a Windows Form.
- All of the built-in .NET controls inherit from this same base class. You can inherit from this same base class as well, and create your own controls.
- When you inherit from the **UserControl** class, you will be automatically supplied with certain properties and events.

➢ **Overview of the Employee Number Control**

- The EmployeeNumber control that you create will allow a user to input a number in the format AA-NNN.
- You must input two letters, followed by a hyphen, followed by three numbers.
- Figure 1 shows how this control might look. The control is made up of a label with the text "Employee Number" and a text box that can be filled in with an employee number.

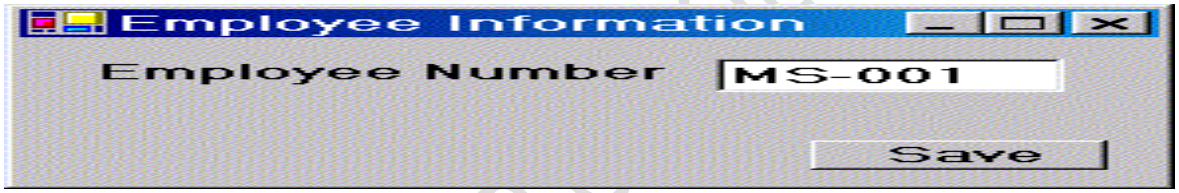


Figure 1. An employee number control on a Windows Form

➢ **Description of the Employee Number User Control**

- In addition to the built-in properties and events that are exposed automatically by the **UserControl** class, you add on properties, events, and methods to this user control.

➢ **Create the User Control**

➢ To create any User Control, you follow a series of steps. The major steps are as follows.

1. Create a new Windows Control Library project.
2. Draw the user interface by selecting other controls from the toolbox that define how you want your control to work.
3. Create any additional properties.
4. Create any events that you want.
5. Write any event procedures for constituent controls.

6. Create any methods that you want.
7. Add some attributes.
8. Build the control project.
9. Add a new Windows Application project so you can test your control.

- **Create the Employee Number Project**
  - Let's create the Employee Number User Control project.
  - Create a new project in Microsoft Visual Studio® .NET.
  - Choose the Windows Control Library template.
  - Set the Name of the project to **PKUserControls**.
  - Click **OK**.
  - You should now have a User Control design surface on which you can draw your user interface.
- **Create the User Interface**
  - To create this Employee Number User Control, you will need to add a label and text box to the User Control design surface.
  - Add the controls and set the properties as listed in Table 1. When you are finished, you should have something that looks like Figure 2.

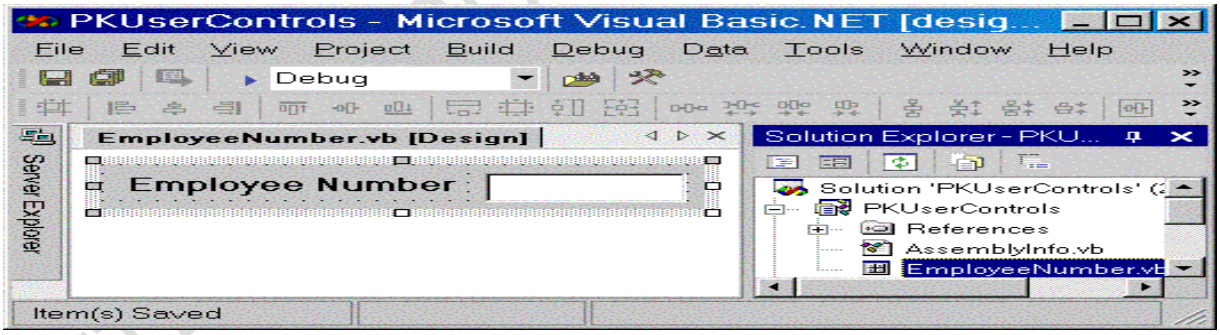


Figure 2. Create the Employee Number User Control using one Label and one Text Box control.

Table 1. Controls and properties to create and set in building the Employee User Control

Control Type	Property	Value
Label	Name	lblEmpNum
	Text	Employee Number
TextBox	Name	txtEmpNum
	Text	
	MaxLength	6
	CharacterCasing	Upper

- **Create Properties**
  - You will now build some additional properties that allow you to control the look of this User Control. You will create the following properties in this control:
    - **EmpID**
    - **LastName**
    - **FirstName**
    - **Salary**
    - **ConnectionString**
    - **Text**
  - You create the first five properties (**EmpID**, **LastName**, **FirstName**, **Salary**, and **ConnectionString**) using private member variables and property statements.
  - The **Text** property is simply a property statement wrapped around the **Text** property of the **txtEmpNum** text box control. The steps are as follows:
    - Create the following member variables just after the Public Class **EmployeeNumber** statement as shown below.
    - Add the **IMPORTS** statement, as this will be needed later in this document.

```
using System.ComponentModel
Public Class EmployeeNumber
{
        Inherits System.Windows.Forms.UserControl
  ' Employee Properties
        Private mintEmpId As Integer
                Private mstrLastName As String
                Private mstrFirstName As String
                Private mdecSalary As Decimal
    ' Connection String
        Private mstrConnectString As String
}
```

  - Create the property statements for each of these properties.

```
Integer Property EmpID()
{
        Get{ Return mintEmpId }
}
String Property LastName()
{
```

```
                    Get{ Return mstrLastName  }
        }
        Property FirstName()
        {
                    Get{ Return mstrFirstName}
        }
        Property Salary()
        {
                Get{  Return mdecSalary  }
        }
        Property ConnectionString()
        {
            Get{ Return mstrConnectString }
             Set(string value)
            {       mstrConnectString = Value}
        }
```

Create the **Text** property as a wrapper around the **txtEmpNum** text box control.

```
    Public Overrides Property Text()
    {
        Get{ Return txtEmpNum.Text }
        Set(string value){ txtEmpNum.Text = Value }
    }
```

- Notice that the **Text** property must use the **Overrides** attribute because the default **Text** property on the **UserControl** class is used to display a title on the User Control.

➢ **Create Events**

- If the user types in an incorrect employee number format, you will want to raise an event back to the client application so that you can inform the user of the problem with the format.
- To raise an event from a control, you must first declare the event within the User Control. Use the Event declaration followed by the name of the event.
- You will also need to pass in the standard parameters you find on all event procedures—*sender* and *e*.
- The *sender* parameter is declared as an object data type. The *e* parameter must be declared as a **System.EventArgs** type, or some derived class from the **System.EventArgs** class.

- In this case, you will want to create your own class that inherits from the **System.EventArgs** class because you want to add an error number property and a message property.
- These two properties allow you to fill in additional information about how the employee number is not in the correct format. Below is the declaration that you will need to write somewhere near the top of the **EmployeeNumber** User Control class.

```
' Create event
Public Event BadEmployeeNumber()
```

- The **EmpNumEventArgs** class must be created prior to creating this event declaration.
- Before you can create the **EmpNumEventArgs** class, you first need to create an enumeration of error numbers.
- The **ErrorNumber** property within the **EmpNumEventArgs** class will be defined as this enumeration type.

➢ **Create Methods**

- An additional check you might wish to perform on an employee number is to see whether it exists within a table in a database.
- The following method shows how you might validate the data within an Employees table, and once you find a valid record, populate the additional properties on this control with other information about the employee.
- For example, you might fill in the properties from other columns in the table: EmployeeId (**iEmp_id**), LastName (**szLast_nm**), FirstName (**sFirst_nm**), and Salary (**cSalary_amt**).

```
Public Function EmployeeIsValid()
{
    Oledbcommand ocmd=new oledbcommand();
    Oledbdatareader oDR =new OleDbDataReader();
    String strSQL;
    strSQL = "SELECT iEmp_id, sEmp_num, szLast_nm, "
    strSQL &= "sFirst_nm, cSalary_amt  "
    strSQL &= "FROM Employees "
    strSQL &= "WHERE sEmp_num = '" & _Trim(txtEmpNum.Text) & "'"
    ' Reset all properties
        mintEmpId = -1
```

```
            mstrLastName = ""
            mstrFirstName = ""
            mdecSalary = 0
                If oDR.Read()
        {
                mintEmpId = CType(oDR.Item("iEmp_id"), Integer)
                mstrLastName = CType(oDR.Item("szLast_nm"), String)
                mstrFirstName = _CType(oDR.Item("sFirst_nm"), String)
                mdecSalary = CType(oDR.Item("cSalary_amt"), Decimal)
        }
                oDR.Close()
        If mintEmpId = -1
                        Return False
        Else
                        Return True
    }
```

- Of course, this method assumes that you have a table called Employees in a database with the structure, as follows:

```
CREATE TABLE Employees
(
   iEmp_id int IDENTITY (1, 1) NOT NULL
   PRIMARY KEY NONCLUSTERED  ,
   sEmp_num char(6) NULL ,
   szLast_nm varchar(25) NULL ,
   sFirst_nm char(15) NULL ,
   cSalary_amt money NULL
)
```

Summary

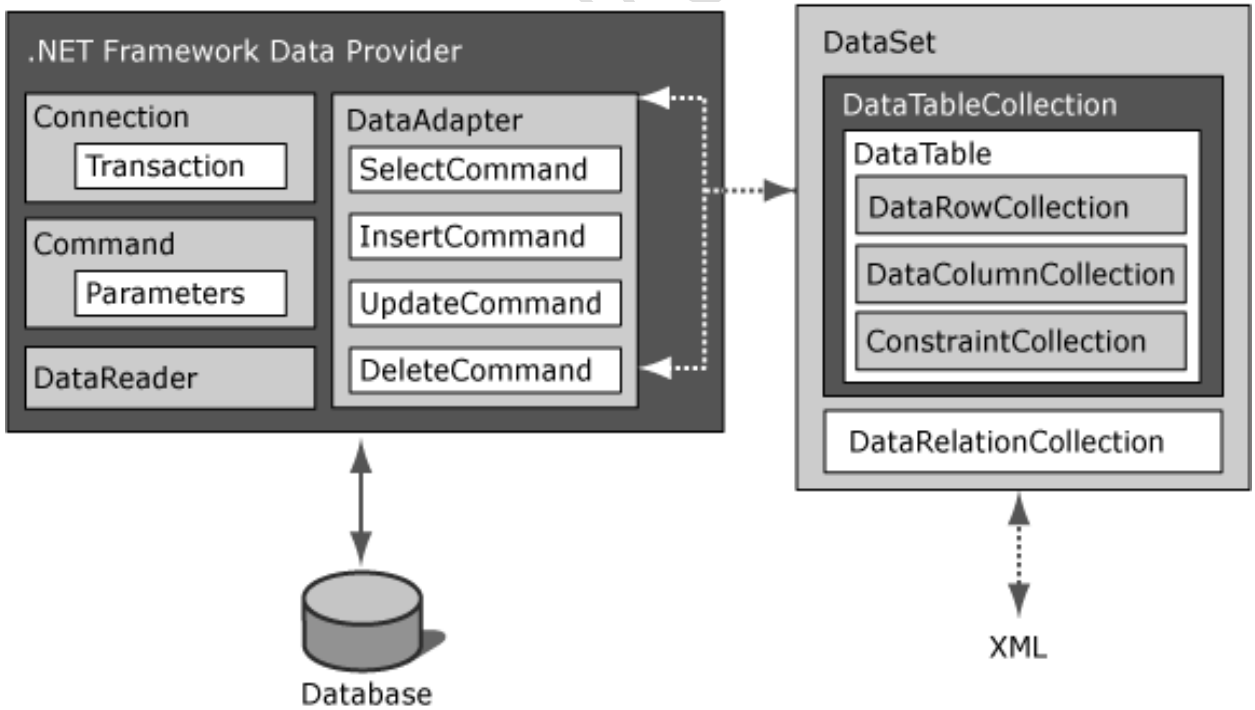
In this document, you learned how to create a User Control that you can use and reuse on any Windows form in any application. To reuse this control in any other project, simply customize the toolbox and point to the DLL that is created. Passing information back in your own custom **EventArgs** object is an excellent way to convey additional information to the user of this control. Adding attributes to your control will give users better information about how to use your control.

CH : 7 ADO .NET Programming

- **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 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 then 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 untill 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 artitecture 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.

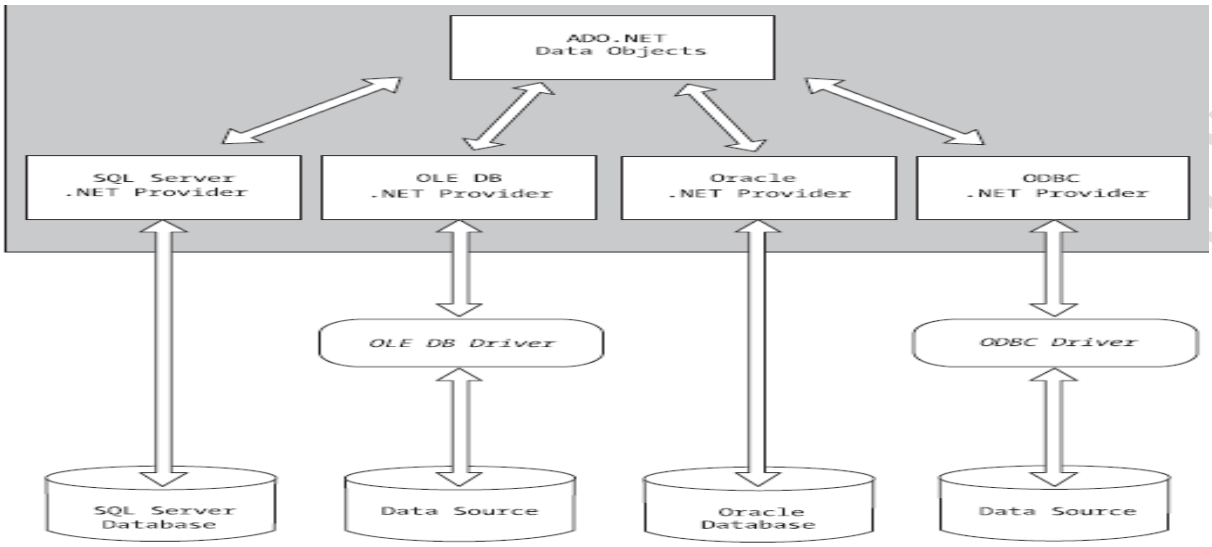
DataColumn : This is used incase you want to modify any column or you want to add a new column to DataSet or DataTable.

➢ **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 operations in local memory. Data is stored in local memory while doing operation.
This creates Heavy Traffic at database server side when no. of user increases.	As data is loaded into local memory, does not create Heavy Traffic at database server side even no. of user increases.
This utilizes memory of Database server as the data remains in database itself.	This utilizes memory of client as the data is loaded locally into client's machine.
Generally, transactional commands like Insert, Update, Delete, etc. are performed using Connected Architecture, as they do not require more time.	Generally, if you want to load data just for display purpose. Disconnected Architecture is best choice as you just want a copy of data into local memory which makes faster data access.
Command and DataReader objects are used for connected architecture.	DataAdapter, DataSet, DataTable, DataRow and DataColumn 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 the 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 roperties 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

➢ **Connection :**

- 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 incase you have given Transactional Commands like INSERT, UPDATE, DELETE, etc. This method returns integer value as answer. If value is > 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, it 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 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 the 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 ordinal 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 have 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 is filled into DataSet or DataTable. The Fill method is probably the DataAdapter method you will use most frequently. Simply stated, the Fill method adds data from your data source to a dataset. The Fill method accepts a variety of parameters including the DataSet object to fill, a string representing the alias for 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	Most important method of DataAdapter. This method is used to save the changes back from DataSet / DataTable to actual database table. Before using Update() method you need to use SqlCommandBuilder class so that it can generate related INSERT, UPDATE, DELETE, ALTER, etc. command automatically 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: 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 updating 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 then 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 within 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 tables 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, or the last time AcceptChanges was called.

Method	Description
Select	Allows you to select group of data from dataset. It has different methods to select the 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, and is a container object for zero or more DataColumn objects. The DataColumn objects define the DataTable column, including the column name, the data type, and any 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 column 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 of the 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 in the DataTable, as defined by the DataTable.Columns collection. Each DataRow has 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, or the last time AcceptChanges was called.
Select	Allows you to select group of data from table. It has different methods to select the 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.
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 new 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 projecting 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 size is 10 rows per page.
AutoGeneragteDeleteButton	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.
SelectedIndexChaged	Raised after index of selected link button is chaged in GridView control.

➢ **SUMMARY:**

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

1. <u>Connected Architecture :</u>

- Connected Architecture simply means, you are connected with the database throughout your operations.
- Because you are constantly connected to database. And so do all other users.

2. <u>DisConnected Architecture</u>

- 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

➢ **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.
- **SQL Server Provider :** This is special data provider which gives access to only SQL Server database.
- **Oracle Provider :** This is special data provider which gives access to only Oracle Server or client.
- **OLEDB Provider :** This provider gives access to any database that has OLEDB driver.
- **ODBC Provider :** This provider gives access to any database that has ODBC (Open DataBase Connectivity) driver.

➢ **Connection :**

- 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.
- ADO.NET provides us with connection classes like the SqlConnection class and OleDbConnection class.

➢ **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

➢ **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.

- 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.

- **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
- **DataSet :**
  - The DataSet object represents an in-memory group of database tables. You can have more then 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.
- **DataTable :**
  - The DataTable object represents an in-memory database table. You can add rows to a DataTable with a DataAdapter.
  - It contains a collection of columns represented by the DataColumnCollection, which defines the schema and rows of the 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.
  - 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.
- **DataColumn :**
  - DataColumn objects represent the columns, that is, the fields, in a data table.
  - DataRow is collection of DataColumns. One DataRow can have multiple columns within it because always a row contains multiple columns.
- **DataView :**

- When you bind to a DataTable, you actually use another object called the DataView.
- 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.

➢ **GridView :**

- 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.
- 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.

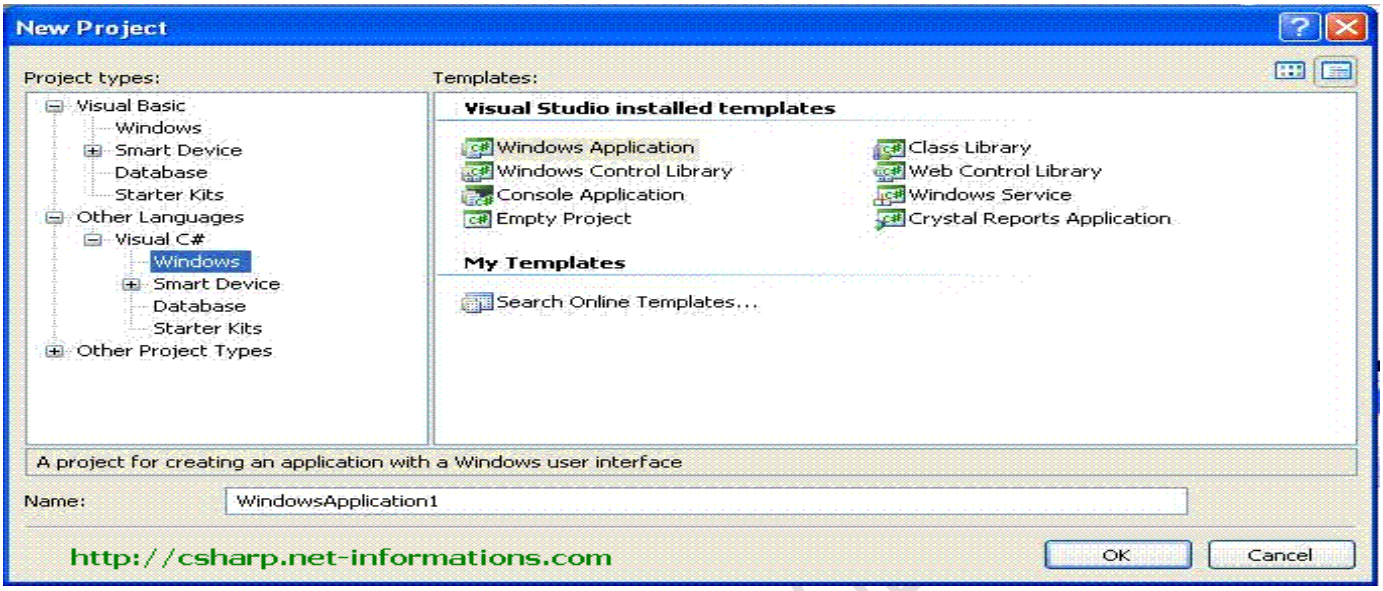
CH : 8 CRYSTAL REPORTS

- 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.

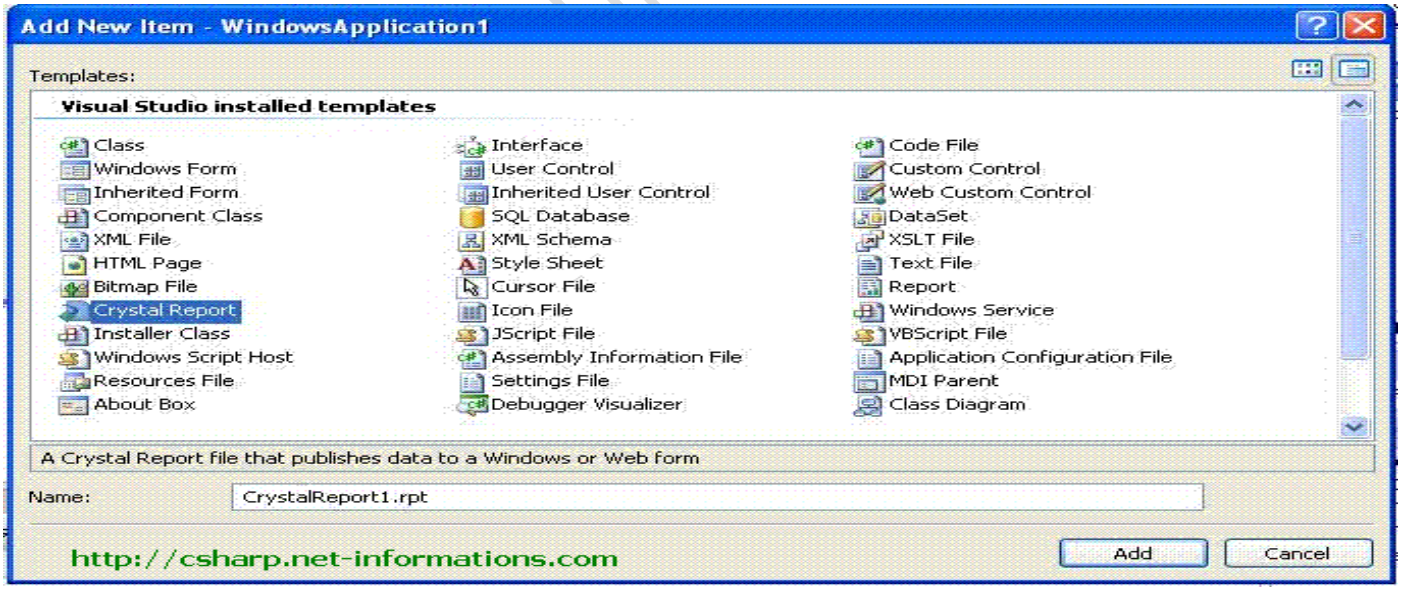
➢ **Steps to create Crystal Report**

- Here we are going to create a new Crystal Reports in C# from Product table in the above mentioned database crystalDB.

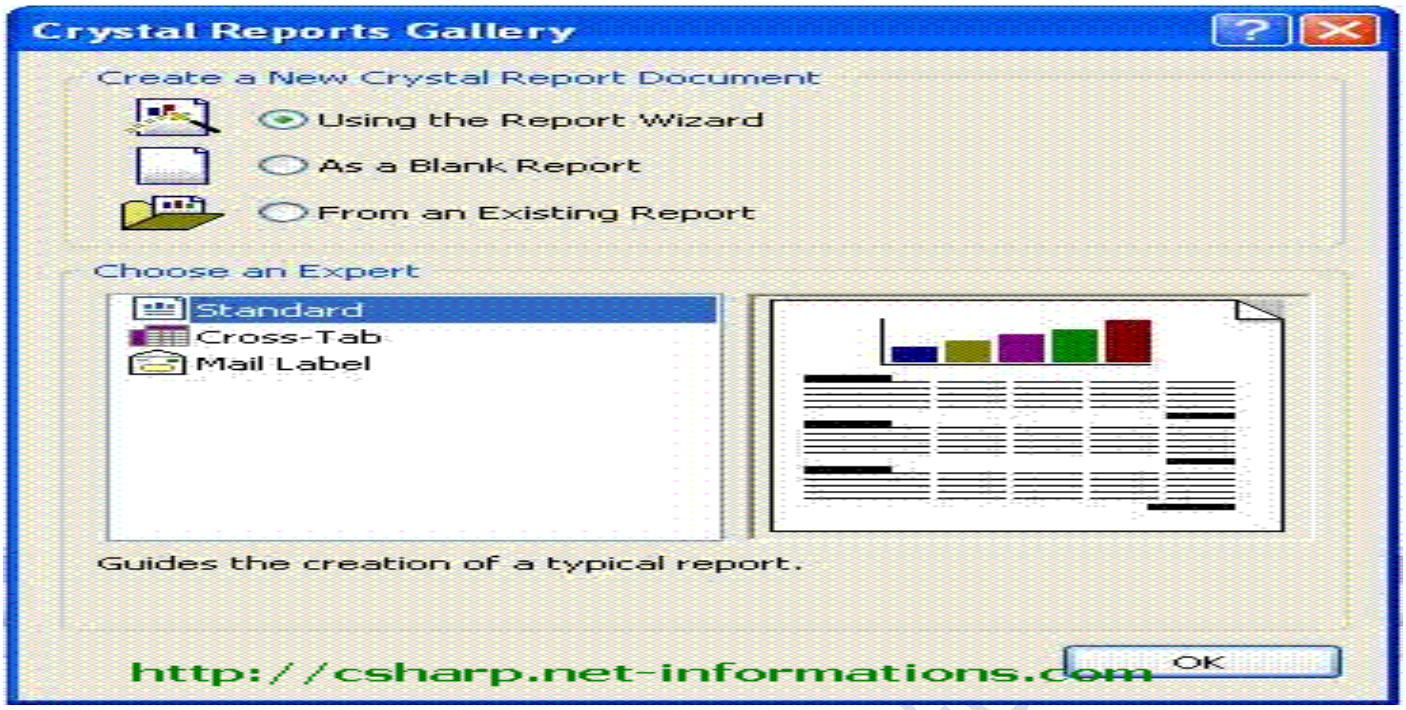
- The Product Table has three fields(Product_id, Product_name, Product_price) and we are showing the whole data from Product table to the C# - Crystal Reports project.
- Open Visual Studio .NET and select a new CSharp Windows project.



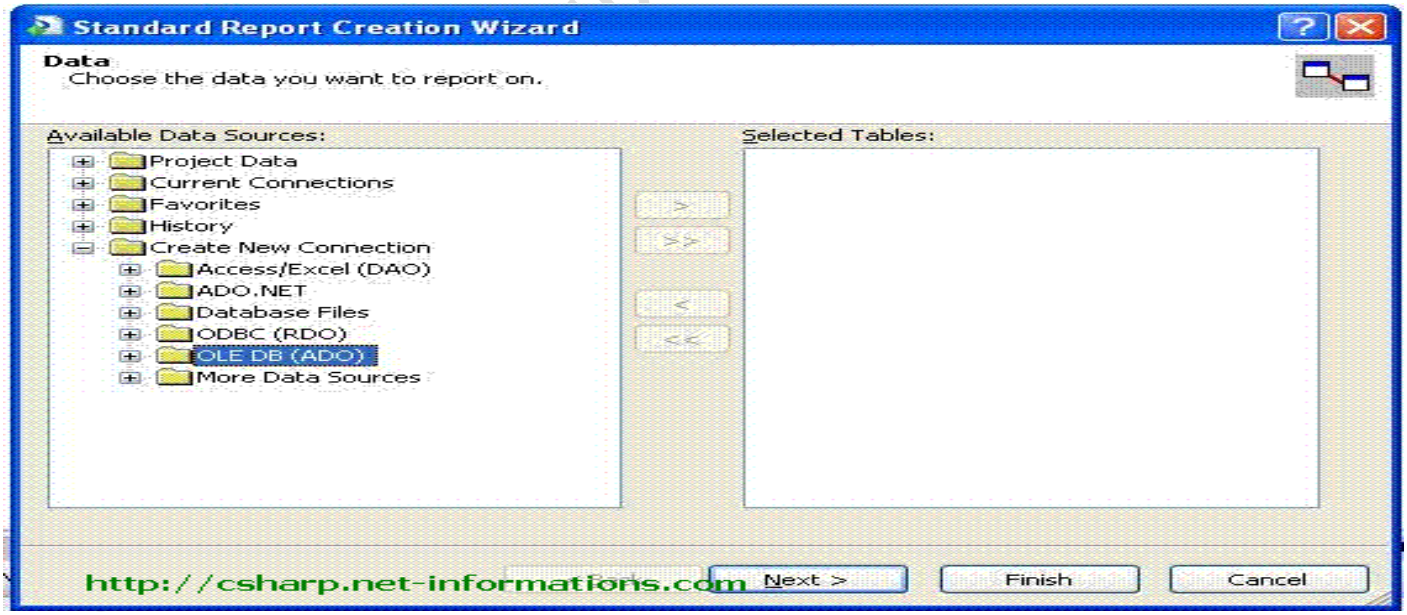
- Now you will get the default Form1.cs.
- From the main menu in Visual Studio C# project select **PROJECT-->Add New Item.** Then Add New Item dialogue will appear and **select Crystal Reports from the dialogue box.**



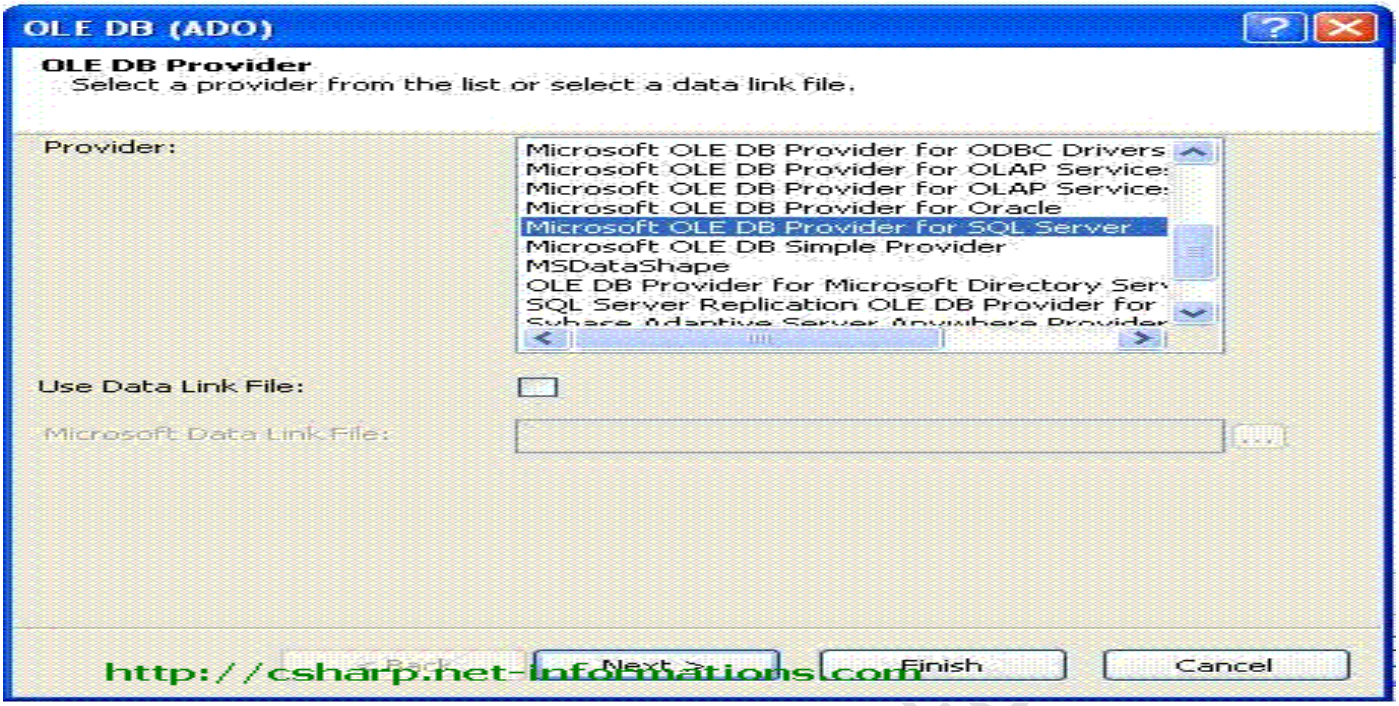
- Select **Report type from Crystal Reports gallery.**



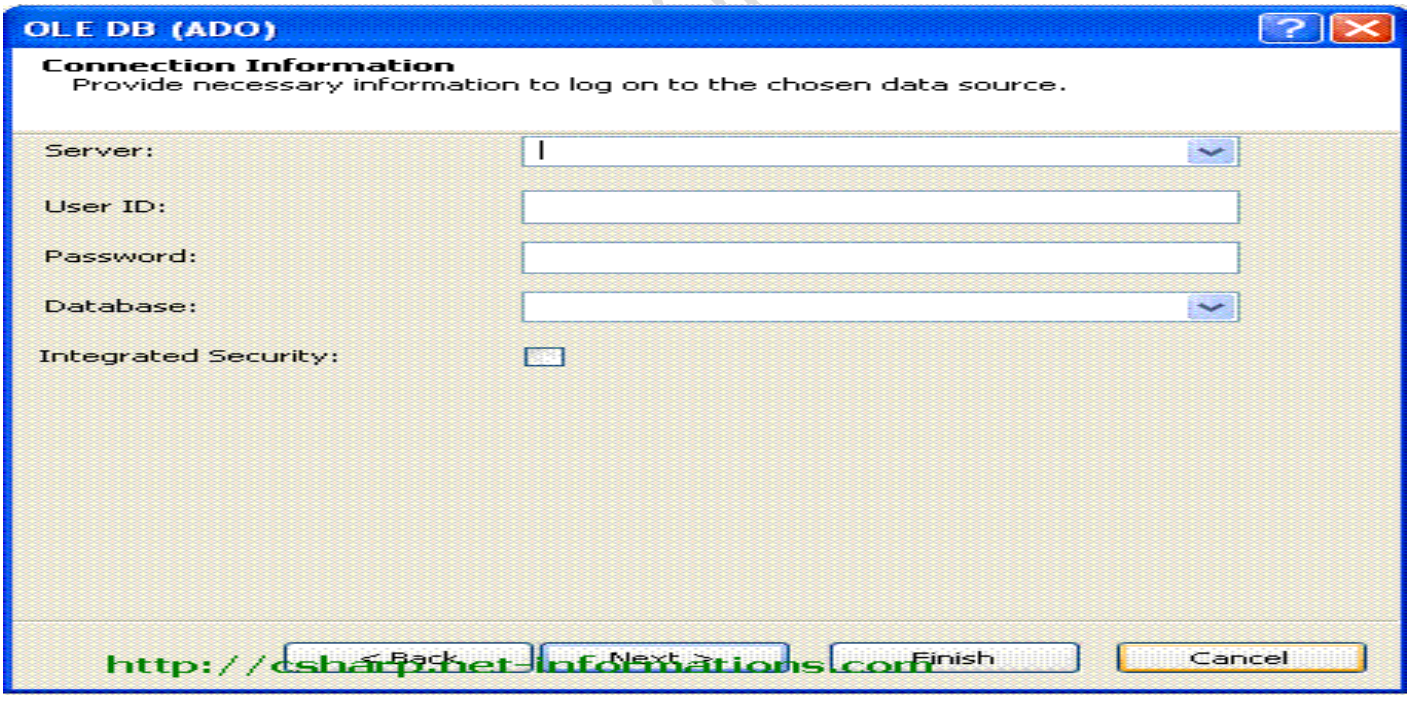
- Accept the **default settings and click OK**.
- Next step is to **select the appropriate connection to your database** (here crstaldb). Here we are going to select **OLEDB Connection** for SQL Server to connect Crystal Reports in C#.
- Select OLE DB (ADO) from Create New Connection .



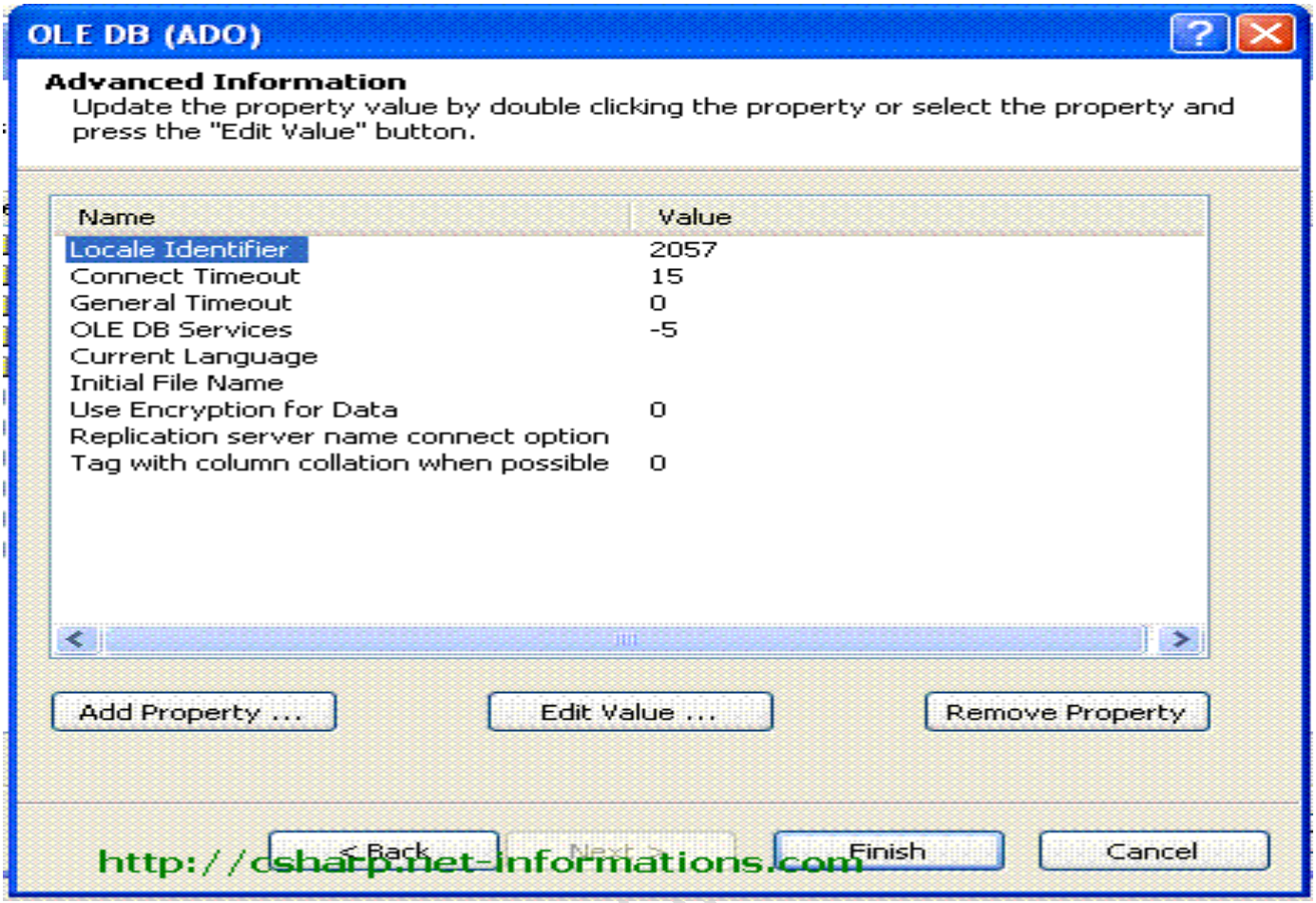
- Select Microsoft OLE DB Provider for SQL Server .



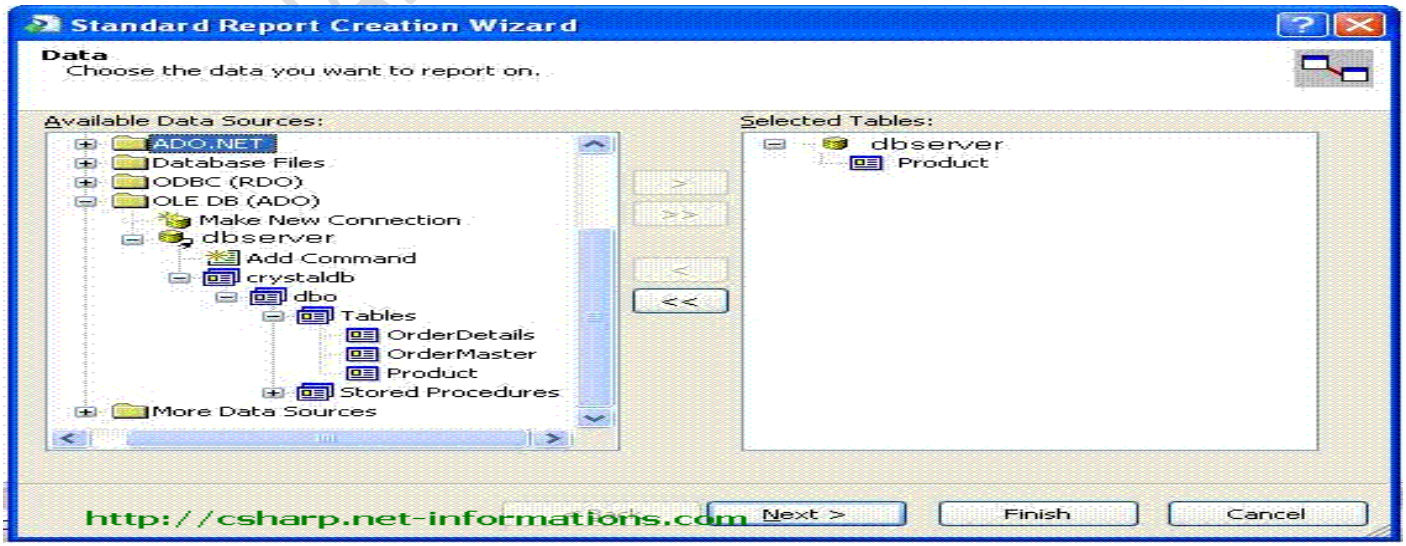
- The next screen is the **SQL Server authentication screen for connecting to the database** - crystalDB. Select your Sql Server name , enter userid , password and select your Database Name .



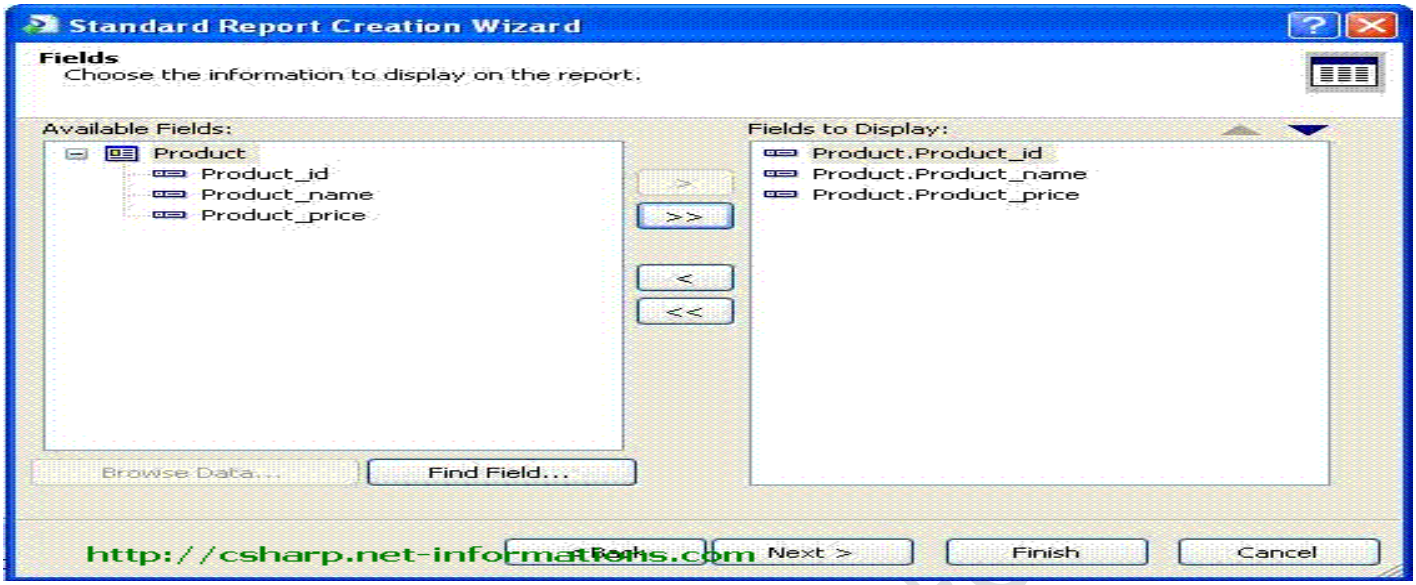
- Click next , Then the screen shows OLE DB Property values , leave it as it is , and then click finish button.



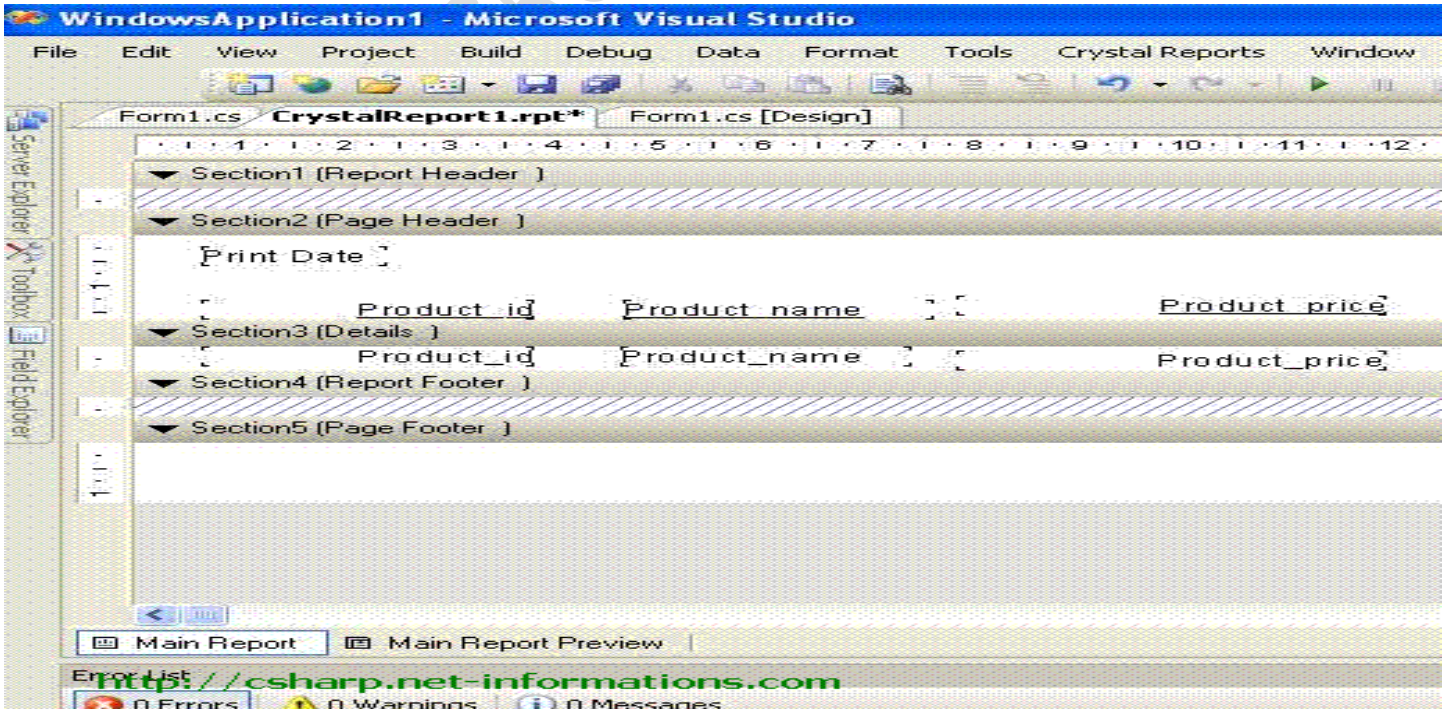
- After you click the finish button , the next window you will get your Server name under OLEDB Connection, from there selected database name (Crystaldb) and click the tables , then you can see all your tables from your database.
- From the tables list double click the Product table then you can see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



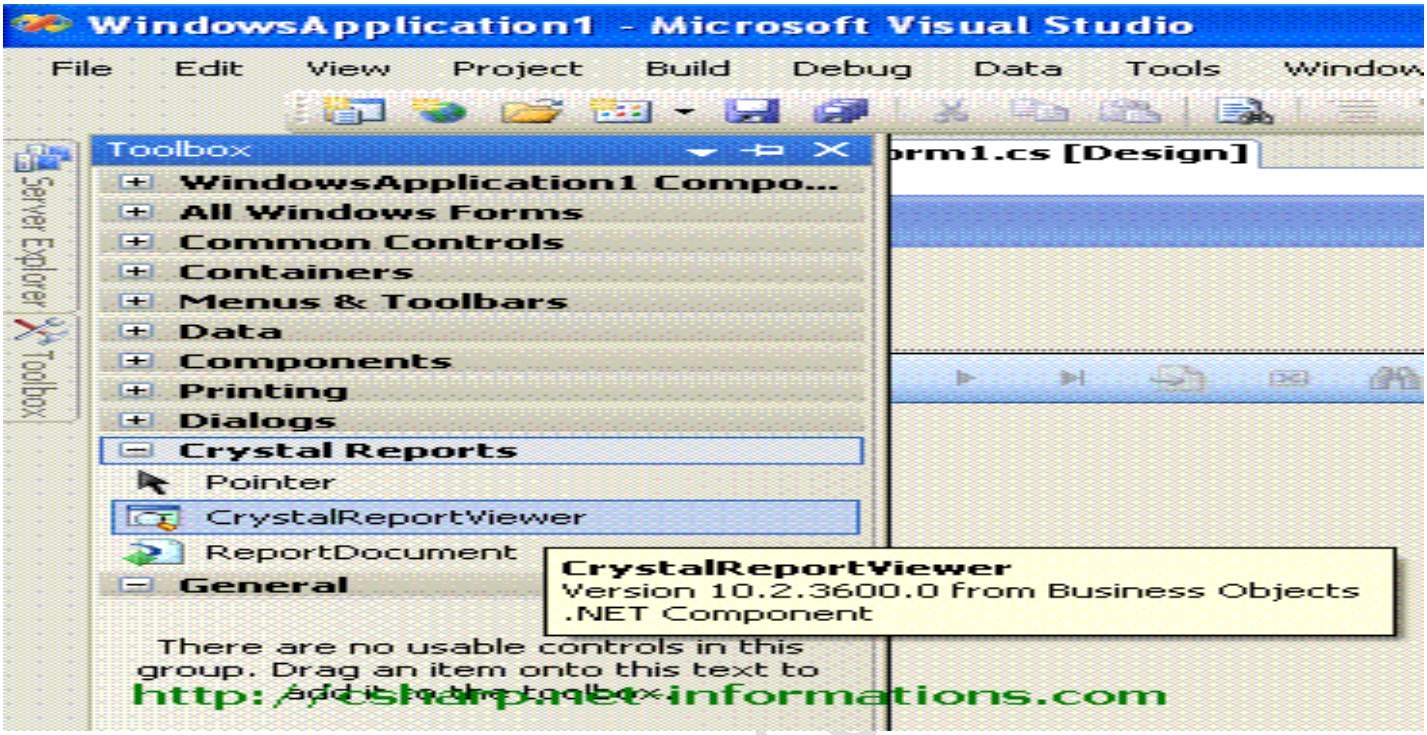
- Click Next Button
- Select all fields from Product table to the right side list .



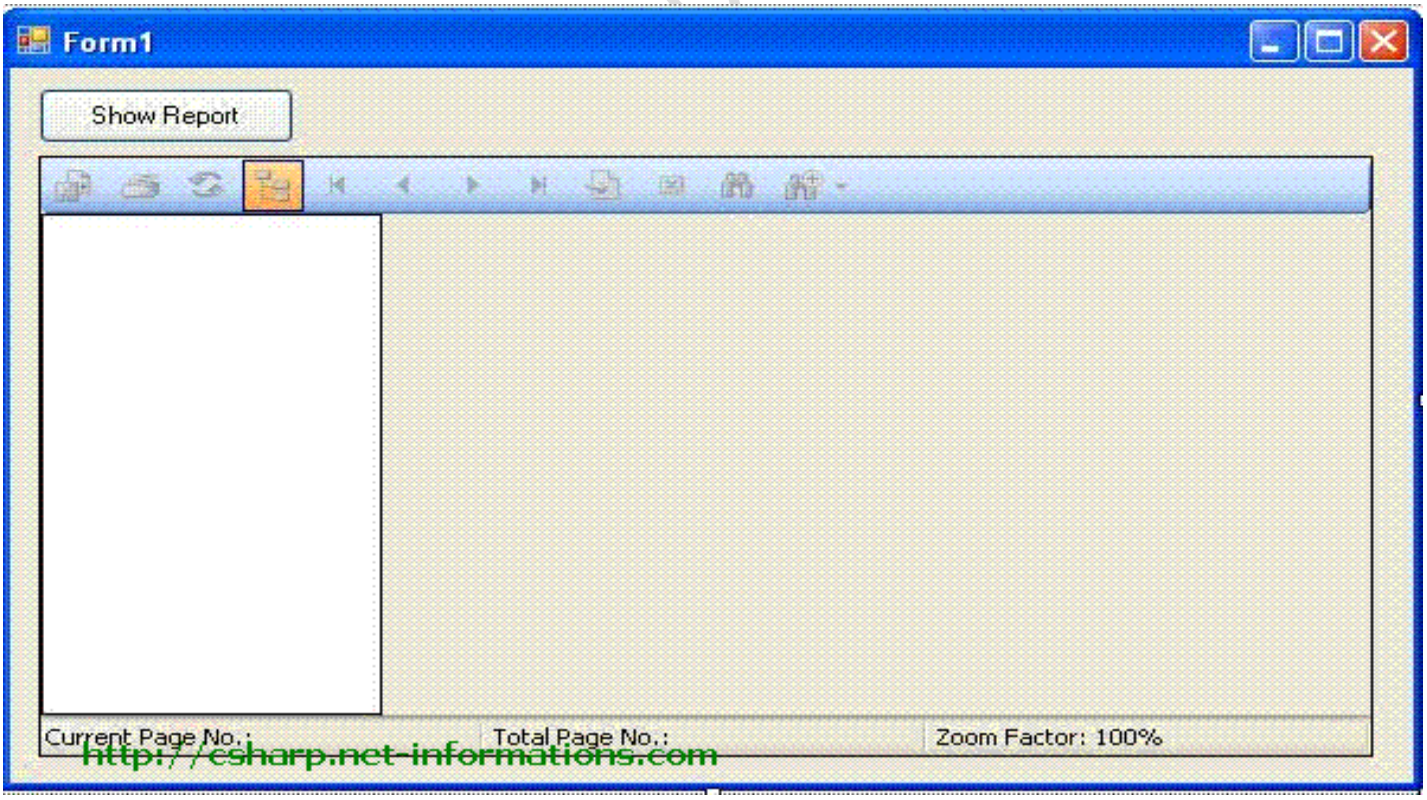
- Click Finish Button. Then you can see the Crystal Reports designer window in your C# project. In the Crystal Reports designer window you can see the selected fields from Product table. You can arrange the field Objects and design of the screen according your requirements. After that your screen is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



- You have to include **CrystalDecisions.CrystalReports.Engine** in your C# Source Code.

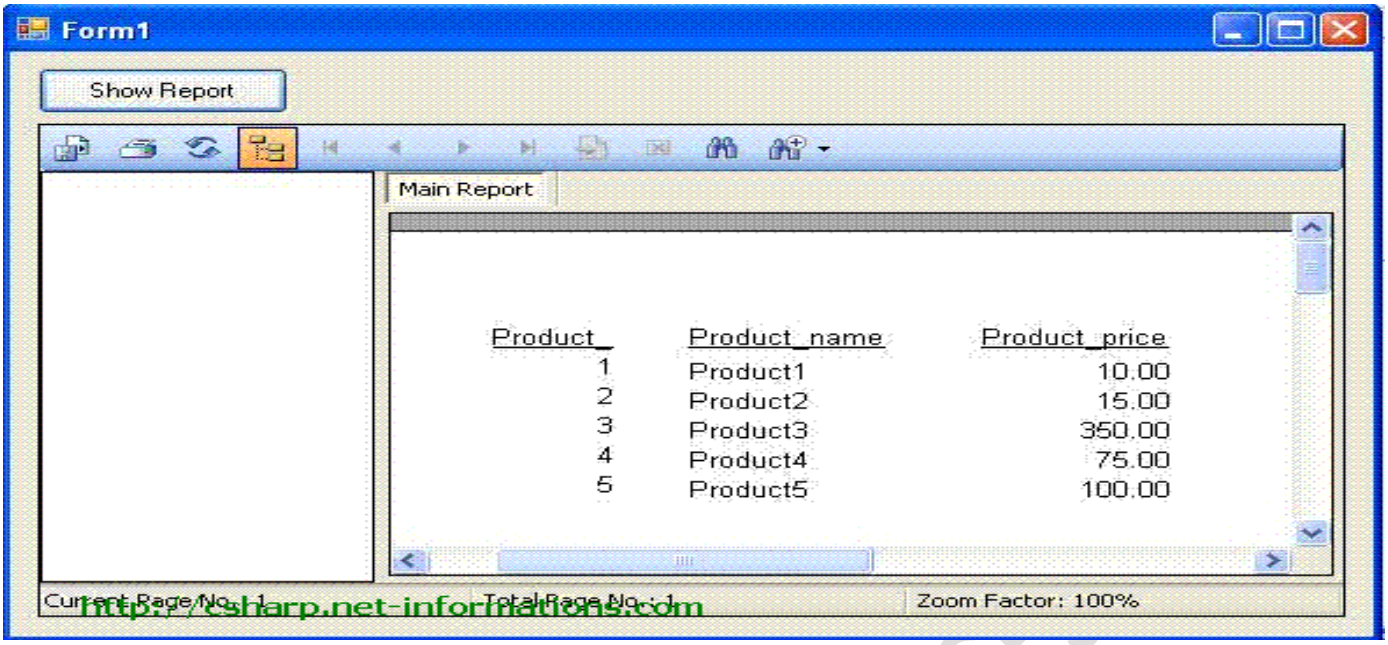
using CrystalDecisions.CrystalReports.Engine;

EXAMPLE:

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

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

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

- NOTES: cryRpt.Load(PUT CRYSTAL REPORT PATH HERE\\CrystalReport1.rpt"); The Crystal Reports file path in your C# project file location, there you can see CrystalReport1.rpt . So give the full path name of Crystal Reports file like c:\projects\crystalreports\CrystalReport1.rpt

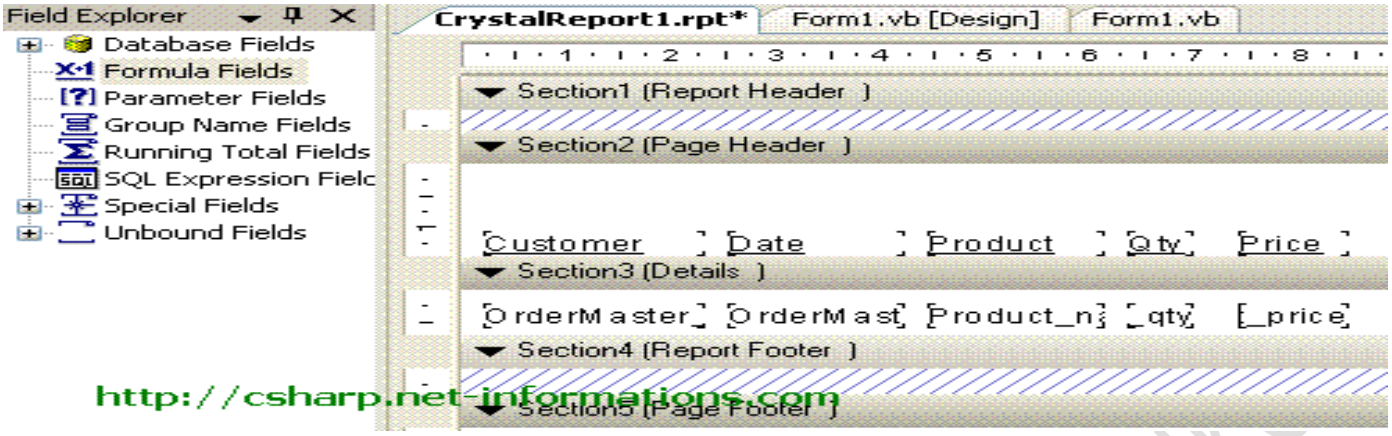
  When you run the source code you will get the report like the following picture.



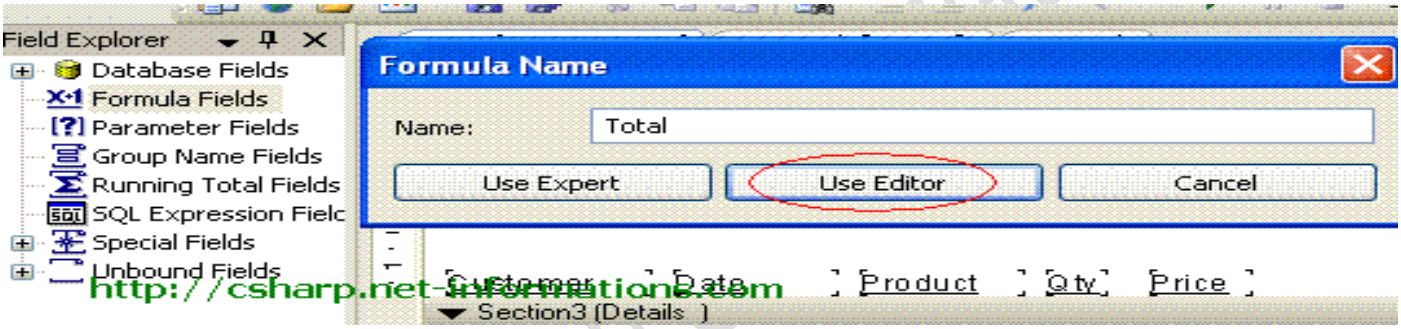
- When you click the button, the application will ask the username and password. Later in this tutorial you can find how to avoid asking username and password - C# Dynamic logon parameters in Crystal Reports.

➢ **Formula field**

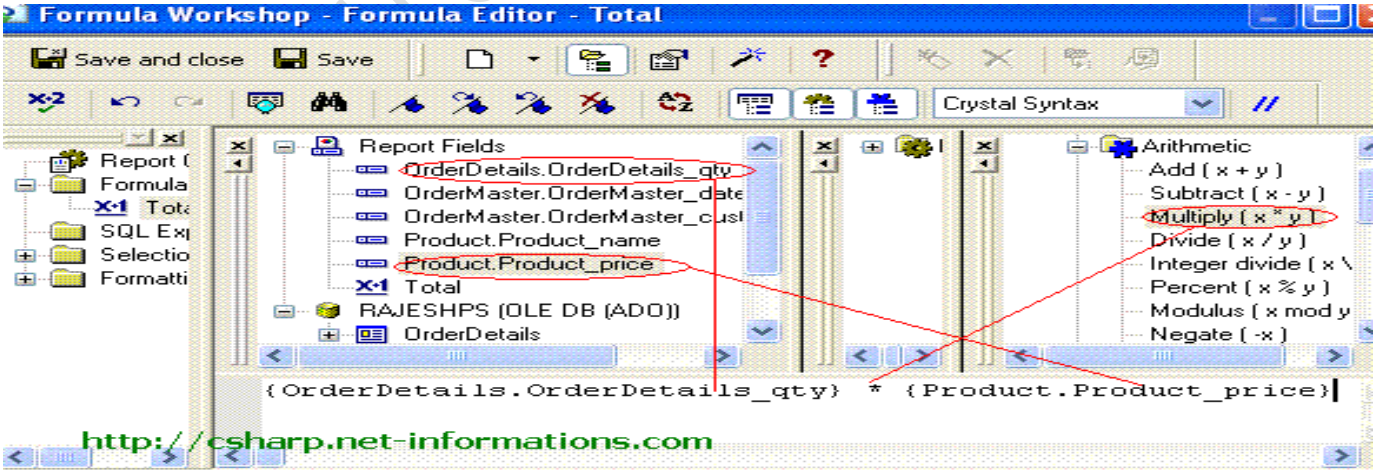
- If you have a Crystal Reports with Qty and Price fields and you need an additional field in your Crystal Reports for Total, that is **TOTAL = QTY X PRICE**
- In these types of situations you can use the Formula Field in Crystal Reports.
- Before starting this Create a new Crystal Reports with fields CustomerName , Order Date , Product Name and Product Price .
- In that report selecting only four fields , here we need one more field Product->Price and one formula field Total.
- After you create the above Crystal Reports, your CR designer screen is look like the following picture :



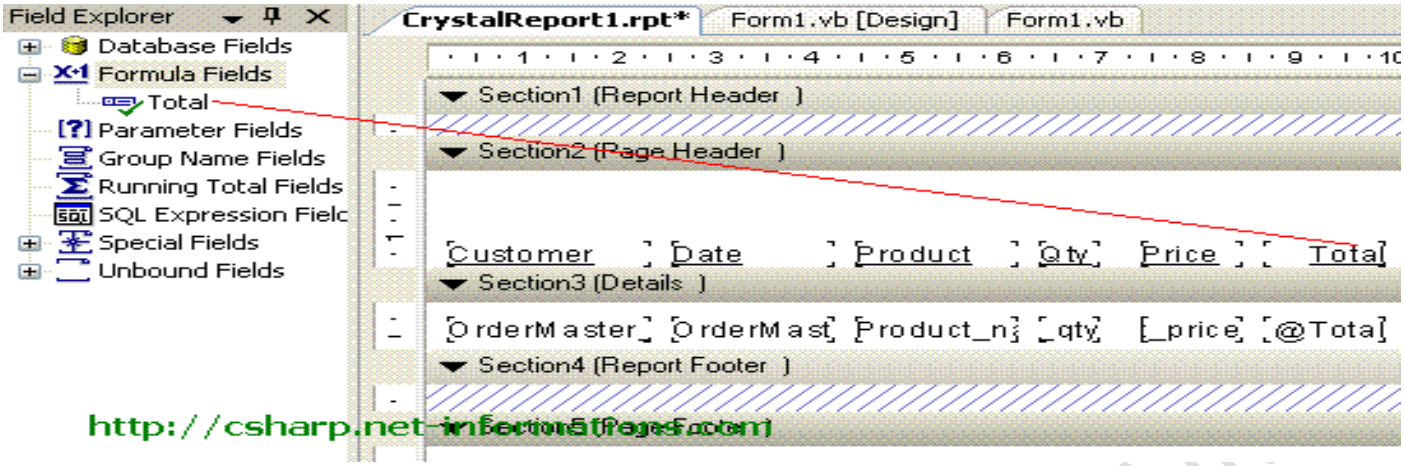
- Next step is to create a Formula Field for showing the result of **Qty X Price** .
- Right Click the Formula Field in the Field Explorer and click New. Then you will get an Input Message Box , type Total in textbox and click Use Editor.



- 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 multipy operator (*) and Product.Price . Double click each field for selection.



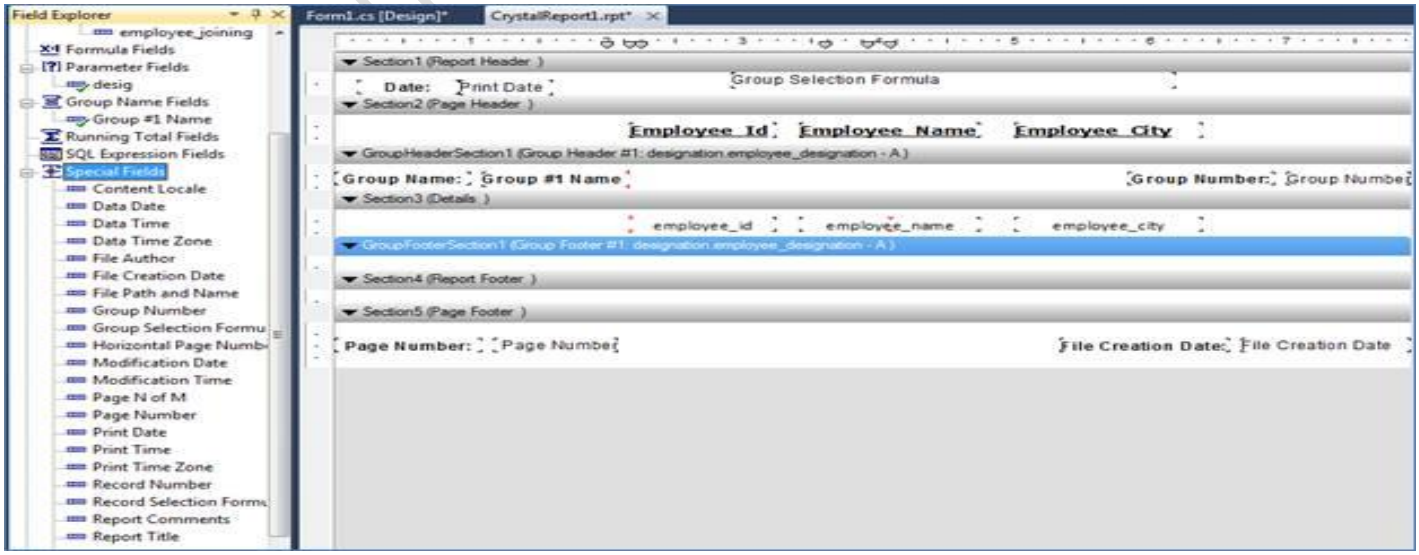
- Now you can see Total Under the Formula Field . Drag the field in to the Crystal Reports where you want to display Total.



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

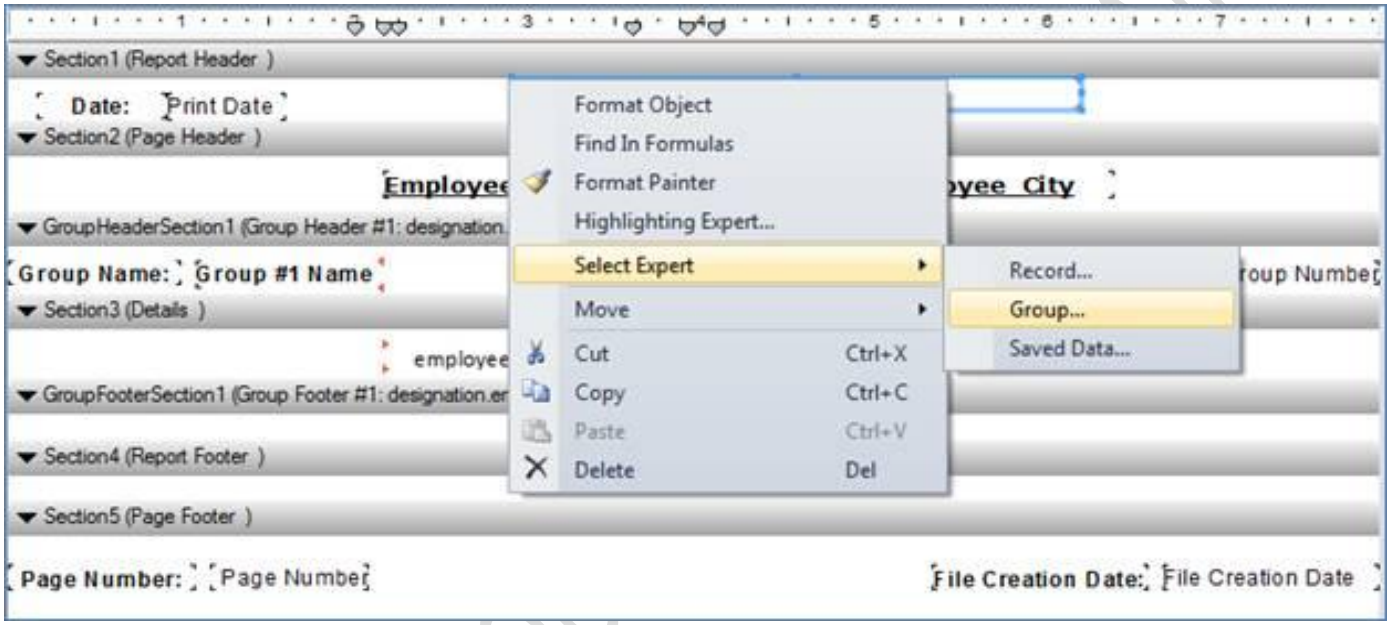
➢ SPECIAL FIELD

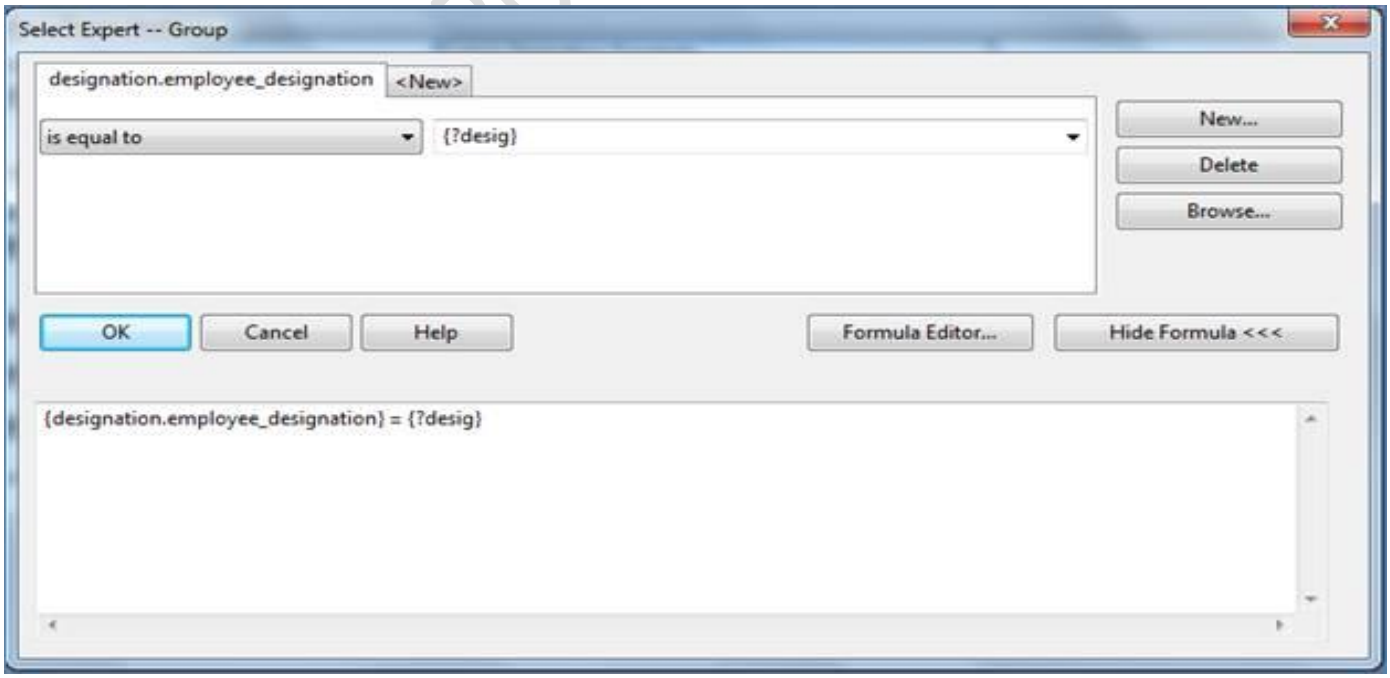
- It provides general information, such as Page Numbers, Print Date, and Report Comments, are located in the Special Fields list.
- In the above crystal report, I tried to display most of the special fields that are commonly used in our real life:
- There are various special fields which we can use for the following purpose:



- **Print Date:** Print Date Field is used to include the current date when the report prints. This field can be placed in any section of your report, depending on how often you want it to print. The date can be changed in the Set Print Date and Time Dialog Box. I have located a print date field in the page header of the crystal report in the above figure.

- **Group Selection Formula:** Group Selection Formula Field is used to insert a selected group into your report. (Use the Select Expert to create a group selection formula for your report.)





- **Record Selection Formula:** Record Selection Formula Field is used to insert a selected record into your report. (Use the Select Expert to create a record selection formula for your report.)

- **File Creation Date:** File Creation Date Field is used to include a field that displays the date when you created the report. I have located a File creation field in the report footer of the crystal report.

- **Record Number:** Record Number Field is used to number each record printed in the Details section of your report.

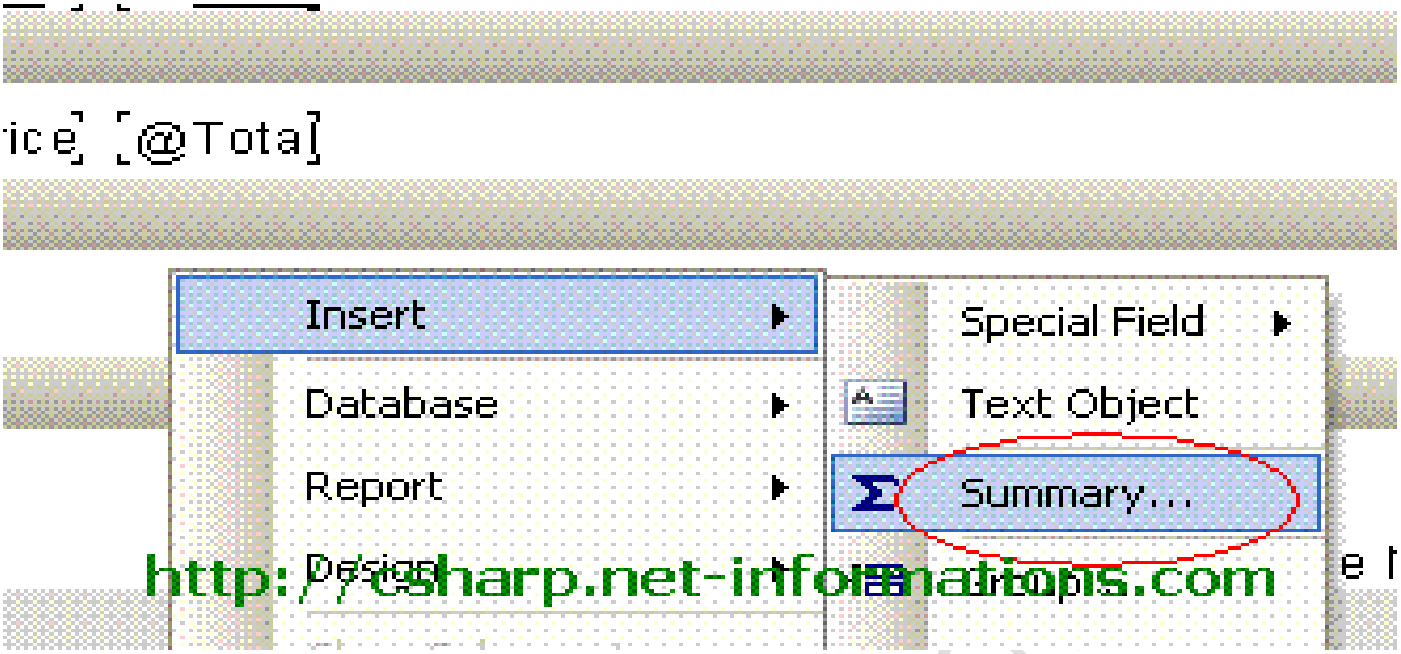
- **Page Number:** Page Number Field is used to insert a field that prints the current page number. These fields are most often placed in the Page Header or Page Footer sections. I have located a Page Number field in the report footer of the crystal report.

- **Group Number:** Group Number Field is used to number each group in your report. You can place this field in either the Group Header or Group Footer section of your report. I have located a Group Number field in the group header section of the crystal report.

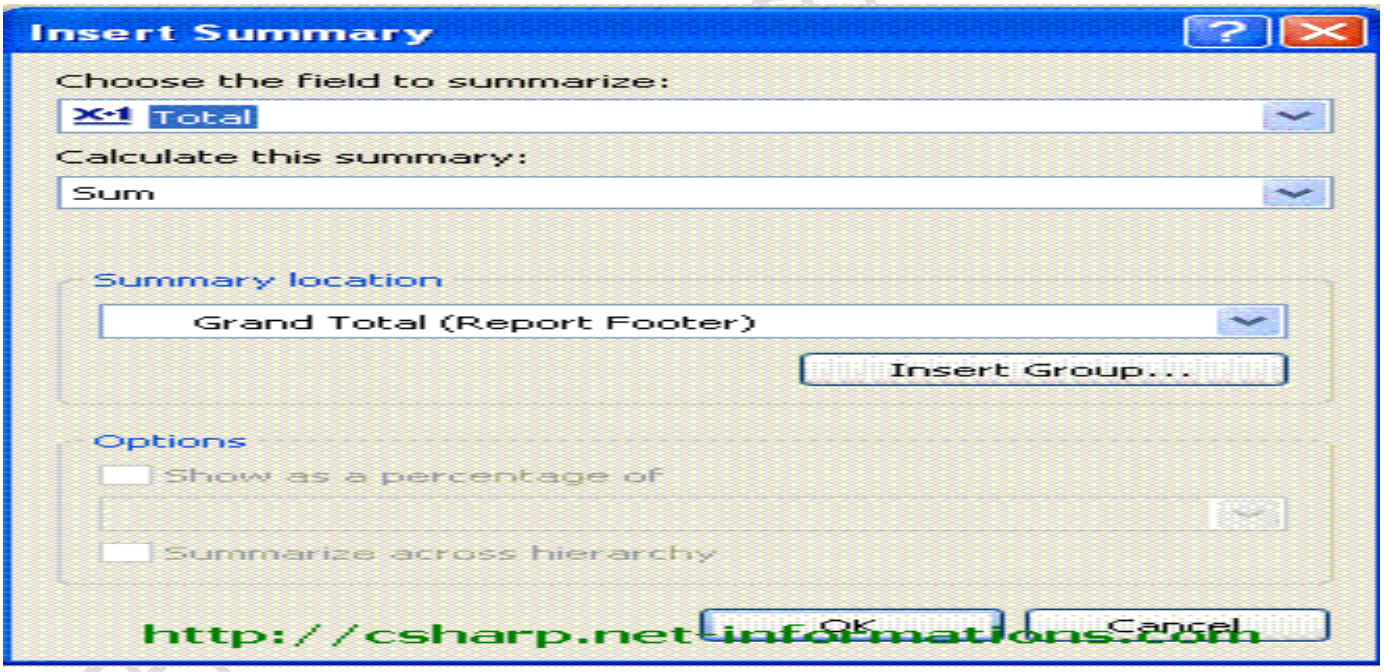
➢ **SUMMARY FIELD**

- Here in this section we are calculating the grand total of the Formula Field - Total . The Total field is a Formula field, the result of qty X price .

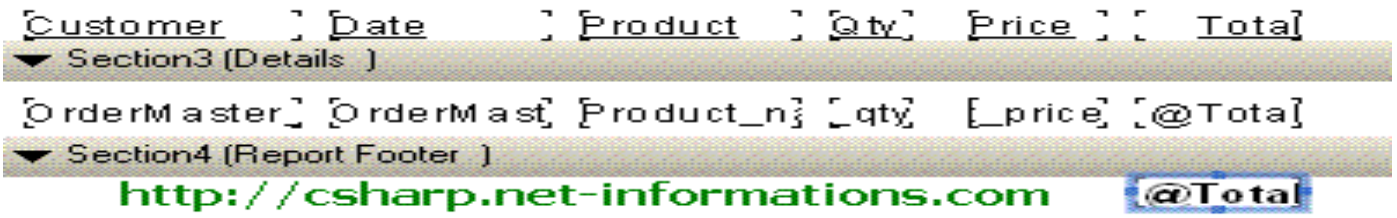
- In the Crystal Reports designer view window, right click on the Report Footer , just below the Total field and select Insert -> Summary .



- 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.



- Now the designing part is over and the next step is to call the Crystal Reports in C# and view it in Crystal Reports Viewer control .

- Select the default form (Form1.cs) you created in C# and drag a button and a CrystalReportViewer control to your form .

➢ **SUMMARY:**

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

➢ **Below are the fields which are used in crystal reports**

- ➢ **Formula field**
- ➢ **SPECIAL FIELD**
  - It provides general information, such as Page Numbers, Print Date, and Report Comments, are located in the Special Fields list.
  - There are various special fields which we can use for the following purpose:
    - **Print Date:**
    - **Group Selection Formula:**
    - **Record Selection Formula:**
    - **File Creation Date:**
    - **Record Number:**
    - **Page Number:**
    - **Group Number:**
  - **SUMMARY FIELD**

CH : 9 SETUP PROJECT

➢ **Creating a Setup and Deployment Project**

- Step 1: Create a Sample .Net Project. I have named this project as "TestBlogProject".

Figure 1 Sample Project

- **Create a Setup Project**
  - Open the TestBlogProject in the VS IDE.

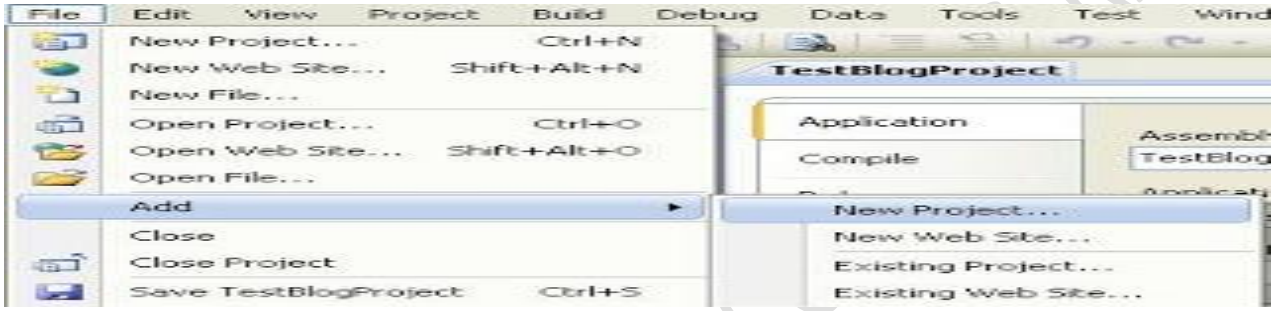


Figure 2: Add New Project

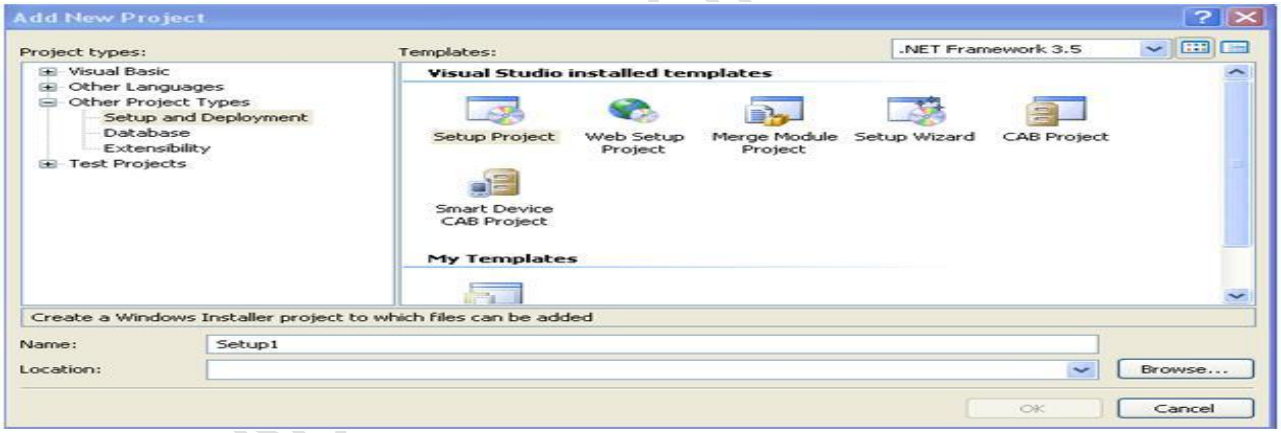


Figure 3: Setup Project Dialog

- There are five different templates that are available when you select the project type 'Setup and Deployment Projects'.

1. **Setup Project:** Creates a Windows Installer project to which files can be added. This project type is the most generic of the five.

2. **Web Setup Project:** Creates a Windows Installer web project to which files can be added.
3. **Merge Module Project:** Creates a Windows Installer Merge Module project to which files can be added. This project type is most commonly used for using third party products.
4. **Setup Wizard:** Creates a Windows Installer project with the aid of a wizard.
5. **Cab Project:** Creates a Cab project to which files can be added.

- For the purpose of this article, I will be using the template Setup Project.
- Provide a name for the project as well choose the location where this project will be established. (See Figure 3) Once you have done this click 'OK' and you should now see a screen like the following.

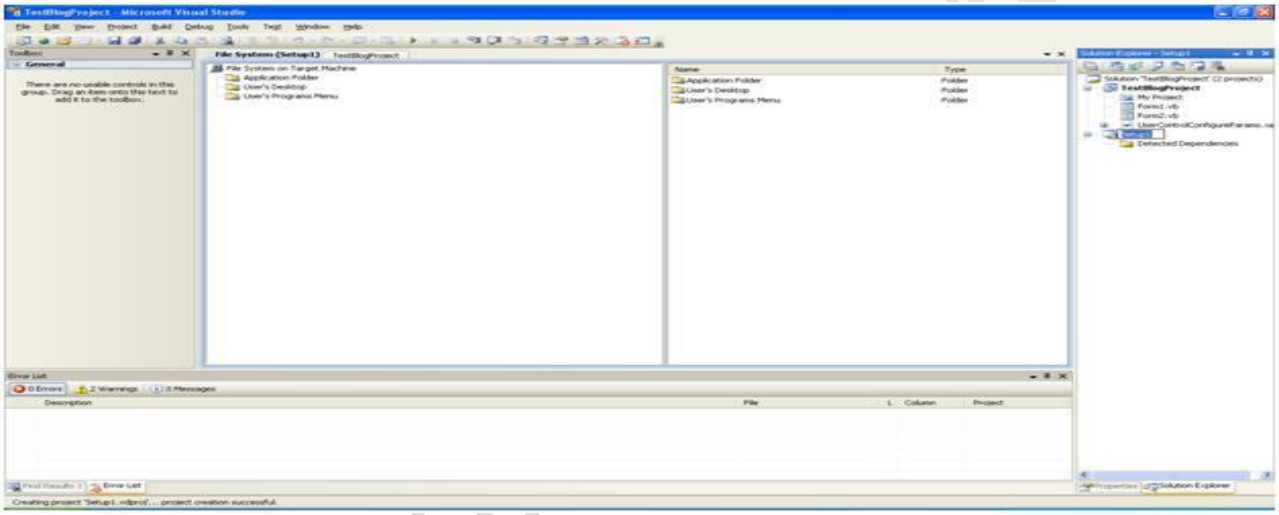


Figure 4: Default Setup Project File View

- Now that we have an empty Setup Project established. Let us discuss about the various editors that are available.

1. Right Click on the Setup Project and Click View.

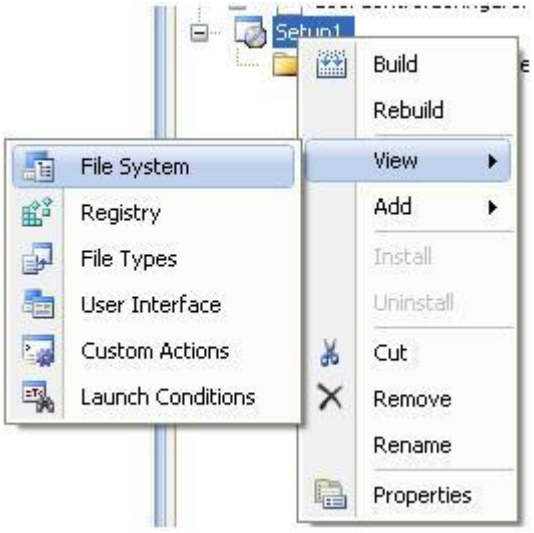


Figure 5: File System Editor

2. Click on the File System

- **File System Editor**
  - The File System (Figure 6) provides you the essential tools to work with the application folder, user's desktop, and the user's programs menu.

Figure 6: File System Editor

- The first thing we want to do here is to include the relevant files and folders we wish to install on the user's machine.
- This is accomplished by right clicking the Application Folder and selecting 'Add'. The following screenshot is an example of something similar you may see.

Figure 7: Add Project Output

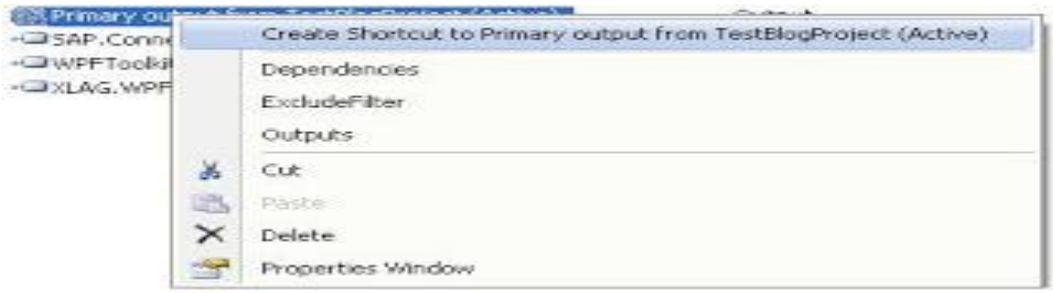
- Now to add project output files to this folder, Click Project Output.

- Click OK.
- Merely add one or more of the files that is vital for your application by clicking on File…. Now that you have you files in place.

Figure 8: Project Output

- We could take additional steps such as manipulating the user's desktop with items such as a shortcut, as well you can add a program group to the user's program group.

Create Shortcut



- Name the shortcut as you want.
- I have given the name as TestBlog.
- Drag and drop to User's Desktop Folder

Figure 9: Desktop Shortcut to application

- The next item you find of immense importance is the User Interface Editor.
- Inside of this editor you may eliminate or include dialogs that are needed for this install project.
- The following screenshot is the default User Interface that is created when I initially created this setup project (See Figure 10).
- The only variation is that I have added a Customer Information dialog.

➢ **Figure 10: User Interface Editor**

- To include the Customer Information dialog that I spoke of follow the following steps.

1. Under Install right click on the 'Start' node

2. Choose 'Add Dialog'

Figure 11: Customer Information

3. Select the Customer Information Dialog and click 'OK'

- As you can see there are is an ample variety of dialogs you can choose from.

➢ **Prerequisites Installation in Windows Installer**

- Most applications have prerequisites: Components such as the .NET Framework runtime must be available on a target computer in order for the application to run.
- The deployment tools in Visual Studio include the capability to automatically detect the existence of components during installation and install a predetermined set of prerequisites — a process known as ***bootstrapping*.**This will be achieved as follows:

To choose which prerequisites to install

1. In **Solution Explorer**, select the deployment project and Right Click
2. Click **Properties**.
3. In the **Property Pages** dialog box, expand the **Configuration Properties** node, and then select the **Build** property page.
4. Click the **Prerequisites** button.
5. In the **Prerequisites** dialog box, make sure that the **Create setup program to install prerequisite components** box is checked.

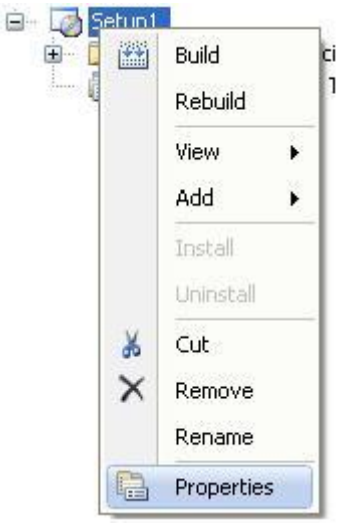
6. In the **Choose which prerequisites to install** list, check the prerequisites that you wish to install, and then click **OK**.

To specify the download location for prerequisites

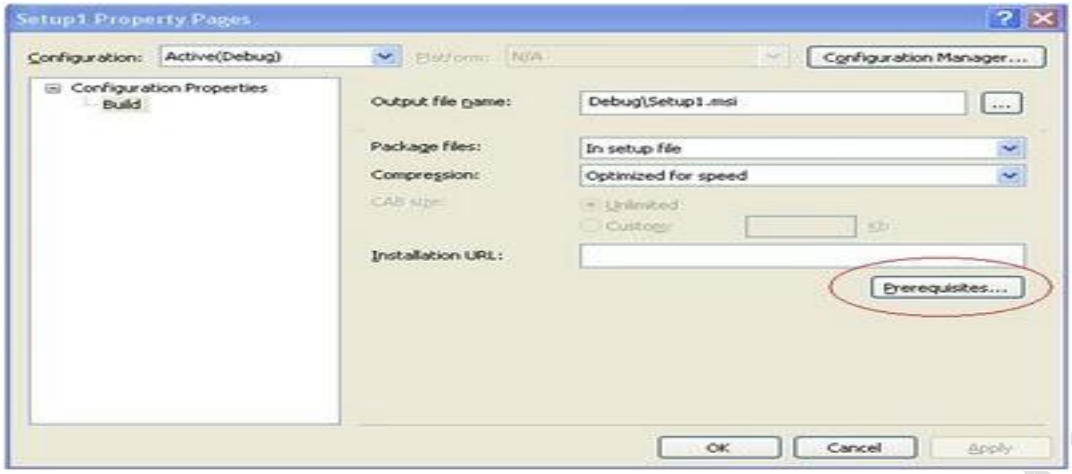
- In **Solution Explorer**, select the deployment project, right click.
- Click **Properties**.
- In the **Property Pages** dialog box, expand the **Configuration Properties** node, and then select the **Build** property page.
- Click the **Prerequisites** button.

  In the **Prerequisites** dialog box, choose a location:

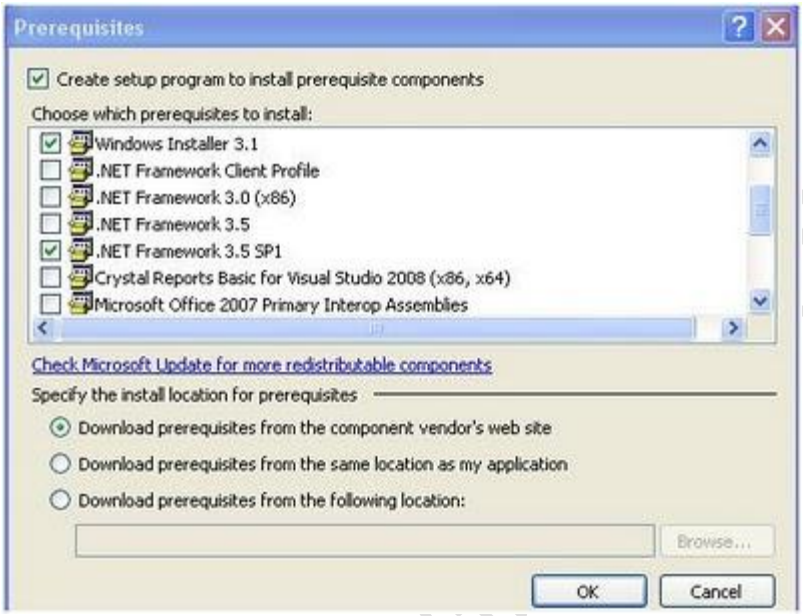
  - If you want to deploy the installers for the prerequisites to a vendor, click **Download prerequisites from the component vendor's web site**.
  - If you want to deploy the installers for the prerequisites to the same location as your application installer, click **Download prerequisites from the same location as my application**.



  - If you want to deploy the installers for the prerequisites to a different location, click **Download prerequisites from the following location** and enter a local path, URL, or file-share location.



- Click **OK** to continue.



I have covered adding a new setup project, about File System Editor, adding installer prerequisites to Windows Installer.