INTRODUCTION TO VISUAL STUDIO 2008:

Microsoft Visual Studio is an Integrated Development Environment (IDE) from Microsoft. It can be used to develop console and graphical user interface applications along with Windows Forms applications, web sites, web applications, and web services in both native code together with managed code for all platforms supported by Microsoft Windows, Windows Mobile, Windows CE, .NET Framework, .NET Compact Framework and Microsoft Silverlight.
Visual Studio includes a code editor supporting IntelliSense as well as code refactoring. The integrated debugger works both as a source-level debugger and a machine-level debugger. Other built-in tools include a forms designer for building GUI applications, web designer, class designer, and database schema designer. It allows plug-ins to be added that enhance the functionality at almost every level - including adding support for source control systems (like Subversion and Visual SourceSafe) to adding new toolsets like editors and visual designers for domain-specific languages or toolsets for other aspects of the software development lifecycle.

FRAMEWORK:

A programming infrastructure created by Microsoft for building, deploying, and running applications and services that use .NET technologies, such as desktop applications and Web services. The .NET Framework contains three major parts:

- the Common Language Runtime
- the Framework Class Library
- ASP.NET.

When you develop any C# application, you will initially have to decide on which .net framework are you going to use in your code. First version of .net framework was released in the year 2002. From then on, new features and enhancements were released in the subsequent versions. So far, the following versions of .net framework have been released:

- .NET Framework Version 1.0 in the year 2002
- .NET Framework Version 1.1 in the year 2003
- .NET Framework Version 2.0 in the year 2005
- .NET Framework Version 3.0 in the year 2006
- .NET Framework Version 3.5 in the year 2007
- .NET Framework Version 4.0 in the year 2010

The recent release is version 4.0. But this is a beta version and it is still not rapidly used in the industry. Hottest version of .NET framework acceptable and used widely is the 3.5 Version. This article will highlight on the new features introduced in .NET 3.5 and it will also give you information about the enhancements done in existing features as per version 3.5.

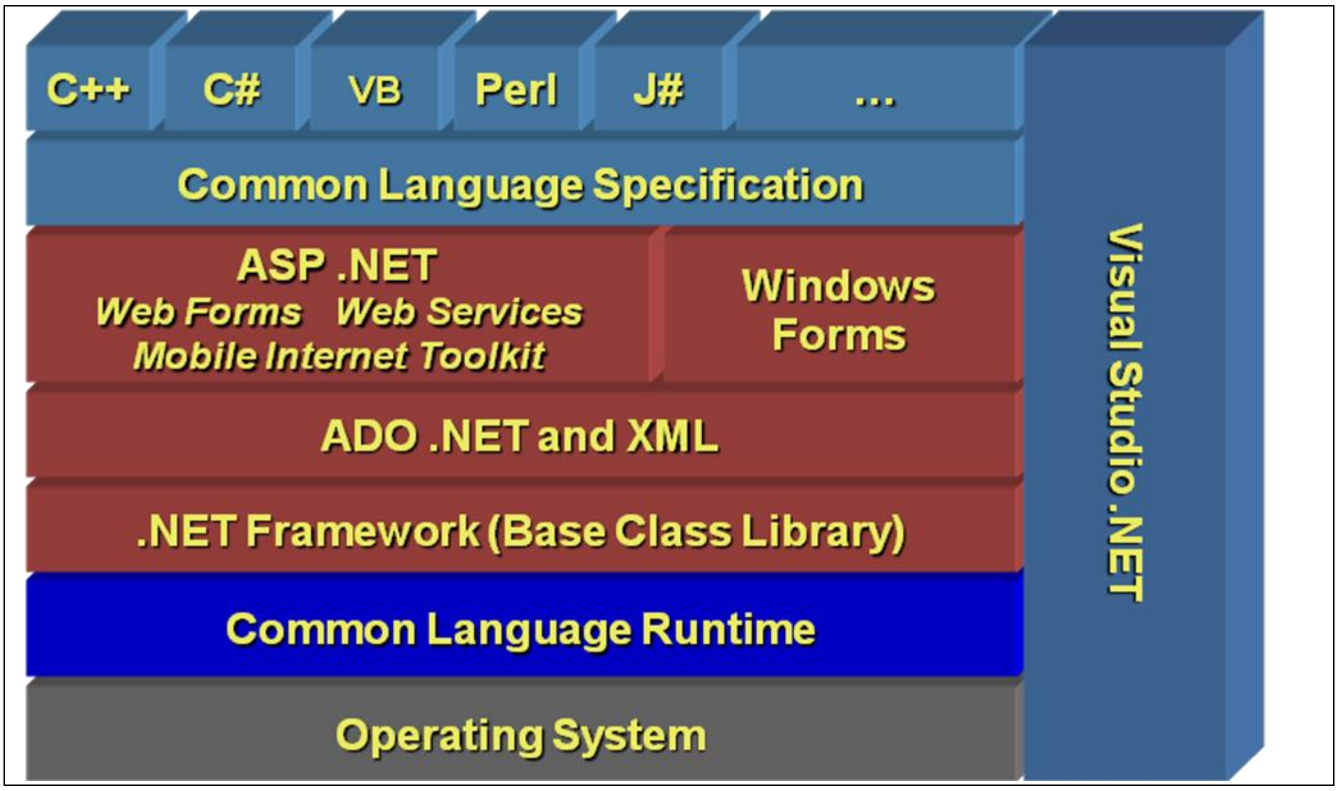
NET is a platform that provides a standardized set of services.

- ❖ It's just like Windows, except distributed over the Internet.
- ❖ It exports a common interface so that it's programs can be run on any system that supports .NET.

The Core of .NET Framework: FCL & CLR

Common Language Runtime:

- ❖ Garbage collection
- ❖ Language integration
- ❖ Multiple versioning support (no more DLL hell!)
- ❖ Integrated security



Common Language Runtime:

Manages running code – like a virtual machine, Threading Memory management

No interpreter: JIT-compiler produces native code – during the program installation or at run time

Managed Code: Code that targets the CLR is referred to as managed code ,All managed code has the features of the CLR

- Object-oriented
- Type-safe
- Cross-language integration
- Cross language exception handling
- Multiple version support

Managed code is represented in special Intermediate Language (IL)

Automatic Memory Management

The CLR manages memory for managed code

- All allocations of objects and buffers made from a Managed Heap
- Unused objects and buffers are cleaned up automatically through Garbage Collection

Some of the worst bugs in software development are not possible with managed code

- Leaked memory or objects
- References to freed or non-existent objects
- Reading of uninitialised variables

Pointer less environment

Multiple Language Support

IL (MSIL or CIL) – Intermediate Language:

- It is low-level (machine) language, like Assembler, but is Object-oriented

CTS is a rich type system built into the CLR:

- Implements various types (int, float, string, …)
- And operations on those types

CLS is a set of specifications that all languages and libraries need to follow

- ❖ This will ensure interoperability between languages

Intermediate Language:

- ❖ .NET languages are compiled to an Intermediate Language (IL)
- ❖ IL is also known as MSIL or CIL
- ❖ CLR compiles IL in just-in-time (JIT) manner – each function is compiled just before execution
- ❖ The JIT code stays in memory for subsequent calls
- ❖ Recompilations of assemblies are also possible

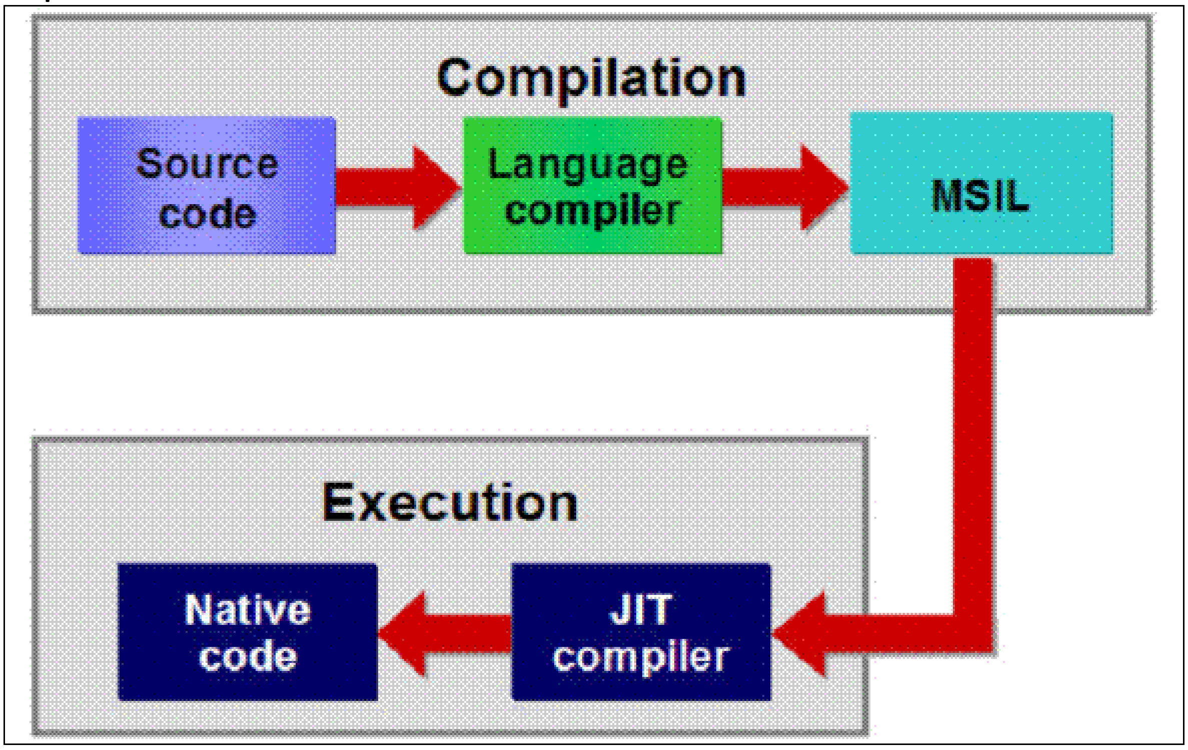
Common Type System (CTS)

- ❖ All .NET languages have the same primitive data types. An int in C# is the same as an int in VB.NET
- ❖ When communicating between modules written in any .NET language, the types are guaranteed to be compatible on the binary level
- ❖ Types can be:
    - ❖ Value types – passed by value, stored in the stack
    - ❖ Reference types – passed by reference, stored in the heap
- ❖ Strings are a primitive data type now

Common Language Specification (CLS):

- ❖ Any language that conforms to the CLS is a .NET language
- ❖ A language that conforms to the CLS has the ability to take full advantage of the Framework Class Library (FCL)
- ❖ CLS is standardized by ECMA

Code Compilation and Execution:



VISUAL STUDIO EDITIONS:

Express Edition:

The Visual Studio Express Editions are an expansion of the Visual Studio and SQL Server product line to include fun, simple and easy-to-learn tools for non-professional developers like hobbyists,

students and novice developers who want to build dynamic Windows applications, Web sites, and Web services. The Express products consist of:
• Visual Basic 2008 Express Edition - Productivity that is ideal for first time or casual Windows programming
• Visual C# 2008 Express Edition – A great combination of power and productivity for the Windows developer.
• Visual C++ 2008 Express Edition – Horsepower with a finer degree of control than other Express Editions
• Visual Web Developer 2008 Express Edition – An easy-to-use environment for dynamic Web application development.
• SQL Server 2008 Express and SQL Server Compact Edition – A powerful and easy-to-use set of databases to complement each express Edition.

Professional Edition:

Visual Studio 2008 Professional Edition is a comprehensive set of tools that accelerates the process of turning the developer's vision into reality. Visual Studio 2008 Professional Edition was engineered to support development projects that target the Web (including ASP.NET AJAX), Windows Vista, Windows Server 2008, The 2007 Microsoft Office system, SQL Server 2008, and Windows Mobile devices. The number of platforms that developers must target to meet business needs is increasing rapidly. Visual Studio 2008 Professional Edition provides the integrated toolset for addressing all of these needs by providing a superset of the functionality available in Visual Studio 2008 Standard Edition.

Standard Edition:

Visual Studio 2008 Standard Edition provides a full-featured development environment for Windows and Web developers. It offers productivity enhancements for building data-driven client and Web applications. Individual developers looking to create connected applications with next-generation user experiences will find Visual Studio 2008 Standard Edition a perfect fit.

NAMESPACE:

A *namespace* is a method of organizing a group of assemblies, classes, or types. A namespace acts as a container—like a disk *folder*—for classes organized into groups usually based on functionality. C# namespace syntax allows namespaces to be nested. For instance, to access the built-in input-output (I/O) classes and members, use the System.IO namespace. Or, to access Web-related classes and members, use the System.Web namespace.
All C# programs should call the System namespace—the father of all .NET Framework namespaces.
A namespace is similar to a Java package. However, whereas Java package names dictate the source files directory structure, C# namespaces dictate only the logical structure.

Using namespaces:

Namespaces are referenced by C# applications by using the using keyword. For instance, the following example references the System namespace:

```
using System;
```

Unlike the Java language, an individual class cannot be referenced with the using keyword. A compilation error would result.
Referencing System namespace gives access to the Console class which can be used to perform console I/O. For instance, the following source code writes a line of text to the Console:

```
Console.WriteLine ("A console message.");
```

Commonly used namespaces

The following is list of some important and frequently used .NET namespaces:

- *System.Collections*
- *System.Data*

- ***System.Diagnostics***
- ***System.Drawing***
- ***System.IO***
- ***System.Net***
- ***System.Reflection***
- ***System.Runtime***
- ***System.Security***
- ***System.Threading***
- ***System.Web***
- ***System.Windows.Forms***
- ***System.Xml***

A C# namespace is the equivalent of a Java language package.

ASSEMBLY AND METADATA

Metadata in .Net is binary information which describes the characteristics of a resource . This information include Description of the Assembly , Data Types and members with their declarations and implementations, references to other types and members , Security permissions etc. A module's metadata contains everything that needed to interact with another module.

During the compile time Metadata created with Microsoft Intermediate Language (MSIL) and stored in a file called a Manifest . Both Metadata and Microsoft Intermediate Language (MSIL) together wrapped in a Portable Executable (PE) file. During the runtime of a program Just In Time (JIT) compiler of the Common Language Runtime (CLR) uses the Metadata and converts Microsoft Intermediate Language (MSIL) into native code. When code is executed, the runtime loads metadata into memory and references it to discover information about your code's classes, members, inheritance, and so on. Moreover Metadata eliminating the need for Interface Definition Language (IDL) files, header files, or any external method of component reference.

Microsoft .Net Assembly is a logical unit of code, that contains code which the Common Language Runtime (CLR) executes. It is the smallest unit of deployment of a .net application and it can be a .dll or an exe . Assembly is really a collection of types and resource information that are built to work together and form a logical unit of functionality. It include both executable application files that you can run directly from Windows without the need for any other programs (.exe files), and libraries (.dll files) for use by other applications.

Assemblies are the building blocks of .NET Framework applications. During the compile time Metadata is created with Microsoft Intermediate Language (MSIL) and stored in a file called Assembly Manifest . Both Metadata and Microsoft Intermediate Language (MSIL) together wrapped in a Portable Executable (PE) file. Assembly Manifest contains information about itself. This information is called Assembly Manifest, it contains information about the members, types, references and all the other data that the runtime needs for execution.

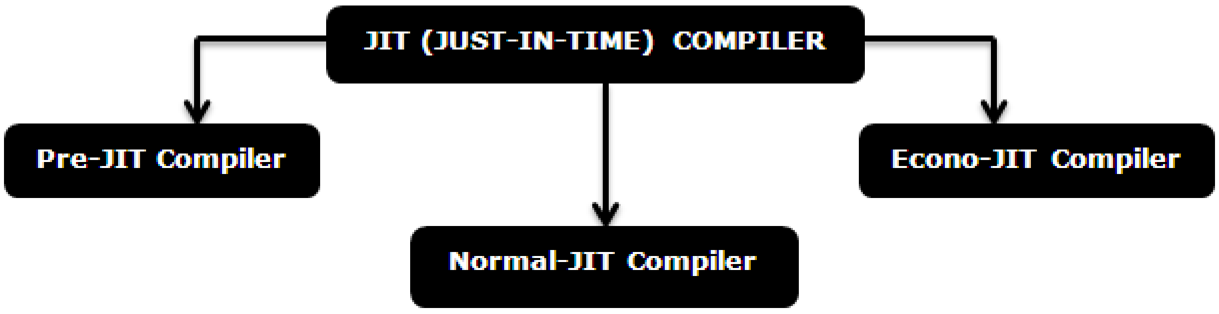
JIT (Just In Time Compilation)

Before MSIL(MS Intermediate Language) can be executed, it must converted by .net Framework Just in time (JIT) compiler to native code, which is CPU specific code that run on some computer architecture as the JIT compiler. Rather than using time and memory to convert all the MSIL in portable executable (PE) file to native code, it converts the MSIL as it is needed during execution and stored in resulting native code so it is accessible for subsequent calls.

JIT Types:

In Microsoft .NET there are three types of JIT (Just-In-Time) compilers which are Explained as Under:

- Pre-JIT Compiler (Compiles entire code into native code completely)
- Econo JIT Compiler (Compiles code part by part freeing when required)
- Normal JIT Compiler (Compiles only that part of code when called and places in cache



Description:

- **Pre-JIT COMPILER**

  Pre-JIT compiles complete source code into native code in a single compilation cycle. This is done at the time of deployment of the application.

- **Econo-JIT COMPILER:**

  Econo-JIT compiles only those methods that are called at runtime. However, these compiled methods are removed when they are not required.

- **Normal-JIT COMPILER:**

  Normal-JIT compiles only those methods that are called at runtime. These methods are compiled the first time they are called, and then they are stored in cache. When the same methods are called again, the compiled code from cache is used for execution.

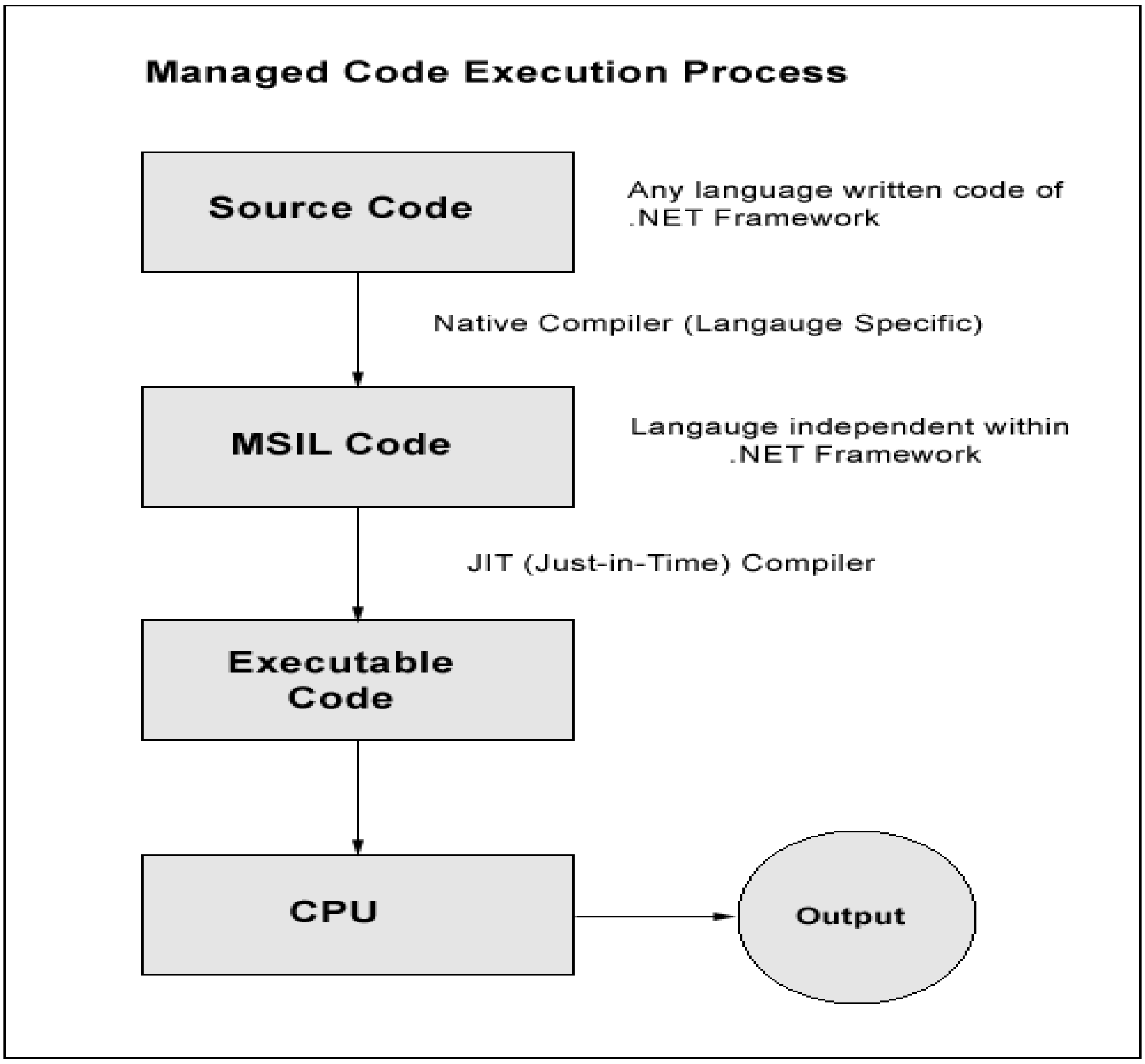
  These methods are compiled the first time they are called, and then they are stored in cache. When the same methods are called again, the compiled code from cache is used for execution

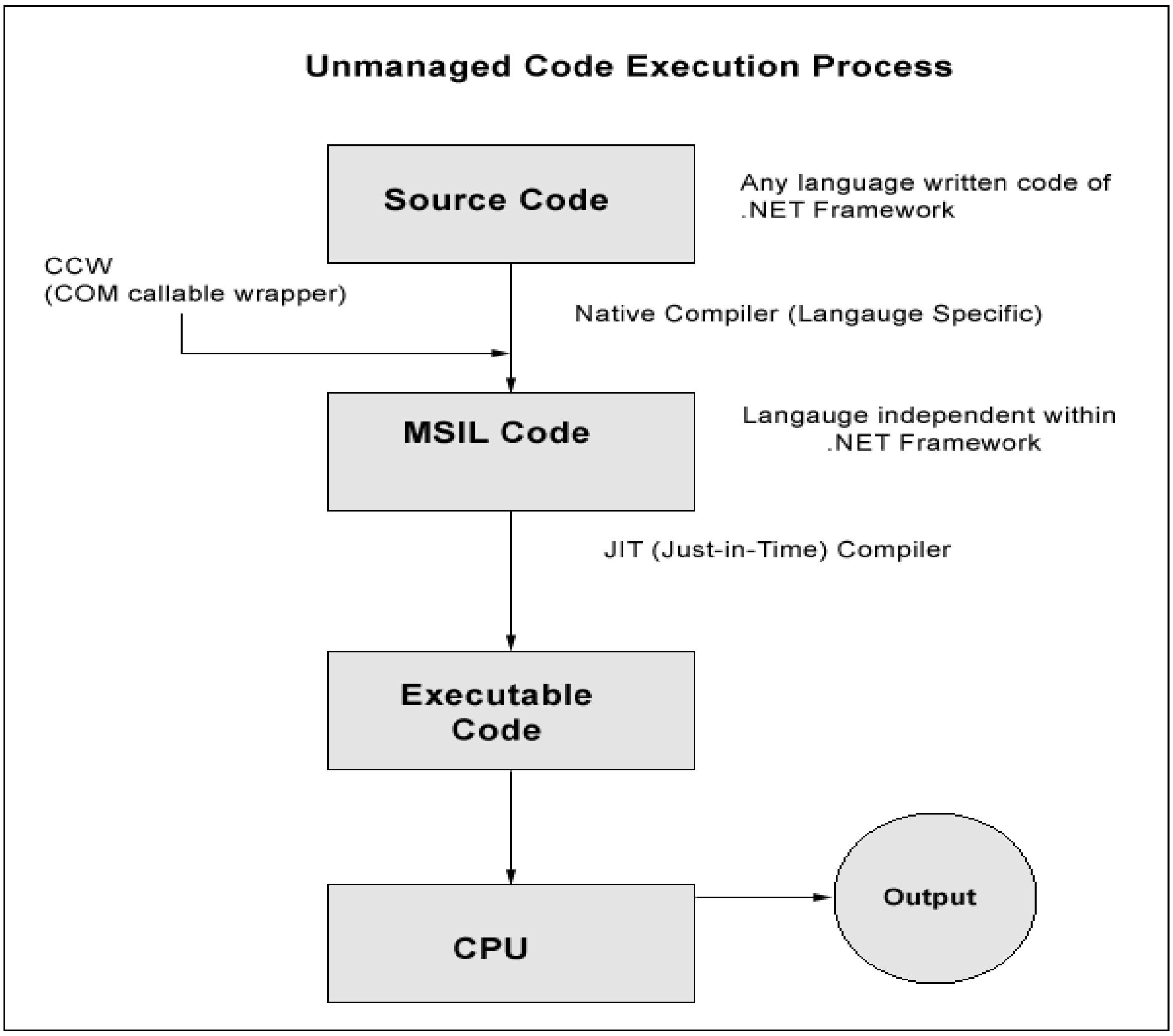
MANAGED CODE AND UNMANAGED CODE

Managed Code in Microsoft .Net Framework, is the code that has executed by the Common Language Runtime (CLR) environment. On the other hand Unmanaged Code is directly executed by the computer's CPU. Data types, error-handling mechanisms, creation and destruction rules, and design guidelines vary between managed and unmanaged object models.

The benefits of Managed Code include programmers convenience and enhanced security. Managed code is designed to be more reliable and robust than unmanaged code , examples are Garbage

Collection , Type Safety etc. The Managed Code running in a Common Language Runtime (CLR) cannot be accessed outside the runtime environment as well as cannot call directly from outside the runtime environment. This makes the programs more isolated and at the same time computers are more secure. Unmanaged Code can bypass the .NET Framework and make direct calls to the Operating System. Calling unmanaged code presents a major security risk.


Managed Code Execution Process
Source Code
Any language written code of .NET Framework
Native Compiler (Langauge Specific)
MSIL Code
Langauge independent within .NET Framework
JIT (Just-in-Time) Compiler
Executable Code
CPU
Output



Unmanaged Code Execution Process
Source Code
Any language written code of .NET Framework
CCW (COM callable wrapper)
Native Compiler (Langauge Specific)
MSIL Code
Langauge independent within .NET Framework
JIT (Just-in-Time) Compiler
Executable Code
CPU
Output

WHAT IS C#?

C# (pronounced "see sharp" or "C Sharp") is one of many .NET programming languages. It is object-oriented and allows you to build reusable components for a wide variety of application types. Microsoft introduced C# on June 26th, 2000 and it became a v1.0 product on Feb 13th 2002. C# is an evolution of the C and C++ family of languages. However, it borrows features from other programming languages, such as Delphi and Java. If you look at the most basic syntax of both C# and Java, the code looks very similar, but then again, the code looks a lot like C++ too, which is intentional. Developers often ask questions about why C# supports certain features or works in a certain way. The answer is often rooted in it's C++ heritage.

TYPES OF C# PROGRAMS:

• **Console applications:**—Console applications run from the command line. Throughout this book, you will create console applications, which are primarily character- or text-based and, therefore, remain relatively simple to understand.

• **Window forms applications:**—you can also create Windows applications that take advantage of the graphical user interface (GUI) provided by Microsoft Windows.

• **Web Services**—Web Services are routines that can be called across the Web.

• **Web form/ASP.NET applications:**—ASP.NET applications are executed on a Web server and generate dynamic Web pages.

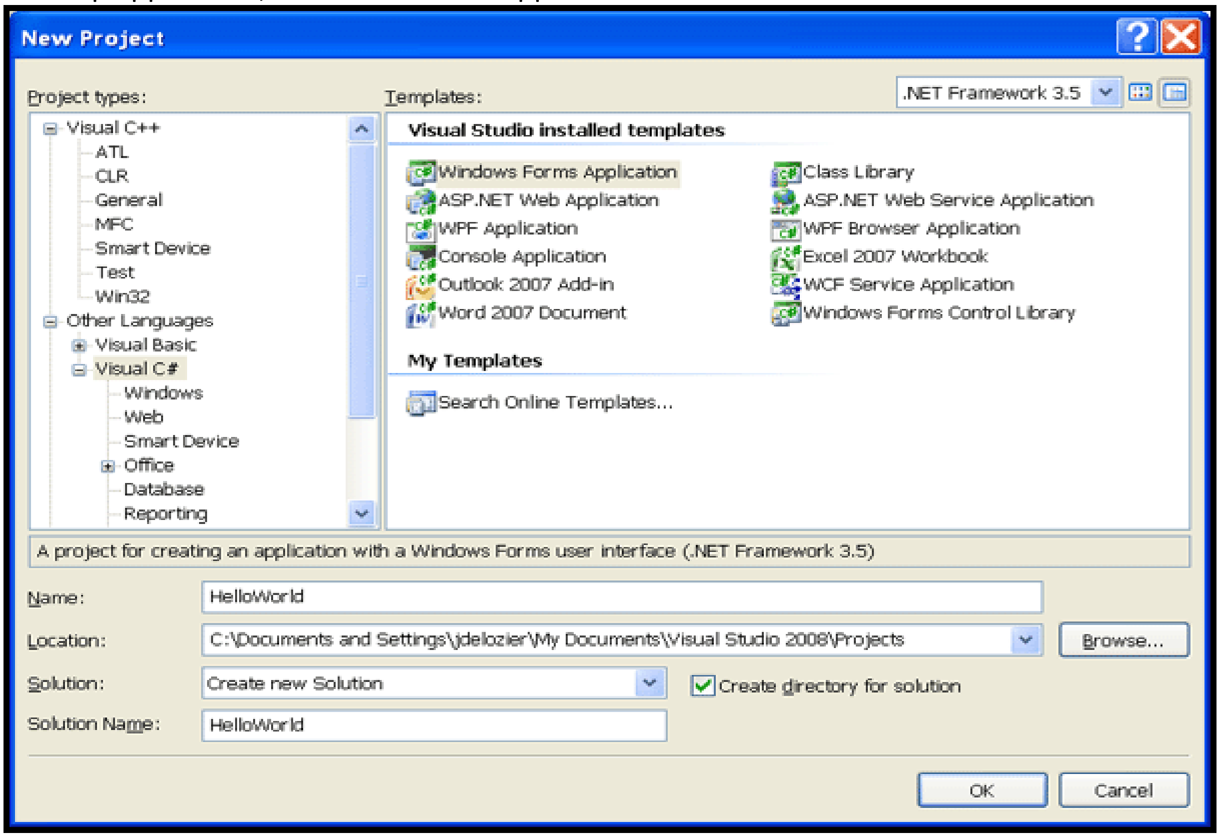
In addition to these types of applications, C# can be used to do a lot of other things,
Including create libraries, create controls, and more.

How Does A C# Application run?

An important point is that C# is a "managed" language, meaning that it requires the .NET Common Language Runtime (CLR) to execute. Essentially, as an application that is written in C# executes, the CLR is managing memory, performing garbage collection, handling exceptions, and providing many more services that you, as a developer, don't have to write code for. The C# compiler produces Intermediate Language (IL) , rather than machine language, and the CLR understands IL. When the CLR sees the IL, it Just-In-Time (JIT) compiles it, method by method, into compiled machine code in memory and executes it. As Mention previously, the CLR manages the code as it executes. Because C# requires the CLR, you must have the CLR installed on your system. All new Windows operating systems ship with a version of the CLR and it is available via Windows Update for older systems. The CLR is part of the .NET, so if you see updates for the .NET Framework Runtime, it contains the CLR and .NET Framework Class Library (FCL). It follows that if you copy your C# application to another machine, then that machine must have the CLR installed too.

Visual Studio Ide:

Programming has been made a lot easier, and using Visual Studio Integrated Development Environment (IDE) to create different types of applications really helps in developing a simple desktop application, web sites or web applications.



A project for creating an application with a Windows Forms user interface (.NET Framework 3.5)

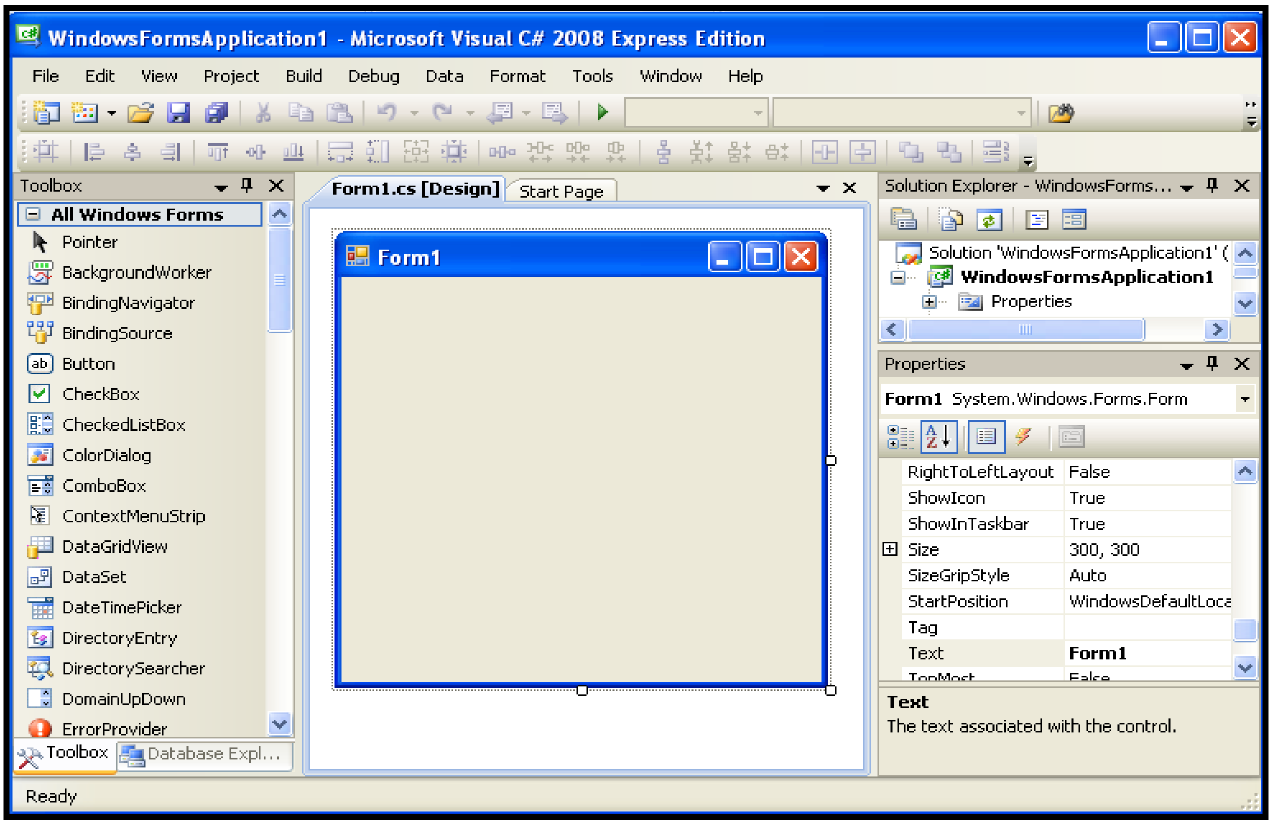
Name: HelloWorld

Location: C:\Documents and Settings\jdelozier\My Documents\Visual Studio 2008\Projects

Solution: Create new Solution

[✓] Create directory for solution

Solution Name: HelloWorld

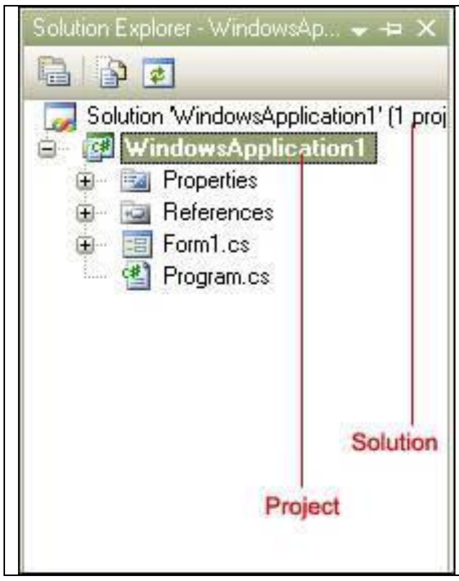


Form1 System.Windows.Forms.Form

RightToLeftLayout	False
ShowIcon	True
ShowInTaskbar	True
Size	300, 300
SizeGripStyle	Auto
StartPosition	WindowsDefaultLoca
Tag	
Text	Form1
TopMost	False

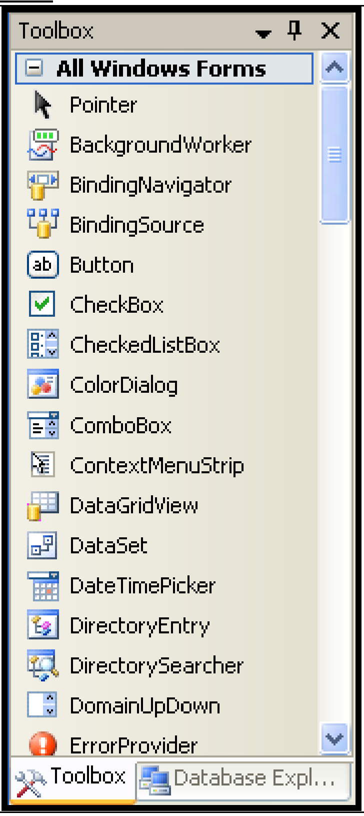
Text
The text associated with the control.

<u>SOLUTION EXPLORER:</u>



In simple terms, the solution explorer displays the various files and folders included in our project. These files can be the C# code files, class files, configuration file, XML files, database files and more. Folders are used to organize the files in groups. We can add new files and folders and import existing files into the project using the solution explorer.To view the solution explorer window, we need to move the mouse on the solution explorer tab in the IDE. The window will open up automatically on the right hand side of the IDE.

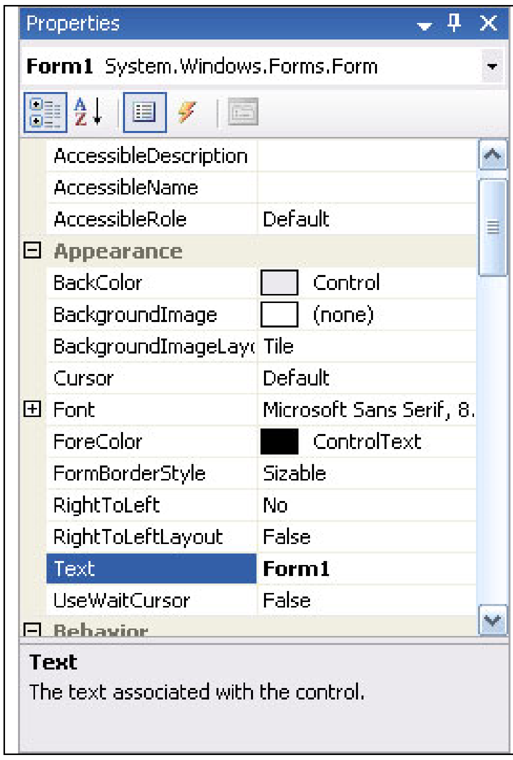
TOOLBAR:



The Toolbox contains the various elements that can be added to the forms while developing windows applications for user interaction. From simple controls like textbox, button, checkbox to the more advanced controls such as the menus and data access controls, all are located in the Toolbox. In the Toolbox, controls are grouped in different categories according to their use. We can collapse and expand the groups by clicking on the + or - symbols beside the name of the group.

Adding a control to a form is very simple. We can drag the control from the Toolbox and drop it on the form or double click the control and it will be added to the form.

PROPERTY WINDOW:



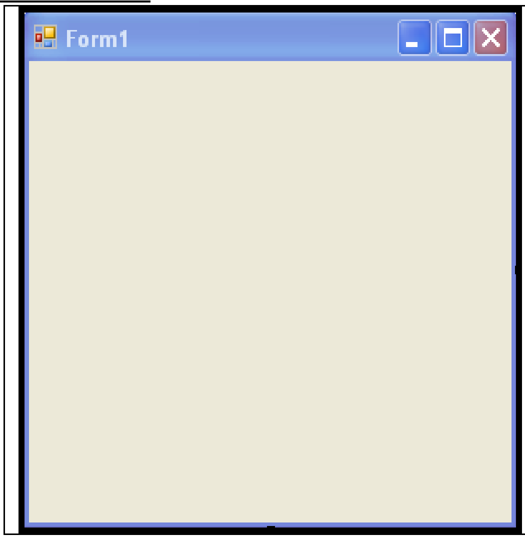
Form1 System.Windows.Forms.Form

AccessibleDescription	
AccessibleName	
AccessibleRole	Default
⊟ **Appearance**	
BackColor	Control
BackgroundImage	(none)
BackgroundImageLayı	Tile
Cursor	Default
⊞ Font	Microsoft Sans Serif, 8.
ForeColor	■ ControlText
FormBorderStyle	Sizable
RightToLeft	No
RightToLeftLayout	False
Text	**Form1**
UseWaitCursor	False
⊟ **Behavior**	

Text
The text associated with the control.

The Properties Window is used for configuring properties and events on controls in your user interface. Properties specify an object's attributes, such as the name, text and color of a control object, without writing code.

FORM DESIGNER:

	The Forms Designer window allows you to edit the user interface of the windows application that you are developing. Objects such as buttons and textboxes may be added quickly allowing you to rapidly create an application with only needing to add a few lines of code yourself.

VARIABLES:

Variables are programming elements that can change during program execution. They are used as storage to hold information at any stage of computation.A variable declaration has the following form:

`typename varname;`

`typename` specifies the variable type.

`varname` is the name of the variable. To declare a variable that can hold a standard numeric integer, you use the following line of code:

`int my_number;`

The name of the variable declared is ***`my_number.`*** The data type of the variable is ***`int.`*** As you will learn in the following section, the type ***`int`*** is used to declare integer variables, You can also declare multiple variables of the same type on one line by separating the variable names with commas. This enables you to be more concise in your listings. Consider the following line:

`int count, number, start;`

This line declares three variables: ***`count, number,`*** and ***`start.`*** Each of these variables is type ***`int,`*** which is for integers.

Now that you know how to declare a variable, it is important to learn how to store values. After all, the purpose of a variable is to store information.

The format for storing information in a variable is as follows:

`varname = value;`

You have already seen that ***`varname`*** is the name of the variable. ***`value`*** is the value that will be stored in the variable. For example, to store the number 5 in the variable, ***`my_variable`***, you enter the following:

`my_variable = 5;`

You can assign a value to a variable any time after it has been declared. You can even do this at the same time you declare a variable:

`int my_variable = 5;`

A variable's value can also be changed. To change the value, you simply reassign a new value:

`my_variable = 1010;`

CONSTANT:

A constant member is defined at compile time and cannot be changed at runtime. Constants are declared as a field, using the `const` keyword and must be initialized as they are declared. For example;

```
public class MyClass
{
  public const double PI = 3.14159;
}
```

`PI` cannot be changed in the application anywhere else in the code as this will cause a compiler error.

STRINGS:

A string in CSharp is an object of type String whose value is text. The string type represents a string of Unicode Characters

String is an alias for System.String in the .NET Framework. Initialize a string with the Empty constant value to create a new String object whose string is of zero length.For examples:

```
C#  Runtime type : System.String
CSharp declaration : string str;
CSharp Initialization : str = "csharp string";
```

DATA TYPES:

Data Types in a programming language describes that what type of data a variable can hold. CSharp is a strongly typed language, therefore every variable and object must have a declared type. The CSharp type system contains three Type categories. They are Value Types , Reference Types and Pointer Types . In CSharp it is possible to convert a value of one type into a value of another type. The operation of Converting a Value Type to a Reference Type is called Boxing and the reverse operation is called Unboxing. When we declare a variable, we have to tell the compiler about what type of the data the variable can hold or which data type the variable belongs to.

```
  Syntax : DataType VariableName
  DataType : The type of data that the variable can hold
  VariableName : the variable we declare for hold the values.
```

EXAMPLE:

```
  int count;
  int : is the data type
  count : is the variable name
```

The above example shows , declare a variable 'count' for holding an integer values.

INTEGER:

C# Alias	.NET Type	Size (bits)	Range
Sbyte	System.SByte	8	-128 to 127
Byte	byte System.Byte	8	0 to 255
short	System.Int16	16	-32,768 to 32,767
ushort	System.UInt16	16	0 to 65,535
char	System.Char	16	A unicode character of code 0 to 65,535
int	System.Int32	32	-2,147,483,648 to 2,147,483,647
uint	System.UInt32	32	0 to 4,294,967,295
long	System.Int64	64	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
ulong	System.UInt64	64	0 to 18,446,744,073,709,551,615

FLOATING-POINT:

C# Alias	.NET Type	Size (bits)	Precision	Range
float	System.Single	32	7 digits	1.5 x 10-45 to 3.4 x 1038
double	System.Double	64	15-16 digits	5.0 x 10-324 to 1.7 x 10308
decimal	System.Decimal	128	28-29 decimal places	1.0 x 10-28 to 7.9 x 1028

OTHER PREDEFINED TYPES:

C# Alias	.NET Type	Size (bits)	Range
bool	System.Boolean	32	true or false, which aren't related to any integer in C#.
object	System.Object	32/64	Platform dependant (a pointer to an object).
string	System.String	16 * length	A unicode string with no special upper bound.

BOXING AND UNBOXING

C# Type System contains three Types , they are Value Types , Reference Types and Pointer Types. C# allows us to convert a Value Type to a Reference Type, and back again to Value Types . The operation of Converting a Value Type to a Reference Type is called Boxing and the reverse operation is called Unboxing**.**

Boxing

```
  1:    int Val = 1;
  2:    Object Obj = Val; //Boxing
```

The first line we created a Value Type Val and assigned a value to Val. The second line , we created an instance of Object Obj and assign the value of Val to Obj. From the above operation (Object Obj = i) we saw converting a value of a Value Type into a value of a corresponding Reference Type . These types of operation is called Boxing.

UnBoxing

```
  1:    int Val = 1;
  2:    Object Obj = Val; //Boxing
  3:    int i = (int)Obj; //Unboxing
```

The first two line shows how to Box a **Value** Type . The next line (int i = (int) Obj) shows extracts the Value Type from the Object . That is converting a value of a Reference Type into a value of a Value Type. This operation is called UnBoxing.
Boxing and UnBoxing are computationally expensive processes. When a value type is boxed, an entirely new object must be allocated and constructed , also the cast required for UnBoxing is also expensive computationally.

C# - Operators

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

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators

This tutorial will explain the arithmetic, relational, logical, bitwise, assignment and other operators one by one.

Arithmetic Operators

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

Show Examples

Operator	Description	Example
+	Adds two operands	A + B will give 30
-	Subtracts second operand from the first	A - B will give -10

*	Multiplies both operands	A * B will give 200
/	Divides numerator by de-numerator	B / A will give 2
%	Modulus Operator and remainder of after an integer division	B % A will give 0
++	Increment operator increases integer value by one	A++ will give 11
--	Decrement operator decreases integer value by one	A-- will give 9

Relational Operators

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

Show Examples

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

Logical Operators

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

Show Examples

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

Bitwise Operators

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

p	q	p & q	p | q	p ^ q
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

ARRAY:

Arrays are using for store similar data types grouping as a single unit. We can access Array elements by its numeric index. The array indexes start at zero. The default value of numeric array elements are set to zero, and reference elements are set to null.

```
  string[] week = new string[7];
  week[0] = "Sunday";
  week[1] = "Monday";
```

The above C# code declare a string array of 7 strings and assign some values to it.

```
string[] week = new string[] {"Sunday","Monday","Tuesday"};
```

The above code declare and initialize a string array with values.

```
string str = week[1];
```

We can access the Arrays elements by providing its numerical index, the above statement we access the second value from the week Array. For finding the end of an Array we used the Length function of Array Object.

Jagged array:

In both the C and C++ languages, each sub array of a particular multi-dimensional array must have identical dimensions. In other words, arrays must be *orthogonal*. However, in both the Java and C# languages, arrays need not be orthogonal; because, arrays are constructed as arrays of arrays.

In C#, each array is one-dimensional. Therefore, *jagged* arrays of varying sizes can be built. The content of a jagged array is arrays of instances or of references to arrays. Therefore, the rows and columns of a jagged array need not be of uniform length. The following C# example illustrates how to construct a jagged array—the code works in either the Java or C# language:

```
int [][]JA = new int[3][];
JA [0] = new int[2]{3,4};
JA [1] = new int[4]{1,2,3,4};
JA [2] = new int[3]{1,2,3};
Console.WriteLine("Array Elements are…");
for(int i=0;i<JA.Length ;i++)
{
     for(j=0;j<JA[i].Length;j++)
      {
               Console.WriteLine(JA[i][j]+"\t");
       }
      Console.witeLine("\n\n");
}
Console.ReadLine();
```

```
Output:
3 4
```

```
1 2 3 4
1 2 3
```

DECISION STATEMENTS:

If –else statement:

The conditional statement if and else in C# is using for check the conditions that we provided in the head of if statement and making decision based on that condition. The conditional statement examining the data using comparison operators as well as logical operators. The else statement is optional , so we can use the statement in two ways ;

```
if (condition)
statement;
if (condition)
        statement;
else
        statement;
```

If the condition is true then the control goes to the body of if block, that is the program will execute the code inside if block. If the condition is false then the control goes to next level, that is if you provide else block the program will execute the code block of else statement, otherwise the control goes to next line of code. If you want o check more than one conditions at the same time, you can use else if statement.

```
if (condition)
        statement;
else if (condition)
        statement;
else
        statement;
```

Just take a real-time example - We have a mark list and we want to analyze the grading of each student. In this case we can use if-Else conational statements.

```
if (totalMarks >= 80)
 {
           MessageBox.Show("Got Higher First Class ");
 }
 else if (totalMarks >= 60)
{
          MessageBox.Show("Got First Class ");
 }
else if (totalMarks >= 40)
{
          MessageBox.Show("Just pass only");
}
else
{
           MessageBox.Show("Failed");
 }
```

LOOPING STATEMENTS:

For:

The **for loop** in C# is useful for iterating over arrays and for sequential processing. That is the statements within the code block of a for loop will execute a series of statements as long as a specific condition remains true.

Syntax:

```
for(initialization; condition; step)
statement
```

```
initialization     : Initialize the value of variable.
condition          : Evaluate the condition
step               : Step taken for each execution of loop body
```

The for loop initialize the value before the first step. Then checking the condition against the current value of variable and executes the loop statement and then performs the step taken for each execution of loop body.

```
int count = 4;
for (int i = 1; i < = count; i++)
{
        MessageBox.Show("Current value of i is - " + i);
}
```

Foreach:

The foreach loop in C# executes a block of code on each element in an array or a collection of items. When executing foreach loop it traversing items in a collection or an array. The foreach loop is useful for traversing each item in an array or a collection of items and displayed one by one.

```
  foreach(variable type in collection)
{
    // code block
  }
  variable type : The variable used for collect the item from Collection

  collection     : Collection of items
  string[] days = { "Sunday", "Monday", "TuesDay"};
  foreach (string day in days)
  {
          MessageBox.Show("The day is : " + day);
  }
```

The above C# example first declared a string array 'days' and initialize the days in a week to that array. In the foreach loop declare a string 'day' and pull out the values from the array one by one and displayed it.

While:

The C# while statement continually executes a block of statements until a specified expression evaluates to false .

```
while (expression) statement
```

Like if statement the while statement evaluates the expression, which must return a boolean value. If the expression evaluates to true, the while statement executes the statement(s) in the while block. The while statement continues testing the expression and executing its block until the expression evaluates to false.

```
int count = 1;
while (count < = 4)
{
        MessageBox.Show("The value of i is : " + count);
        count = count + 1;
}
```

The C# while statement executes a statement or a block of statements until a specified expression evaluates to false . The above program the loop will execute the code block 4 times.You can implement an infinite loop using the while statement as follows:

```
while (true)
{
        // statements
    }
```

Do...While:

The C# while statement executes a statement or a block of statements until a specified expression evaluates to false . In some situation you may want to execute the loop at least one time and then check the condition. In this case you can use do..while loop.

The difference between do..while and while is that do..while evaluates its expression at the bottom of the loop instead of the top. Therefore, the statements within the do block are always executed at least once. From the following example you can understand how do..while loop function.

Switch case:

The C# switch statement allows you to choose from many statements based on multiple selections by passing control to one of the case statements within its body. The switch statement executes the case corresponding to the value of the expression . The switch statement can include any number of case instances.

```
  switch (expression)
  {
     case expression:
           //your code here
           jump-statement
     default:
           //your code here
           jump-statement
  }
```

expression : An integral or string type expression

jump-statement : A jump statement that transfers control out of the case body.

If any of the expression passed to switch case does not match with case statement the control will go to default: statement . If there is no default: statement control will go outside of the switch statement.

EXCEPTION USING TRY AND CATCH:

The C# language uses exceptions to handle errors and other exceptional events. Exceptions are the occurrence of some conditions that changes the normal flow of execution. Exceptions are occurred in situations like your program run out of the memory, file does not exist in the given path, network connections are dropped etc. More specifically for better understanding, we can say it as Runtime Errors occurs during the execution of a program that disrupts the normal flow of instructions In .NET languages, Structured Exceptions handling is a fundamental part of Common Language Runtime . All exceptions in the Common Language Runtime are derived from a single base class , also you can create your own custom exception classes. You can create an exception class that inherits from Exception class. Creating an exception object and handing it to the runtime system is called throwing an exception. C# Exception handling uses the try, catch, and finally keywords to attempt actions that may not succeed, to handle failures, and to clean up resources afterwards.

```
  try
  {
        //your code here
  }
  Catch (exception type)
  {
        //your code here
  }
finally
```

The code in the finally block will execute even if there is no Exceptions. That means if you write a finally block , the code should execute after the execution of try block or catch block.

```
try
{
        //your code here
}
Catch (exception type)
{
        //if the exception occurred and your code here
}
finally
{
        //your code here
}
```

From the following CSharp code, you can understand how to use try.Catch statements. Here we are going to divide a number by zero.

EXAMPLE:

```
try
{       int val = 100;
        int div = 0;
        int resultVal;
        resultVal = (val / div);
        MessageBox.Show("The result is  : " + resultVal);
 }
catch (System.Exception  ex)
{
  MessageBox.Show("Exception catch here - details  : " +   x.ToString());
}
finally
{
     MessageBox.Show("Enter finally block ");
 }
```

Chapter: 2:-CLASS AND INHERITANCE,PROPERTY, INDEXER, POINTERS, DELEGATES, EVENTS, COLLECTIONS

CLASS:

Classes are at the heart of every object-oriented language. , "Fundamentals of Object-Oriented Programming," a class is the encapsulation of data and the methods that work on that data. That's true in any object-oriented language.

Class is group of data members and methods. A class is a template that defines the form of an object. It means to use the class members and methods we should create instance or object of class. To declare a class we use class keyword. The general form of a class definition that contains only instance variables and methods.

The syntax used for defining classes in C# is simple, especially if you usually program in C++ or Java. You place the class keyword in front of the name of your class and then insert the class's members between the "curlies," like so: -

```
class classname
{
     declare instance variables
     access type variablename;
    //declare methods
    access ret-type methodname(parameters)
    {
      //body of method
    }
}
```

Here

1. Access specifies the access.
2. Type specifies the type of variable, and
3. Variablename is the variable’s name

Class Members:

Data members are those members that contain the data for the class like fields, constant, and events. Data members can be either static or instance. As usual for object oriented language, a class member is always an instance member is it is explicitly declared as static.

Function members:

Function member are those members that provide some functionality for manipulating the data in the class. They include methods, properties, constructors, finalizers, operators, and indexers.

OBJECT:

When you want to work with class members at that time you have to create instance of class which is known as object. For ex. If you want to represent an employee using employee class than you have to create instance of employee class first then you can provide data to its attribute like empid, fname and etc. All the objects of same class are different than each other in terms of attribute’s value. The general form for declaring an instance and initialized the object is shown here.

```
            classname instancename;
            instancename  =new classname();
```

here , new operator is used to create an object of the class. Now to access members and methods of the class we will use following form:

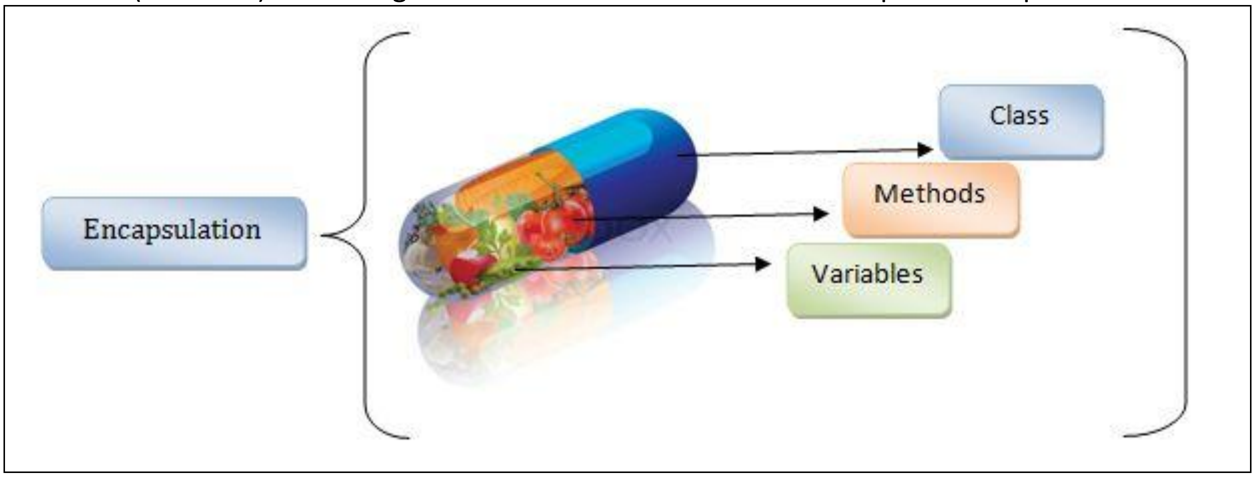
```
               objectname.memebername;
               objectname.methodname;
```

here to access these variables and methods, ou will use the dot(.) operator. The dot operator links the name of an object with the name of member.

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.

"Encapsulation is a process of binding data members (variables and properties) and member functions (methods) into a single unit". And a class is the best example of encapsulation.



INHERITING A CLASS:

Inheritance is the ability to create a class from another class, the "parent" class, extending the functionality and state of the parent in the derived, or "child", class.

Inheritance in C# also allows derived classes to overload methods from their parent class.C# is an object-oriented programming language. It supports code reusability. C# programs consist of classes. New classes can be created from an existing class. The new class may have some of the properties of an existing class. The mechanism of constructing one class from another is called inheritance. This is one of the ideas behind reusability of code. Another feature related to inheritance and reusability of code is polymorphism. Polymorphism permits the same method name to be used for different operations on different data types. Thus, C# supports code reusability by the features of inheritance and polymorphism.

PROGRAM:

```
using System;
public class ParentClass
{
    public ParentClass()
    {
        Console.WriteLine("Parent Constructor.");
    }

    public void print()
    {
        Console.WriteLine("I'm a Parent Class.");
    }
}
public class ChildClass : ParentClass
{
    public ChildClass()
    {
        Console.WriteLine("Child Constructor.");
    }

    public static void Main()
    {
        ChildClass child = new ChildClass();

        child.print();
    }
}
```

```
Output:  Parent Constructor.
   Child Constructor.
   I'm a Parent Class
```

POLYMORPHISM:

Polymorphism is an object-oriented programming concept that refers to the ability of a variable, function or object to take on multiple forms. A language that features polymorphism allows developers to program in the general rather than program in the specific.

ABSTRACT CLASS

A class defined as abstract is used as a base class. Such a class is used for the purpose of inheritance only i.e. other classes are derived from this class. We cannot create an object of an abstract class. The syntax for creating an abstract class is:

```
abstract class class-name
{
        // members of the class
}
```

An abstract class may contain methods and properties. The classes derived from the abstract class inherit these methods and properties. The abstract class may also contain abstract methods and properties. Abstract method and properties do not have any functionality. The derived class defines their full functionality.Here is an example of an abstract class:

```
abstract class MyAbstract
{
        public abstract void AbMethod();
}
```

In the above example, the class MyAbstract is defined as an abstract class. The class contains a method - AbMethod() - also defined as abstract. Note that the method does not have any implementation i.e. we have not specified what the method is doing. If we create an object of this class, we will get an error. This is because we cannot create objects of an abstract class. To use this class in a program, we must first create a class that inherits from this abstract class:

```
class MyDerived : MyAbstract
{
        public override void AbMethod()
        {
                Console.WriteLine("A Method");
        }
 }
```

The MyDerived class is derived from the MyAbstract class. The derived class also provides the complete implementation of the abstract method. We should use the override keyword in such a case. This MyDerived class can have its objects in the program.

SEALED CLASS:

A class defined with the keyword sealed is used to prevent derivation from it i.e. other classes cannot inherit from this class. Hence, a sealed class cannot be a base class for other classes. A sealed class cannot be abstract as well. The syntax for a sealed class definition is:

```
sealed class class-name
{
        //members of the class
}
```

```
using System;
class Class1
{
    static void Main(string[] args)
    {
        SealedClass sealedCls = new SealedClass();
        int total = sealedCls.Add(4, 5);
        Console.WriteLine("Total = " + total.ToString());
    }
}
// Sealed class
sealed class SealedClass
{
    public int Add(int x, int y)
    {
        return x + y;
    }
}
```

OVERLOADING OPERATOR:

Operator overloading, also known as overloading, provides a way to define and use operators such as +, -, and / for user-defined classes or structs. It allows us to define/redefine the way operators work with our classes and structs. This allows programmers to make their custom types look and feel like simple types such as int and string. It consists of nothing more than a method declared by the keyword operator and followed by an operator. There are three types of overloadable operators called unary, binary, and conversion. Not all operators of each type can be overloaded.

Unary operator overloading example:

```
using System;
class Complex
{
   private int x;
   private int y;
   public Complex()
```

```
    {
    }
    public Complex(int i, int j)
    {
       x = i;
       y = j;
    }
    public void ShowXY()
    {
      Console.WriteLine(\"{0} {1}\",x,y);
    }
    public static Complex operator -(Complex c)
    {
           Complex temp = new Complex();
           temp.x = -c.x;
           temp.y = -c.y;
           return temp;
     }
}
class MyClient
{
     public static void Main()
     {
          Complex c1 = new Complex(10,20);
          c1.ShowXY(); // displays 10 & 20
          Complex c2 = new Complex();
          c2.ShowXY(); // displays 0 & 0
          c2 = -c1;
          c2.ShowXY(); // diapls -10 & -20
       }
}
```

METHODS WITH "ref" AND "out" PARAMETERS:

ref variable is also used to send object to a function as reference-by-reference. In C#, object reference usually sends reference-by-value meaning the reference of a variable is copied to other variable. They are pointing to the same object but have their own copy of reference. The parameter variable cannot change the original variable reference.

Sending reference-by-reference is a method that both the real variable and the parameter variable as the same copy of reference, the parameter variable in the calling function can change the object pointed by the original variable or by variable in the caller function.

```
using System;
class Program
{
    static void Main()
    {
       int val = 0;
       Example1(val);
       Console.WriteLine(val); // Still 0!
       Example2(ref val);
       Console.WriteLine(val); // Now 2!
       Example3(out val);
       Console.WriteLine(val); // Now 3!
    }
    static void Example1(int value)
    {
        value = 1;
    }
    static void Example2(ref int value)
    {
        value = 2;
    }
    static void Example3(out int value)
    {
        value = 3;
    }
}
```

```
Output
0
2
3
```

OVERLOADING METHOD:

Method overloading allows the programmer to define many methods with the same name but with a different set of parameters. Each combination of parameter types is known as a *signature* of the method. When a call is made to one of these overloaded methods, the compiler automatically determines which of the methods should be used according to the arguments used in the call and the available method signatures.One of the greatest advantages of method overloading is the improvement that it provides to code readability and maintainability. In addition of two or three integers, we add two integer values as a method, add three integer values as a method, as well. i.e:

```
public class AddingNumbers
{
    public int Add(int a, int b)
    {
        return a+b;
    }
    public int Add(int a, int b, int c)
    {
        return a+b+c;
    }
}
CALLING METHODS
public int add(int x, int y)
public int add(int x, int y, int z)
```

CONSTRUCTORS IN C-SHARP:

Constructor is a special method of a class which will invoke automatically whenever instance or object of class is created. Constructors are responsible for object initialization and memory allocation of its class. If we create any class without constructor, the compiler will automatically create one default constructor for that class. There is always at least one constructor in every class.

What is constructor?

- Constructor is used to initialize an object (instance) of a class.
- Constructor is a like a method without any return type.
- Constructor has same name as class name.
- Constructor follows the access scope (Can be private, protected, public, Internal and external).
- Constructor can be overloaded.

Default Constructor:

A constructor that takes no parameters is called a default constructor. When a class is initiated default constructor is called.

Constructor Overloading:

Constructors can be overloaded. Same constructor declared with different parameters in the same class is known as constructor overloading. Compiler differentiates which constructor is to be called depending upon the number of parameters and their sequence of data types.

```
using System;
public Class OverloadConst
{
      public OverloadConst()
      {
      //Default constructor
      }
      public OverloadConst(int age)
      {
      //Single parameter constructor
      }
      public OverloadConst(int age, string name)
      {
      //two parameter constructor
      }
}
```

Now, following are the ways through which we can instantiate the object

```
OverloadConst obj=new OverloadConst();//Default constructor will get called
OverloadConst obj=new OverloadConst(33);//Single parameter constructor will
get called
OverloadConst obj=new OverloadConst(35,"jjkcc");//Two parameters constructor
will get called
```

INTERFACE:

An interface looks similar to a class. It contains methods and properties but has no implementation. We do not specify what the methods and properties will actually do. The reason is that classes and structures inherit them and they provide the actual implementation for each interface member defined. The syntax for defining interface is as follows:

```
        interface IMyInterface
        {
                void SomeMethod();
        }
```

Notice that the SomeMethod() method does not have any implementation. This is because we do not use an interface directly in our program. We need a class or structure that will implement the interface:

```
        class MyClass : IMyInterface
        {
                public void SomeMethod()
                {
                        Console.WriteLine("This is a method");
                }
         }
```

Here, the class MyClass implements the SomeMethod() method of the interface. An interface is a type of contract that specifies that the class or structure inheriting the interface must implement its members.

INHERITING AN INERFACE:

An interface looks like a class, but has no implementation. The only thing it contains are definitions of events, indexers, methods and/or properties. The reason interfaces only provide definitions is because they are inherited by classes and structs, which must provide an implementation for each interface member defined.

PROGRAM:

```
using System;
 public interface A
{
  void meth1();
  void meth2();
}
public interface B : A
 {
  void meth3();
}
class MyClass : B
{
    public void meth1()
   {
        Console.WriteLine("Implement meth1().");
    }
    public void meth2()
   {
     Console.WriteLine("Implement meth2().");
   }
   public void meth3()
   {
    Console.WriteLine("Implement meth3().");
   }
}
public class IFExtend
{
  public static void Main()
  {
    MyClass ob = new MyClass();
     ob.meth1();
    ob.meth2();
    ob.meth3();
  }
}
```

Creating and Using Properties

Properties are members that provide a flexible mechanism to read, write, or compute the values of private fields. Properties can be used as if they are public data members, but they are actually special methods called accessors. You use a property to associate actions with the reading and writing of an object's attribute. Properties are implemented as member variable with public access. In C#, properties appear to be fields ato the user of a class, but they use method to get and set values. Properties provide:

A useful way to encapsulate information inside a class(Security).

Concise syntax.

Flexibility due to Field – like Access

1. Use **get accessor** statements to provide read access
2. Use **set accessor** statement to provide write access

Syntax of the get accessor:

```
Public <return type> <property name>
{
  get
     {
         return <attribute name>
      }
}
```

For example:

```
Public String Name
{
    get
      {
              return name;
     }
}
```

Syntax of the set Accessor:

```
Public <return type> <property name>
{
      Set
     {
           Attribute name =value;
    }
}
```

For example

```
Public string Name
{
     set
      {
          Name=value;
      }
}
```

Property consists of:

(1) It enables a class to expose a public way of getting and setting values, while hiding implementation or verification code.

(2) A **get** property accessor is used to return the property value, and a **set** accessor is used to assign a new value. These accessors can have different access levels.

(3) The **value**s keyword is used to define the value being assigned by the set indexer.

(4) Properties that do not implement a **set** method are read only.

Property Example:

```
using System;
class MyClass
{
       private int x;
       public int X
      {
      get
        {
          return x;
        }
       set
       {
            x = value;
       }
    }
}
class MyClient
{
    public static void Main()
    {
        MyClass mc = new MyClass();
        mc.X = 10;
       int xVal = mc.X;
       Console.WriteLine(xVal);//Displays 10
    }
}
Output: 10
```

INDEXER

Indexers behave like arrays in that they use the square-bracket syntax to access their members. The .NET collection classes use indexers to accomplish the same goals. Their elements are accessed by index. Indexers are implemented like properties because they have get and set accessors, following the same syntax. Given an index, they obtain and return an appropriate value with a get accessor. Similarly, they set the value corresponding to the index with the value passed into the indexer. Indexers also have a parameter list, just like methods. The parameter list is delimited by brackets. Normally, parameter types are commonly int, so a class can provide array-like operations, but other useful parameter types are string or a custom enum. For example:

```
using System;
class IndexDemo
    {
        private string[] data = new string[5];
        public string this[int index]
        {
                  get
                     {
                        return data[index];
                     }
                 set
                   {
                         data[index] = value;
                  }
           }
    }
    class MyClient
    {
        public static void Main()
        {  IndexDemo mc = new IndexDemo();
            mc[0] = "Rajesh";
            mc[1] = "Ravi";
            mc[2] = "Raj";
            mc[3] = "Dev";
            mc[4] = "Mumbai";
              Console.WriteLine("{0} ,{1} ,{2} ,{3}  ,{4}",
mc[0],mc[1], mc[2], mc[3], mc[4]);
        }
}
```

POINTER:

A pointer is a variable that holds the memory address of another type. In C#, pointers can only be declared to hold the memory addresses of value types (except in the case of arrays). In C sharp, they are considered as an unsafe code. We can use pointers in the similar manner as we would be using in the C++ or VC++, but only change is that in C sharp we have to build it up with the unsafe option/ unsafe keyword.

Declaring a Pointer type:

The general form of declaring a pointer type is as shown below

```
      type *variable_name;
```

Where * is known as the de-reference operator. For example the following statement

```
int *x ;
```

Declares a pointer variable x, which can hold the address of an int type. The reference operator (&) can be used to get the memory address of a variable.

```
int x = 100;
```

The &x gives the memory address of the variable x, which we can assign to a pointer variable

```
int *ptr = & x;.
Console.WriteLine((int)ptr) // Displays the memory address
Console.WriteLine(*ptr) // Displays the value at the memory address.
```

Unsafe Codes :

The C# statements can be executed either as in a safe or in an unsafe context. The statements marked as unsafe by using the keyword unsafe runs outside the control of Garbage Collector. Remember that in C# any code involving pointers requires an unsafe context. We can use the unsafe keyword in two different ways. It can be used as a modifier to a method, property, and constructor etc. For example

```
using System;
class MyClass
{
    public unsafe void Method()
    {
        int x = 10, y = 20;
        int* ptr1 = &x;
        int* ptr2 = &y;
        Console.WriteLine((int)ptr1);
        Console.WriteLine((int)ptr2);
        Console.WriteLine(*ptr1);
        Console.WriteLine(*ptr2);
    }
}
class MyClient
{
    public static void Main()
    {
        MyClass m = new MyClass();
        m.Method();
    }
}
```

Out put:

```
1242224, 1242224, 10 ,20
```

Using fixed:

The fixed modifier is often used when working with pointers. It prevents a managed variable from being moved by the garbage collector. This is needed when a pointer refers to a field in a class object, for example. Since the pointer has no knowledge of the actions of the garbage collector, if the object is moved, the pointer will point to the wrong object.

```
using System;
class MyClass
{
      public unsafe void Method()
      {
        int []iArray = new int[10];
        for(int count=0; count < 10; count++)
               iArray[count] = count*count;
         fixed(int *ptr = iArray)
         Display(ptr);
      }
     public unsafe void Display(int *pt)
     {
        for(int i=0; i < 10;i++)
               Console.WriteLine(*(pt+i));
     }
}
class MyClient
{
      public static void Main()
      {
          MyClass mc = new MyClass();
          mc.Method();
      }
}
```

Output:0,1,2,9,16,25,36,49,61,81

DELEGATES:

A delegate in C# is similar to a function pointer in C or C++. Using a delegate allows the programmer to encapsulate a reference to a method inside a delegate object. The delegate object can then be passed to code which can call the referenced method, without having to know at compile time which method will be invoked.

When you want to create a delegate in C# you make use of delegate keyword.

The name of your delegate can be whatever you desire. However, you must define the delegate to match the signature of the method it will point to. For example the following delegate can point to any method taking two integers and returning an integer.

```
      public delegate int DelegateName(int x, int y);
```

PROGRAM:

```
using System;
namespace Delegate
{
    public delegate int MyDelegate(int x,int y);
class MyClass
{
    public static int add(int x, int y)
    {
        return x + y;
    }
    public static int mul(int x, int y)
    {
        return x * y;
    }
}
class Program
```

```
 {
        static void Main(string[] args)
        {
            MyDelegate del1 = new MyDelegate(MyClass.add);
            int result = del1(5, 5);
            Console.WriteLine(" 5 + 5 ={0}", result);
            MyDelegate del2 = new MyDelegate(MyClass.mul);
            int resultm = del2(5, 5);
            Console.WriteLine(" 5 X 5 ={0}", resultm);
            Console.ReadKey();
        }
    }
}
```

Output:

```
5 + 5 = 10
5 X 5 = 25
```

MULTICAST DELEGATES:

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

```
using System;
public delegate void MulticastDelegate(int x, int y);
public class MyClass
{
    public static void Add(int x, int y)
    {
        Console.WriteLine("You are in Add() Method");
        Console.WriteLine("{0} + {1} = {2}\n", x, y, x + y);
    }
    public static void Multiply(int x, int y)
    {
        Console.WriteLine("You are in Multiply() Method");
        Console.WriteLine("{0} X {1} = {2}", x, y, x * y);
    }
}
class Program
{
    static void Main(string[] args)
    {
        MulticastDelegate del = new MulticastDelegate(MyClass.Add);
        del += new MulticastDelegate(MyClass.Multiply);
        Console.WriteLine("calling Add() and Multiply() Methods.");
        del(5, 5);
        del -= new MulticastDelegate(MyClass.Add);
        Console.WriteLine("\n****Add() Method removed.****\n");
        del(5, 5);
    }
}
```

```
calling Add() and Multiply() Methods
You are in add() method
5 + 5 =10
You are in multiply() method
5 X 5 =25
****Add() Method removed.****
You are in multiply() method
5 X 5 =25
```

EVENTS:

Events are the messages sent by an object to indicate the occurrence of an event. Event can also be defined as a member that enables an object to provide notification. Events provide every powerful means of inter-process communication. The most familiar example of events are graphical user interface, events are fired when any control is clicked on the GUI. Events are not used only for graphical user interfaces. Events provide a generally useful way for objects to signal state changes that may be useful to the client of that object. In C# events are used with delegates.

CREATING EVENTS:

SYNTAX:

```
[modifiers] event type declarator;
```

The modifier can be one or a combination of the following keywords: **public**, **private**, **protected**, **internal**, **abstract**, **new**, **override**, **static**, **virtual**, or **extern**.The **event** keyword is required. It is followed by the name of the delegate that specifies its behavior. If the event is declared in the main class, it should be made static. Like everything in a program, an event must have a name. This would allow the clients to know what (particular) event occurred.

EXAMPLE:

```
public class MyClass
{
    public delegate void MyDelegate(string message);
    public event MyDelegate MyEvent;
}
```

PROGRAM:

```
using System;
namespace delegate_custom
{
    class Program
    {
        public delegate void MyDelegate(int a);
        public class XX
       {
            public event MyDelegate MyEvent;
            public void RaiseEvent()
            {
                MyEvent(20);
                Console.WriteLine("Event Raised");
            }
            public void Display(int x)
            {
                Console.WriteLine("Display Method {0}", x);
            }
        }
        static void Main(string[] args)
        {
            XX obj = new XX();
```

```
                obj.MyEvent += new MyDelegate(obj.Display);
                obj.RaiseEvent();
                Console.ReadLine();
           }

      }

}
```

```
Display method 20
Event Raised
```

- **COLLECTION AND GENERICS**

UNDERSTANDING COLLECTIONS:

CSharp Collections are data structures that hold data in different ways for flexible operations. C# Collection classes are defined as part of the System.Collections or System.Collections.Generic namespace.Most collection classes implement the same interfaces, and these interfaces may be inherited to create new collection classes that fit more specialized data storage needs.

THE ARRAYLIST:

ArrayList is one of the most flexible data structure from CSharp Collections. ArrayList contains a simple list of values. ArrayList implements the IList interface using an array and very easily we can add, insert, delete, view etc. It is very flexible because we can add without any size information, which is it will grow dynamically and also shrink.

```
Add : Add an Item in an ArrayList
Insert : Insert an Item in a specified position in an ArrayList
Remove : Remove an Item from ArrayList
RemoveAt: remove an item from a specified position
Sort : Sort Items in an ArrayList
```

How to add an Item in an ArrayList ?

```
Syntax : ArrayList.add(object)
object : The Item to be add the ArrayList
ArrayList arr;
arr.Add("Item1");
```

How to Insert an Item in an ArrayList ?

```
Syntax : ArrayList.insert(index,object)
index : The position of the item in an ArrayList
object : The Item to be add the ArrayList
ArrayList arr;
arr.Insert(3, "Item3");
```

How to remove an item from arrayList ?

```
Syntax : ArrayList.Remove(object)
object : The Item to be add the ArrayList
arr.Remove("item2")
```

How to remove an item in a specified position from an ArrayList ?

```
Syntax : ArrayList.RemoveAt(index)
index : the position of an item to remove from an ArrayList
ItemList.RemoveAt(2)
```

How to sort ArrayList ?

```
Syntax : ArrayList.Sort()
```

The System.Collections.ArrayList class is similar to arrays, but can store elements of any data type. We don't need to specify the size of the collection when using an ArrayList (as we used to do in the case of simple arrays). The size of the ArrayList grows dynamically as the number of elements it contains changes. An ArrayList uses an array internally and initializes its size with a default value

called Capacity. As the number of elements increase or decrease, ArrayList adjusts the capacity of the array accordingly by making a new array and copying the old values into it. The Size of the ArrayList is the total number of elements that are actually present in it while the Capacity is the number of elements the ArrayList can hold without instantiating a new array. An ArrayList can be constructed
Like this:

```
ArrayList list = new ArrayList();
```

We can also specify the initial Capacity of the ArrayList by passing an integer value to the constructor:

```
ArrayList list = new ArrayList(20);
```

PROGRAM:

```
static void Main()
{
        ArrayList list = new ArrayList();
        list.Add(45);
        list.Add(87);
        list.Add(12);
        foreach(int num in list)
       {
          Console.Write(num);
        }
}
```

```
Output :45 87 12
```

BITARRAY:

You have an array of boolean values or bits and want to store them in a compact and easy-to-use object in the C# programming language. The BitArray class in System.Collections offers an ideal and clear interface to bitwise operations, allowing you to perform bitwise operations, and count and display bits.

`Count` - Gets the number of elements contained in the BitArray.

```
BA.Count;
```

`Length` - Gets or sets the number of elements in the BitArray.

```
BA.Length;
```

`Set(position,value)` - Sets the bit at a specific position in the BitArray to the specified value.

```
BA.Set(2, false);
```

PROGRAM:

```
using System;
using System.Collections;
class Program
{
    static void Main()
    {
        BitArray BA = new BitArray(5);
        BA[0] = true;
        BA[1] = false;
        BA[2] = true;
        BA[3] = false;
        BA[4] = true;
        foreach (bool bit in BA)
        {
            Console.Write(bit+" ");
        }
    }
}
```

Output : true false true false true

HASHTABLE:

Hashtable in C# represents a collection of key/value pairs which maps keys to value. Any non-null object can be used as a key but a value can. We can retrieve items from hashTable to provide the key. Both keys and values are Objects. The commonly used functions in Hashtable are:

```
   Add          : To add a pair of value in HashTable
   ContainsKey: Check if a specified key exist or not
   ContainsValue: Check the specified Value exist in HashTable
  Remove        : Remove the specified Key and corresponding Value
```

Add : To add a pair of value in HashTable

```
  Syntax : HashTable.Add(Key,Value)
  Key : The Key value
  Value : The value of corresponding key
  Hashtable ht;
  ht.Add("1", "Sunday");
```

ContainsKey : Check if a specified key exist or not

```
  Synatx : bool HashTable.ContainsKey(key)
  Key         : The Key value for search in HahTable
  Returns : return true if item exist else false
  ht.Contains("1");
```

ContainsValue : Check the specified Value exist in HashTable

```
  Synatx : bool HashTable.ContainsValue(Value)
  Value : Search the specified Value in HashTable
  Returns : return true if item exist else false
  ht.ContainsValue("Sunday")
```

Remove : Remove the specified Key and corresponding Value

```
  Syntax : HashTable.Remove(Key)
  Key : The key of the element to remove
  ht.Remove("1");
```

PROGRAM:

```
Hashtable h = new Hashtable();
h.Add("key1", "value1");
h.Add("key2", "value2");
h.Add("key3", "value3");
foreach (DictionaryEntry s in h)
{
        Console.write(s.Key + "    -    " + s.Value );
}
```

THE QUEUE :

The Queue works like FIFO system, a first-in, first-out collection of Objects. Objects stored in a Queue are inserted at one end and removed from the other. The Queue provides additional insertion, extraction, and inspection operations. We can Enqueue (add) items in Queue and we can Dequeue (remove from Queue) or we can Peek (that is we will get the reference of first item) item from Queue. Queue accepts null reference as a valid value and allows duplicate elements.Some important functions in the Queue Class are follows:

```
  Enqueue : Add an Item in Queue
  Dequeue : Remove the oldest item from Queue
  Peek    : Get the reference of the oldest item
```

Enqueue : Add an Item in Queue

```
  Syntax : Queue.Enqueue(Object)
  Object : The item to add in Queue
  days.Enqueue("Sunday");
```

Dequeue : Remove the oldest item from Queue (we don't get the item later)

```
  Syntax : Object Queue.Dequeue()
  Returns : Remove the oldest item and return.
  days.Dequeue();
```

Peek : Get the reference of the oldest item (it is not removed permanently)

```
  Syntax : Object Queue.Peek()
  returns : Get the reference of the oldest item in the Queue
  days.peek();
```

PROGRAM:

```
using System;
using System.Collections;

class Program
{
    static void Main()
    {
        Queue queue = new Queue();
        queue.Enqueue(2);
        queue.Enqueue(4);
        queue.Enqueue(6);
        while (queue.Count != 0)
        {
            Console.Write(queue.Dequeue());
        }

    }
}
```

Output : 2 4 6

The output shows that the queue removes items in the order they were inserted. The other methods of a Queue are very similar to those of the ArrayList and Stack classes

THE SORTEDLIST:

The SortedList class represents a collection of key-and-value pairs that are sorted by the keys and are accessible by key and by index. A SortedList is a hybrid between a Hashtable and an Array. When an element is accessed by its key using the Item indexer property, it behaves like a Hashtable. When an element is accessed by its index using GetByIndex or SetByIndex, it behaves like an Array.

Properties are:

- **`Capacity`** - Gets or sets the capacity of a SortedList object.
- **`Count`** - Gets the number of elements contained in a SortedList object

Method is:

- **`Add(key,value)`** - Adds an element with the specified key and value to a SortedList object. The value can be a null reference.

If Count already equals Capacity, the capacity of the SortedList object is increased by automatically reallocating the internal array, and the existing elements are copied to the new array before the new element is added.

- A SortedList internally maintains two arrays to store the elements of the list; that is, one array for the keys and another array for the associated values. Each element is a key-and-value pair that can be accessed as a DictionaryEntry object. A key cannot be a null reference, but a value can be.

```
using System;
using System.Collections;
class Program
{
        static void Main()
        {
            SortedList sl = new SortedList();
                sl.Add(2, "Java");
                sl.Add(1, "C#");
                sl.Add(4, "VB.Net");
                sl.Add(3, "C++");
                 for(int i=0; i<sl.Count; i++)
                  {
                   Console.WriteLine("{0} \t {1}",
                   sl.GetKey(i), sl.GetByIndex(i));
                  }
        }
}
```

```
Output:      1     C#
             2     Java
             3     C++
             4     VB.Net
```

The program stores the names of different programming languages (in string form) using integer keys. Then the for loop is used to retrieve the keys and values contained in the SortedList (sl). Since this is a sorted list, the items are internally stored in a sorted order and when we retrieve these names by the GetKey() or the GetByIndex() method, we get a sorted list of items in output.

THE STACK:

The Stack class represents a last-in-first-out (LIFO) Stack of Objects. Stack follows the push-pop operations. That is we can Push (insert) Items into Stack and Pop (retrieve) it back. Stack is implemented as a circular buffer. 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. As elements are added to a Stack, the capacity is automatically increased as required through reallocation. Commonly used methods:

```
  Push : Add (Push) an item in the Stack data structure
  Pop  : Pop return the last Item from the Stack
  Contains: Check the object contains in the Stack
```

Push: Add (Push) an item in the Stack data structure

```
  Syntax : Stack.Push(Object)
```

```
Object : The item to be inserted.
Stack days = new Stack();
days.Push("Sunday");
```

Pop : Pop return the item last Item from the Stack

```
Syntax : Object Stack.Pop()
Object : Return the last object in the Stack
days.Pop();
```

Contains : Check the object contains in the Stack

```
Syntax : Stack.Contains(Object)
Object : The specified Object to be search
days.Contains("Tuesday");
```

PROGRAM:

```
using System;
using System.Collections;
class Test
 {
     static void Main()
       {
           Stack stack = new Stack();
           stack.Push(2);
           stack.Push(4);
           stack.Push(6);
           while (stack.Count != 0)
           {
               Console.Write(stack.Pop());
           }
       }
   }
```

Output: 6 4 2

Note that we have used a while () loop here to iterate through the elements of the stack. One thing to remember in the case of a stack is that the Pop() operation not only returns the element at the top of stack, but also removes the top element so the Count value will decrease with each Pop() operation.

WHAT ARE GENERICS?

Generics are used to help make the code in your software components much more reusable. They are a type of data structure that contains code that remains the same; however, the data type of the parameters can change with each use. Additionally, the usage within the data structure adapts to the different data type of the passed variables. In summary, a generic is a code template that can be applied to use the same code repeatedly. Each time the generic is used, it can be customized for different data types without needing to rewrite any of the internal code. While generics would be new, the functionality that is provided by them can be obtained in C# today. This functionality is done by using type casts and polymorphism. With generics, however, you can avoid the messy and intensive conversions from reference types to native types. Additionally, you can create routines that are much more type-safe.

Benefits of Generics

There are several advantages to using a generic class.Generics facilitate a strongly typed programming model, preventing data types other than those explicitly intended by the members within the parameterized class. Compile-time type checking reduces the likelihood of InvalidCastException type errors at runtime.

1. Using value types with generic class members no longer causes a cast to object; they no longer require a boxing operation. (For example, path.Pop() and path.Push() do not require an item to be boxed when added or unboxed when removed.)
2. Generics in C# reduce code bloat. Generic types retain the benefits of specific class versions, without the overhead. (For example, it is no longer necessary to define a class such as CellStack.)
3. Performance increases because casting from an object is no longer required, thus eliminating a type check operation. Also, performance increases because boxing is no longer necessary for value types.
4. Generics reduce memory consumption because the avoidance of boxing no longer consumes memory on the heap.
5. Code becomes more readable because of fewer casting checks and because of the need for fewer type-specific implementations.
6. Editors that assist coding via some type of IntelliSense work directly with return parameters from generic classes. There is no need to cast the return data for IntelliSense to work.

GENERIC LIST:

Using a Generic List in C# is an efficient method of storing a collection of variables. And probably the best part is that the List is strongly typed and casting (which degrades performance) will no longer be necessary. Your collection of variables should be of the same data type. Generic Lists were first introduced into the .NET Framework 2.0.

```
using System;
using System.Collections.Generic;
class Program
{
    static void Main(string[] args)
    {
        List<int> GL = new List<int>();
        GL.Add(99);
        GL.Add(10);
        GL.Add(22);
        GL.Add(55);
        foreach (int x in GL)
        {
            Console.Write( x+",");
        }

    }
}
```

Output is : 99,10,22,55

GENERIC STACK:

generic class is Stack<T> while T is the *parameter type* for our class. We sometimes used the *parametric polymorphism* term to talk about generics. In fact, our Stack<T> class can take several forms (Stack<int>, Stack<string> etc). It is then polymorphic and parameterized by one type.

```
using System;
using System.Collections.Generic;
class Test
{
    static void Main()
    {
        Stack<int> s = new Stack<int>();
        s.Push(2);
```

```
            s.Push(4);
            s.Push(6);
            while (s.Count != 0)
            {
                Console.Write(s.Pop()+",");
            }
        }
    }
```

```
Output : 6,4,2
```

GENERIC QUEUE:

The Queue is similar to the List, except it acts as the name implies. The first notable difference is the way in which objects are stored and removed from the Queue; Queues use a First-In-First-Out way of providing elements. To add an item to the back of the queue we use the 'Enqueue' method. To remove and retrieve an element from the front of a Queue we use the 'Dequeue' method:

```
using System;
using System.Collections.Generic;
class Program
{
    static void Main(string[] args)
    {
        Queue<double> GQ = new Queue<double>();
        GQ.Enqueue(5.51);
        GQ.Enqueue(4.32);
        GQ.Enqueue(6.55);
        GQ.Enqueue(9.21);
        double sum = 0.0;
        while (GQ.Count > 0)
        {
            double val = GQ.Dequeue();
            Console.WriteLine("   " + val);
            sum = sum + val;
        }
        Console.WriteLine("Sum is   :" + sum);
    }
}
```

```
Output :5.51 ,4.32,6.55,9.21
Sum is 25.59
```

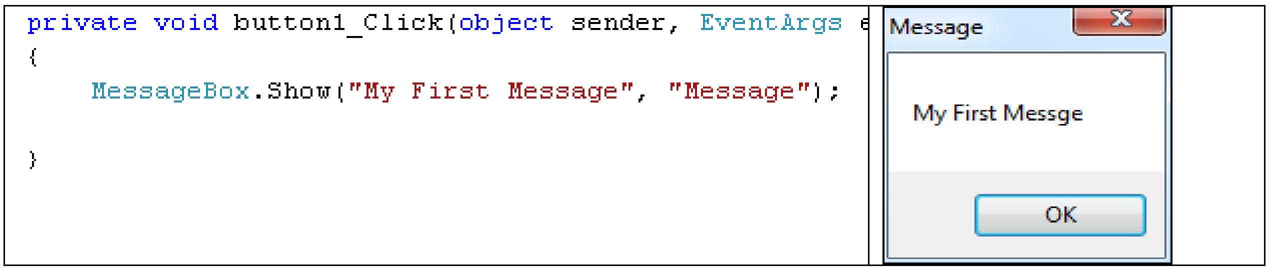
WINDOWS FORMS:

C# and .Net provide extensive support for building Windows Applications. The most important point about windows applications is that they are 'event driven'. All windows applications present a graphical interface to their users and respond to user interaction. This graphical user interface is called a 'Windows Form', or 'WinForm' for short. A windows form may contain text labels, push buttons, text boxes, list boxes, images, menus and vast range of other controls. In fact, a WinForm is also a windows control just like a text box, label, etc. In .Net, all windows controls are represented by base class objects contained in the System.Windows.Forms namespace.

MESSAGEBOX:

First, the MessageBox.Show method is a static method, which means you do not need to create a new MessageBox() anywhere in your code. Instead, you can simply type "MessageBox" and press the period, and then select Show.

PROGRAM:



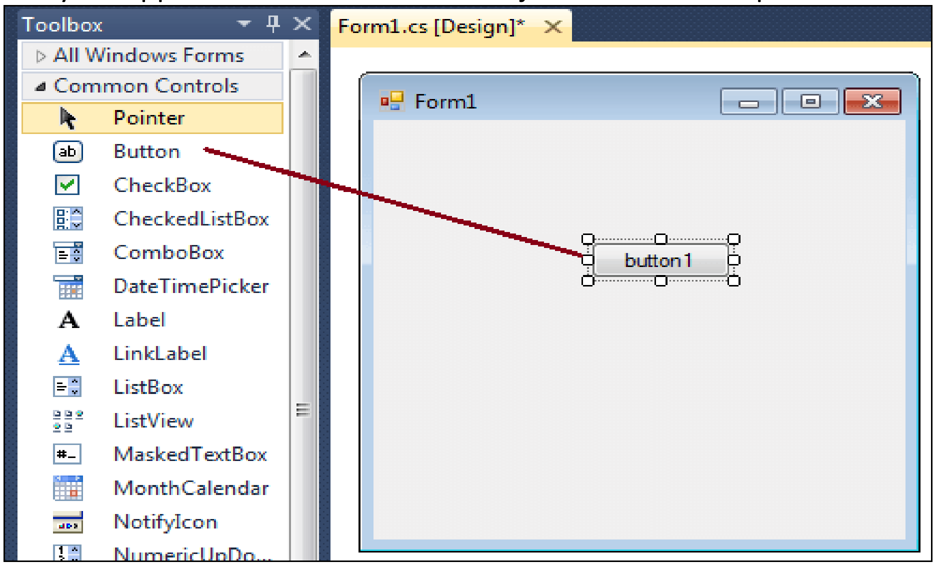
```
private void button1_Click(object sender, EventArgs e
{
    MessageBox.Show("My First Message", "Message");

}
```

```
Message
My First Messge
OK
```

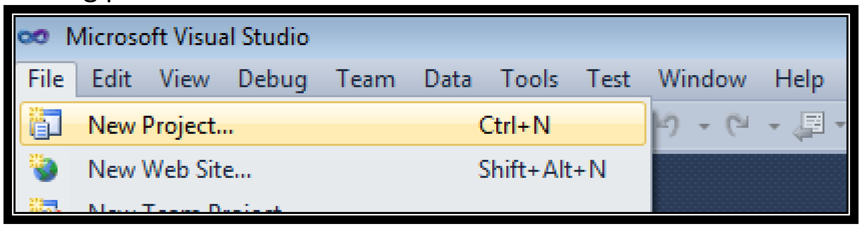
The first thing you will notice is that we use an enum called DialogResult. This is built into the C# libraries and, as you may have guessed, is used in catching dialog results. At this point hitting OK or the X button will both send an DialogResult.OK back, so this will capture either. What do we do if we want more that just one choice? Well, MessageBox can handle many things, and one of those is having more than one button in the dialog.

C# Windows Forms

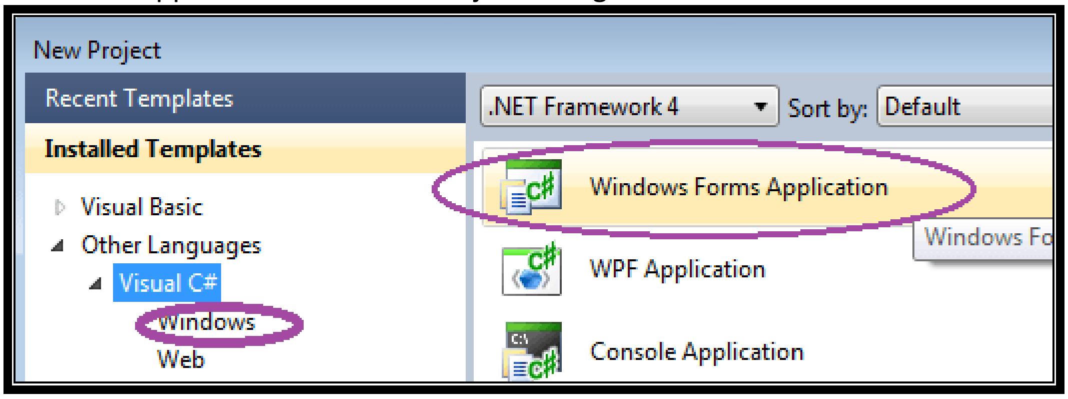
C# programmers have made extensive use of forms to build user interfaces. Each time you create a Windows application, Visual Studio will display a default blank form, onto which you can drag the controls onto your applications main form and adjust their size and position.



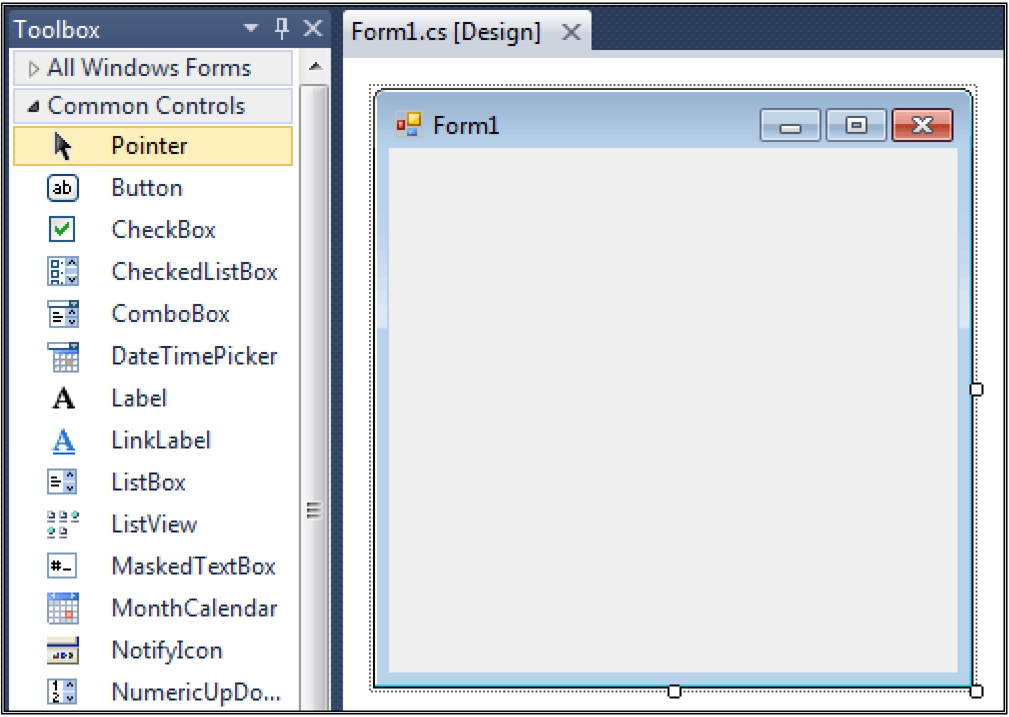
The first step is to start a new project and build a form. Open your Visual Studio and select File->New Project and from the new project dialog box select Other Languages->Visual C# and select Windows Forms Application. Enter a project name at the bottom of the dialouge box and click OK button. The following picture shows how to create a new Form in Visual Studio.


Microsoft Visual Studio
File Edit View Debug Team Data Tools Test Window Help
New Project... Ctrl+N
New Web Site... Shift+Alt+N

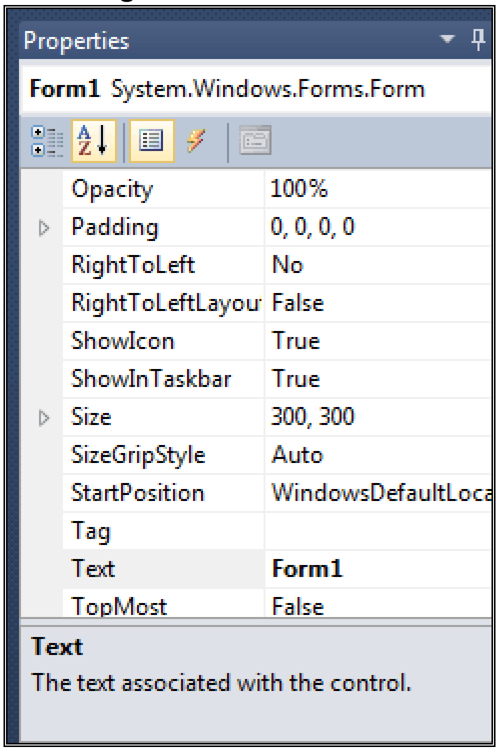
Select Windows Forms Application from New Project dialog box.



After selecting Windows Forms Application , you can see a default Form (Form1) in your new C# project. The Windows Form you see in Designer view is a visual representation of the window that will open when your application is opened. You can switch between this view and Code view at any time by right-clicking the design surface or code window and then clicking View Code or View Designer. The following picture shows how is the default Form (Form1) looks like.



At the top of the form there is a title bar which displays the forms title. Form1 is the default name, and you can change the name to your convenience . The title bar also includes the control box, which holds the minimize, maximize, and close buttons.
If you want to set any properties of the Form, you can use Visual Studio Property window to change it. If you do not see the Properties window, on the View menu, click Properties window. This window lists the properties of the currently selected Windows Form or control, and it's here that you can change the existing values.

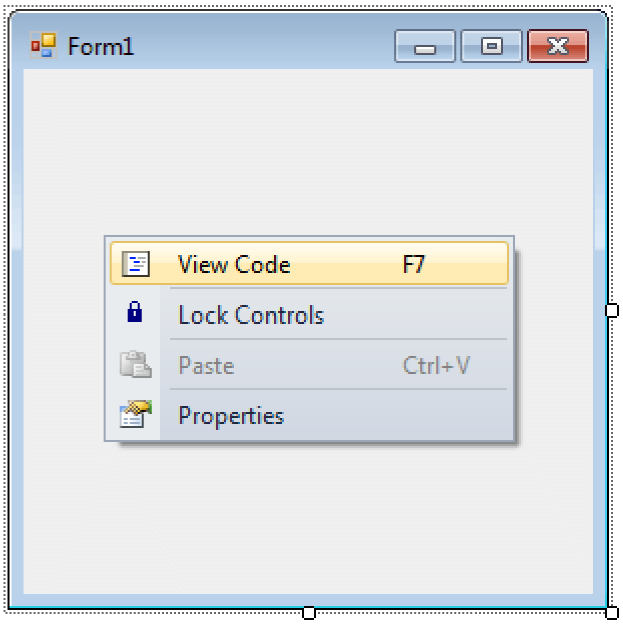

Properties

Form1 System.Windows.Forms.Form

Opacity	100%
Padding	0, 0, 0, 0
RightToLeft	No
RightToLeftLayou	False
ShowIcon	True
ShowInTaskbar	True
Size	300, 300
SizeGripStyle	Auto
StartPosition	WindowsDefaultLoca
Tag	
Text	**Form1**
TopMost	False

Text
The text associated with the control.

For example , to change the forms title from Form1 to MyForm, click on Form1 and move to the right side down Properties window, set Text property to MyForm. Then you can see the Title of the form is changed. Likewise you can set any properties of Form through Properties window.
You can also set the properties of the Form1 through coding. For coding, you should right-click the design surface or code window and then clicking View Code.



When you right click on Form then you will get code behind window, there you can write your code For example , if you want to change the back color of the form to Brown , you can code in the Form1_Load event like the following.

```
private void Form1_Load(object sender, EventArgs e)
{
    this.BackColor = Color.Brown;
}
```

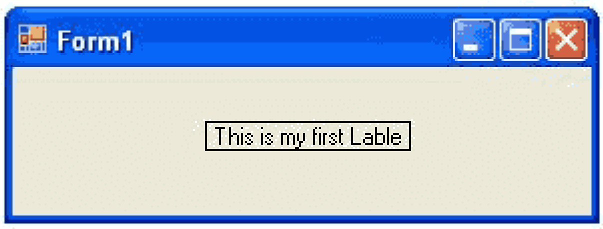
BUTTON:

The Button control can be used to initiate a function, confirm an entry, compare values and many other actions. The button can be clicked by the use of the mouse or if focused with the enter key or the space bar. The button can be disabled and only enabled under certain application conditions or changes.

The most important event of the Button control is the Click event. This event occurs when the user clicks the button. We can write the code for this event by double clicking on the button or double clicking in the cell next to click event in the properties window.

LABEL:

Labels are one of the most frequently used C# control. We can use the Label control to display text in a set location on the page. Label controls can also be used to add descriptive text to a Form to provide the user with helpful information. The Label class is defined in the System.Windows.Forms namespace.



Add a Label control to the form - Click Label in the Toolbox and drag it over the forms Designer and drop it in the desired location.

If you want to change the display text of the Label, you have to set a new text to the Text property of Label.

```
label1.Text = "This is my first Label";
```

TEXTBOX:

The TextBox control displays text entered at design time that can be edited by users at run time, or changed programmatically. Typically, a TextBox control is used to display a single line of text or allow a single line of text to be input by the user. It may also be set up to display/input multiline text with scrollbars, for use such as the notepad application. Optionally it may also be set up to display a password character instead of the actual text entered. There are other properties that allow the behaviour of the TextBox to be configured in several ways such as only entering uppercase or lowercase letters. A TextBox usually has a Label associated with it. This is usually positioned to the left or the top of the TextBox.

PROPERTIES:

Setting MultiLine

Clicking the smart tag icon on a TextBox control in the designer and ticking MultiLine enables the TextBox to have more than one line of text.

Using AutoComplete

Three of the properties allow the user to auto-complete text within a TextBox control. This can only happen when Multiline is set to false.

- *AutoCompleteMode is set to Suggest.*
- *AutoCompleteSource is set to CustomSource.*
- *AutoCompleteCustomSource is set to a list of the C# keywords.*

Masking characters as passwords

The properties PasswordChar and UseSystemPasswordChar can be used to create a TextBox that does not show the characters when they are types in. This will happen if either PasswordChar is set to a value or UseSystemPasswordChar is set to true. However, if both are set then the system password character will be used.

BASIC EVENTS FOR TEXTBOX:

Enter Event:

This event occurs when the TextBox gains focus. You can use this event if you want to notify the user before he starts entering data in the TextBox.

In the below example I have demonstrated how to use the TextBox's Enter event to Notify the user to enter a Age in the TextBox whenever the textbox gains focus.

```
private void textBox1_Enter(object sender, EventArgs e)
{
        MessageBox.Show("Must Enter Age");
}
```

TextChanged :

The event is fired whenever the value of the Text property of the TextBox control changes. It is also the Default event of the TextBox. We can use this event in scenarios where I want to make some event occur when the user is changing the text in the TextBox. In The Bellow Example I will show how the TextBox TextChanged event is used to identify the Text Being entered in the TextBox.

```
private void textBox1_TextChanged(object sender, EventArgs e)

{
        MessageBox.Show(textBox1.Text);
}

private void textBox1_TextChanged(object sender, EventArgs e)
{
        if (textBox1.Text != "")
        {
                Try
                {
                        int num = Convert.ToInt32(textBox1.Text);
                }
                catch (Exception ex)
                {
                        MessageBox.Show("Not A Number");
                        textBox1.Text = "";
                }
        }
}
```

KeyDown Event:

Here is the code that detects when Enter is pressed. The trick is to examine the KeyEventsArg e and check KeyCode. Keys is an enumeration, and it stores special values for many different keys pressed. KeyCode must be compared to Keys.

```
private void textBox1_KeyDown(object sender, KeyEventArgs e)
{

       if (e.KeyCode == Keys.Enter)
        {
           MessageBox.Show("You pressed enter! Good job!");
        }
        else if (e.KeyCode == Keys.Escape)
        {
           MessageBox.Show("You pressed escape! What's wrong?");
        }
}
```

Leave Event:

This event occurs when the TextBox loses focus and the TextBox is no longer active. You can use this event when you want to fire an event after the user leaves the textbox. In the below example I have demonstrated how to use the Leave event to check that the TextBox accepts a proper Age before leaving.

```
private void textBox1_Leave(object sender, EventArgs e)
{
if (Convert.ToInt32(textBox1.Text) < 18)
      {
            MessageBox.Show("Not A valid Age");
            textBox1.Focus();
      }
}
```

RICHTEXTBOX:

The Windows RichTextBox control allows users to enter and edit text using more advanced text formatting offered with the TextBox control. The RichTextBox control can load RFT format as well as TXT format files for reading or editing and provide methods for saving the edited files. The RichTextBox control can also be set to detect Url information and respond to the user clicking the Url. Some advanced RichText formatting include.

SelectionFont.
SelectionColor.
SelectionAlignment.
SelectionBackColor.
SelectionBullet.
SelectionHangingIndent.
SelectionProtected.

RADIOBUTTON:

The RadioButton control allows you to choose one item within the parent control or group of items. When a user selects one option within a group of options the other options clear automatically. Typically the options are on the line of Yes/No, Male/Female, True/False etc. The RadioButton is very similiar to the CheckBox control. The main difference is with the CheckBox control, the user can select multiple choices within the same group of options.

```
private void radioButton1_CheckedChanged(object sender, EventArgs e)
{
   if (radioButton1.Checked)
```

```
      {
        MessageBox.Show("checked");
       }
 }
```

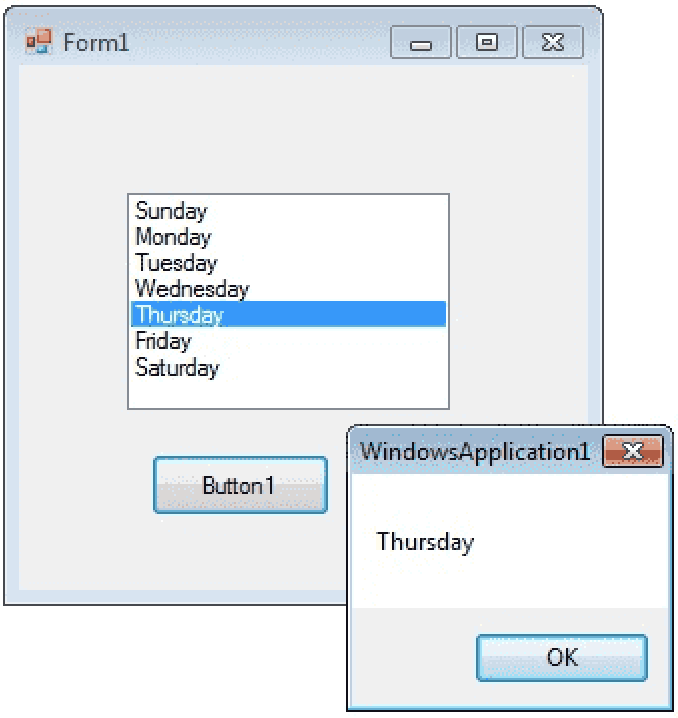
CHECKBOX:

The CheckBox control allows the user to choose or tick all that apply type questions from a list of options. The RadioButton is similar, but where the RadioButton only allows you to choose one item within the parent control as Yes/No, Male/Female, True/False etc, the CheckBox cotrol allows the user to choose from a combination of options.

```
private void checkBox1_CheckedChanged(object sender, EventArgs e)
{
     if(checkBox1.Checked)
     {
          MessageBox.Show("checked");
     }
}
```

LISTBOX:

The System.Windows.Forms.ListBox control enabled you to display a list of items from which the user can select one or multiple items from the list. The default is for the ListBox to list the items vertically however setting the multicolumn property to true the items are listed horizontally. The control is typically used to display data from a database. The items listed within the ListBox cannot be edited directly.



Thursday

Some commonly used properties of the ListBox control are:

SelectedItem	Contains the selected item from the listbox
SelectedIndex:	Zero based index of the selected item in the listbox.
Items:	Collection of all the items in the listbox. Used for adding or removing items from the list using the Add() and Remove() methods.
Sorted:	Indicates if the items in the list are sorted or not.
SelectionMode:	None - no item can be selected.

	One - only one item can be selected. MultiSimple -multiple items can be selected by clicking them with mouse. MultiExtended -multiple items can be selected using the Ctrl, Shift and arrow keys.

PICTUREBOX:

You want to use the PictureBox control in your Windows Forms program using the .NET Framework and C# language. The PictureBox control provides a rectangular region where you can draw an image; is supports many image formats, has an adjustable size, can access image files from your disk or from the Internet; and can resize images in several different ways.

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

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

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

There are five different PictureBoxSizeMode is available to PictureBox control.

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

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

SCROLLBAR:

To create a scroll bar control, on the Toolbox, you can click either the VScrollBar or the HScrollBar button then click a container. The scroll bar is one of the earliest controls of the Microsoft Windows operating system. Each control is implemented through a class of the same name. The VScrollBar and the HScrollBar classes are based on the ScrollBar class that essentially provides all of their functionalities.

The Minimum and Maximum Positions:

When using a scroll bar, the user can navigate from one end of the control to the other end. These are the control's minimum and maximum values represented respectively by the Minimum and the Maximum properties.

The Value of a Scroll Bar:

The primary technique the user applies to a scrollbar is to click one of the arrows at the ends of the control. As the bar slides inside of the control, it assumes an integer position from Minimum to Maximum. At one time, the position that the bar has is the Value property.

The Small Change:

When the user clicks an arrow of a scrollbar, the bar slides by one unit. This unit is represented by the SmallChange property and is set to 1 by default. If you want the bar to slide more than one unit, change the value of the SmallChange property to a natural number between Minimum and Maximum.

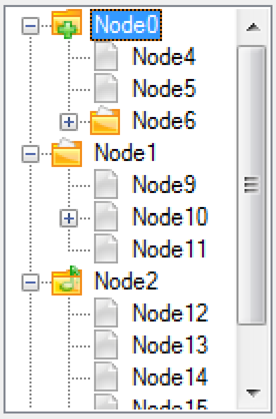
The Large Change:

When the user clicks in the scrolling region or presses the Page Up or Page Down keys, the bar would jump by LargeChange up to the scrolling amount value. You can also change the LargeChange property programmatically.

TREEVIEW:

You want to get started using a **TreeView control** in the .NET Framework and C# programming language. The TreeView control must have nodes added to it through the Nodes instance collection and can be inserted in the Visual Studio designer. It allows you to represent hierarchal text and icon data in a convenient and easy-to-deploy way.

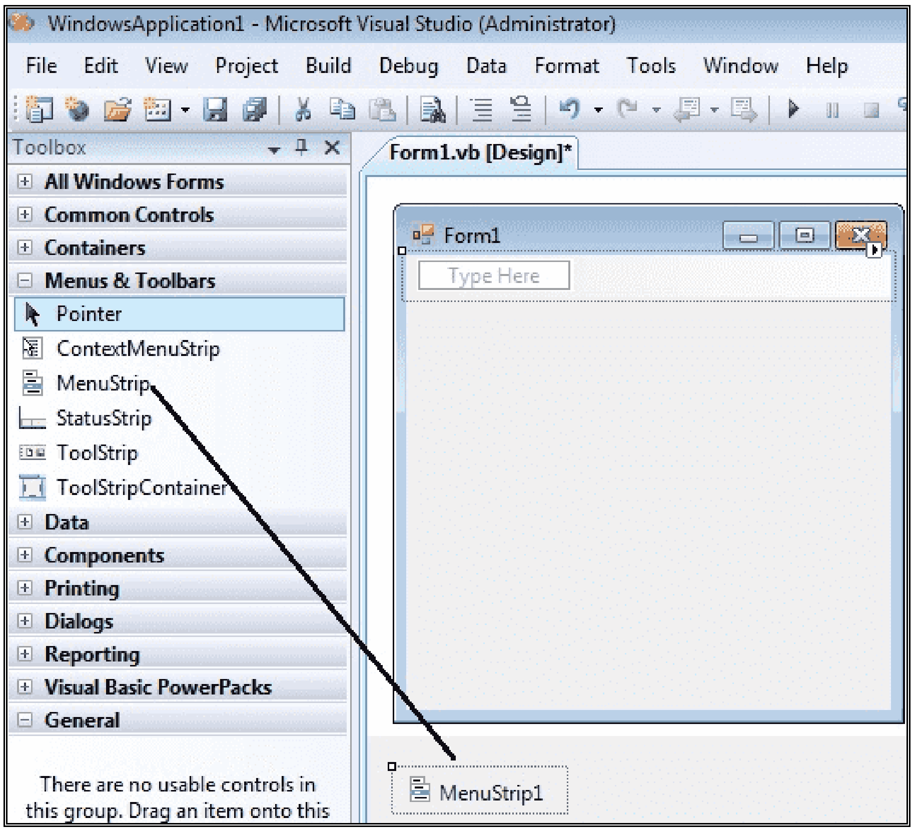
```
TreeNode treeNode = new TreeNode("Windows");
treeView1.Nodes.Add(treeNode);
treeNode = new TreeNode("Linux");
treeView1.Nodes.Add(treeNode);
```



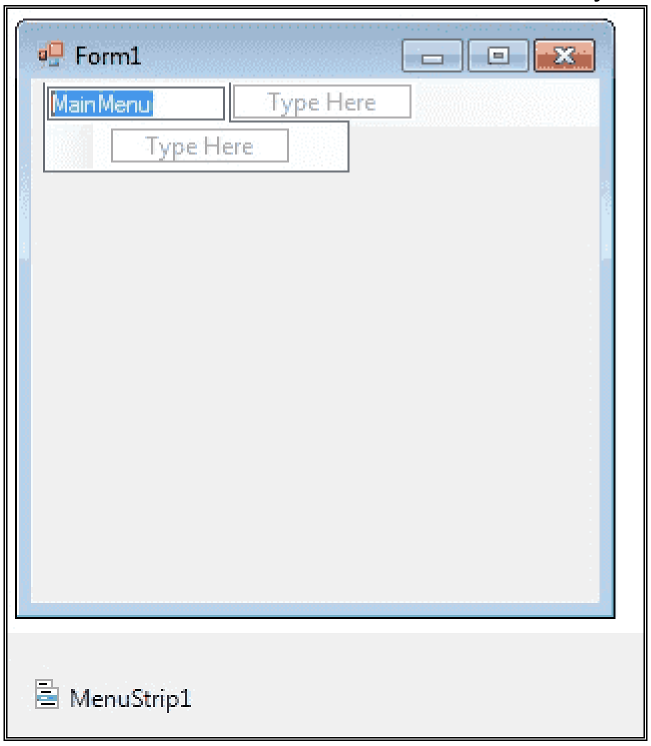
MENU

A Menu on a Windows Form is created with a MainMenu object, which is a collection of MenuItem objects. MainMenu is the container for the Menu structure of the form and menus are made of MenuItem objects that represent individual parts of a menu.

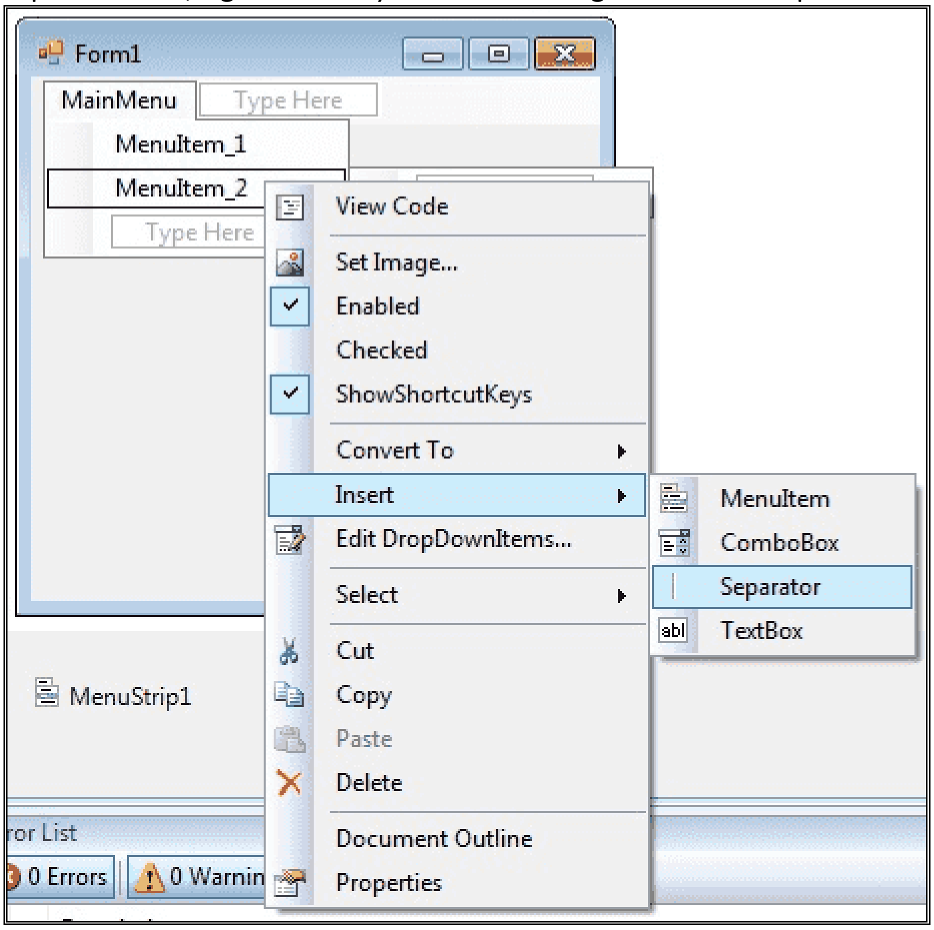
You can add menus to Windows Forms at design time by adding the MainMenu component and then appending menu items to it using the Menu Designer.



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



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



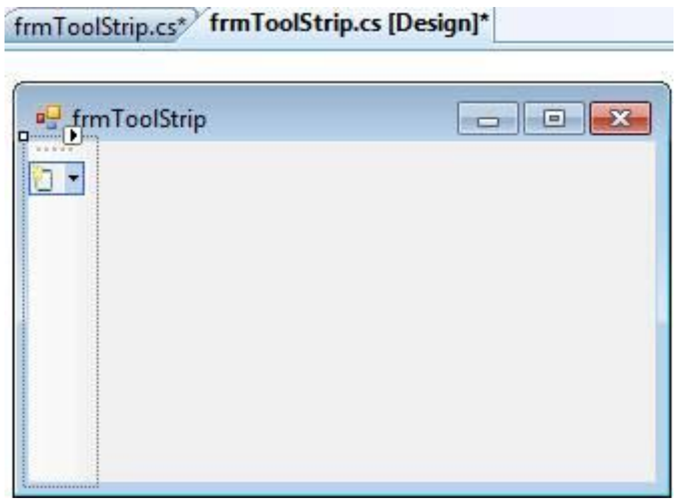
After creating the Menu on the form , you have to double click on each menu item and write the programs there depends on your requirements. The following C# program shows how to show a messagebox when clicking a Menu item.

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

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void menu1ToolStripMenuItem_Click(object sender, EventArgs e)
        {
            MessageBox.Show("You are selected MenuItem_1");
        }
    }
}
```

TOOLSTRIP:

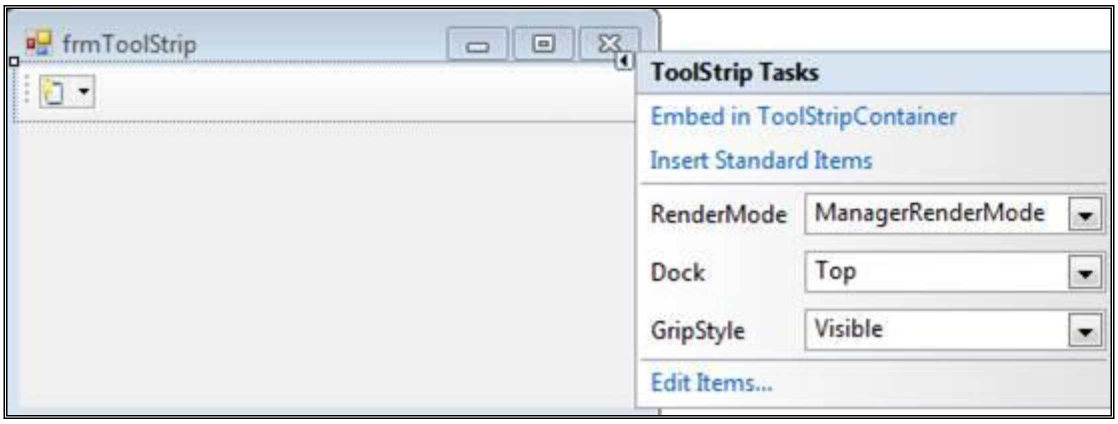
ToolStrip is a container for ToolStripItem elements. Each individual element on the ToolStrip is a ToolStripItem that manages the layout and event model for the type it contains. The ToolStrip controls provide a common interface for Menus and Strips in Windows Forms.

Drag and drop ToolStrip Control from toolbox on the window Form.

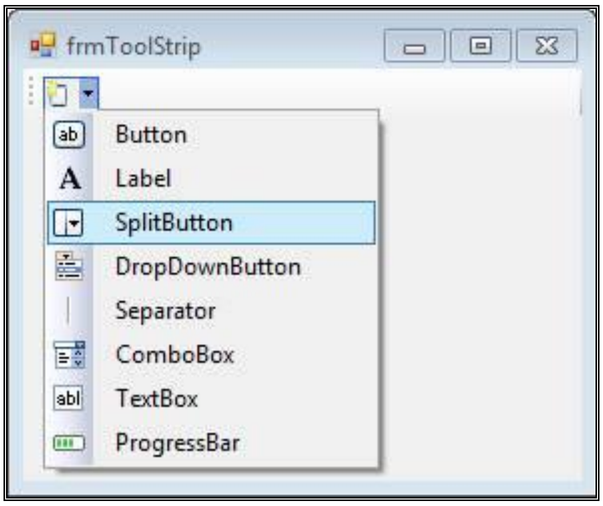


Add ToolStrip Item which you want to show. Add one of the items in your ToolStrip that derives from ToolStrip Item.

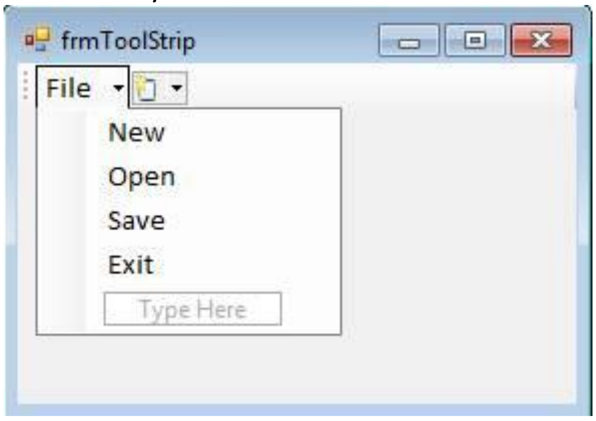
You can also add standard item through smart tag.



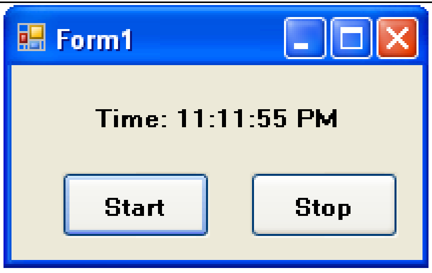
RenderMode: ManagerRenderMode
Dock: Top
GripStyle: Visible



Here SplitButton is added
Enter text which you want to show as Menu.



TIMER:

	The timer is a useful class that can be used to measure time elapsed between two events. The timer can be started by setting the .Enabled property to true or by using the Start() method. Likewise it can also be turned off by setting the .Enabled property to false or by using the Stop() method.

```
private void timer1_Tick(object sender, EventArgs e)
        {
            label1.Text = (string.Format("Time: {0}\n",
DateTime.Now.ToLongTimeString()));
        }
        private void BtnStart_Click(object sender, EventArgs e)
        {
            timer1.Start();
        }
        private void BtnStop_Click(object sender, EventArgs e)
        {
            timer1.Start();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            timer1.Interval = 1000;
        }
```

PANEL:

The Panel control is a container for other controls, customarily used to group related controls. Panels are used to subdivide a form by function, giving the user a logical visual cue of control grouping.

You can add controls to the Panel at design time and runtime. When you move the Panel control, all of its contained controls move too. The Panel control is displayed by default without any borders or scroll bars.

Docking a Panel in a Form

A Panel can be docked into the parent container, by selecting the Panel and clicking on the Smart Tag icon. This causes a small menu to appear, simply click on Dock in parent container.

To undock the Panel, click on the Smart Tag icon and select Undock in parent container.

GROUPBOX:

When designing user interfaces in Windows Forms, you can use the GroupBox control to create a square shape where you can place other controls. This creates an important level of visual continuity, and helps makes your programs easier to use.

Docking the GroupBox:

Top, Bottom, Left, Right

These will push the control to the selected edge.

Center: This will expand the control to fill the entire window.

None: The default; will disable docking.

DIALOGBOX:

ColorDialogBox:

To provide the selection of colors on Microsoft Windows applications, the operating system provides a common dialog box appropriate for such tasks. You can use the Color dialog box for various reasons such as letting the user set or change a color of an object or specifying the background color of a control or the color used to paint an object.

```
DialogResult result = colorDialog1.ShowDialog();
if (result == DialogResult.OK)
 {
     this.BackColor = colorDialog1.Color;
 }
```

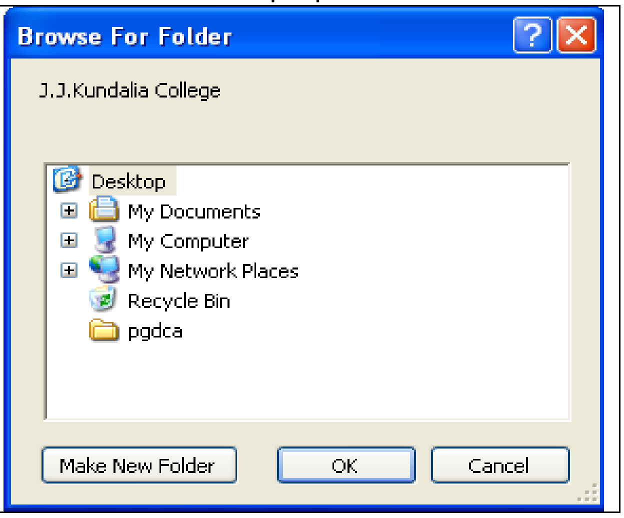
In this code, we show the ColorDialog instance, and then see if the user clicked the OK button to accept. If OK was clicked, the parent form's background color is set to the color selected in the dialog.

FolderBrowserDialogBox

A FolderBrowserDialog control is used to browse and select a folder on a computer. A typical FolderBrowserDialog looks like Figure 1 where you can see Windows Explorer like features to navigate through folders and select a folder.

We can create a FolderBrowserDialog control using a Forms designer at design-time or using the FolderBrowserDialog class in code at run-time (also known as dynamically). Unlike other Windows Forms controls, a FolderBrowserDialog does not have and not need visual properties like others.

```
DialogResult result =
folderBrowserDialog1.ShowDialog();
if (result == DialogResult.OK)
{
 string[] files =
Directory.GetFiles(folderBrowserDialog
1.SelectedPath);
     MessageBox.Show("Files found: " +
files.Length.ToString(), "Message");
                 textBox1.Text =
folderBrowserDialog1.SelectedPath;
Environment.SpecialFolder root =
folderBrowserDialog1.RootFolder;
```



FontDialogBox:

To assist the user with selecting a font for an application, Microsoft Windows provides the Font dialog box:

```
if ( fontDialog.ShowDialog() != DialogResult.Cancel )
{
   textBox1.Font = fontDialog.Font;
}
```

OpenFileDialog

You need to open a file specified by the user in your C# program targeting the .NET Framework, showing a dialog box. The Windows Forms platform has an excellent OpenFileDialog control which allows you to display the standard Windows dialog box, and then you can read in the results in your C# code.

```
string s;
DialogResult result = openFileDialog1.ShowDialog();
    if (result == DialogResult.OK) // Test result.
     {
         s = openFileDialog1.FileName;
         MessageBox.Show(result.ToString());
         string ss = File.ReadAllText(s);
         textBox1.Text = ss;
    }
```

PROGRESSBAR:

The ProgressBar control in Windows Forms provides a simple and visual way to indicate how far along an operation is. Often, you will use the ProgressBar with a long-running task such as one that is best implemented with a BackgroundWorker.

One of the useful properties on the ProgressBar is the **ForeColor property**. This can be set to a Color value to change the color of the bar itself that is drawn on the screen.

The Minimum and Maximum:

To show its effect, the progress bar draws its small rectangles as a bar. These small rectangles are from a starting position to an ending position. This means that the progress bar uses a range of values. This range is controlled by the Minimum and the Maximum properties whose default values are 0 and 100 respectively.

The Value of a Progress Bar:

At one particular time, the most right rectangle of a progress bar is referred to as its position and it is represented by the Value property. At design time, to set a specific position for the control, change the value of the Value property whose default is 0.

ProgressBar.Style property: The ProgressBar.Style property is based on the ProgressBarStyle enumeration that has three members:

- ***Blocks:*** With this value, the progress bar draws small distinct and adjacent rectangles:
- ***Continuous:*** The progress bar appears smooth as the same rectangles are not separated:
- ***Marquee:*** The progress bar shows an animation that consists of drawing a group of small rectangles that keep moving from left to right and restart.

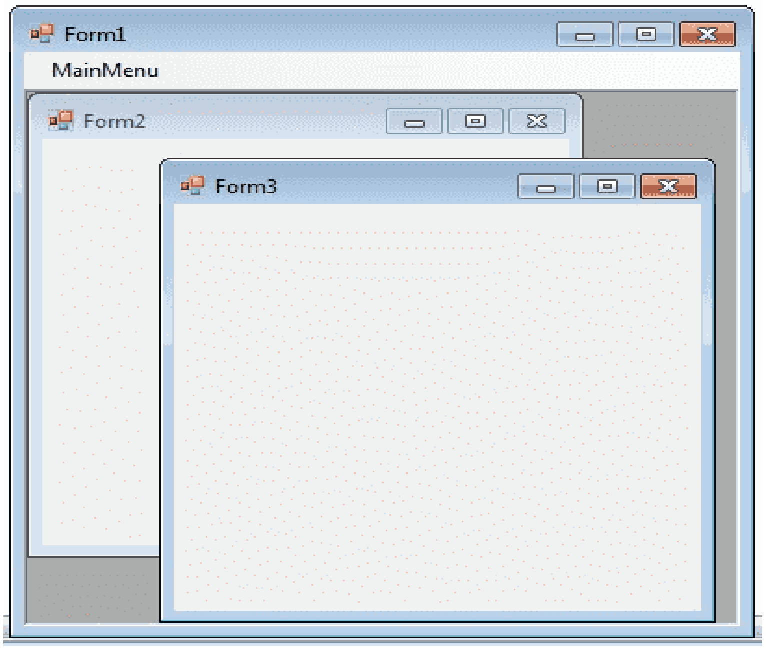
If you decide to use the Marquee style, the progress bar will continually show its animation of small rectangles moving at a speed set from the MarqueeAnimationSpeed integral property and you cannot get the value of the progress bar.

The Step Value of a Progress Bar

The number of units that the object must increase its value to is controlled by the Step property. By default, it is set to 10. Otherwise, you can set it to a different value of your choice.

MDI(Multiple Document Interface):

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



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

```
IsMdiContainer = true;
```

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

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

```
form.MdiParent = this;
form.Dock=DockStyle.Fill;
form.Show();
```

Introduction:

Data Access is the main part of any application that you develop. More concern is given on how you store the data and manipulate the data. Any web based application or window based applications stores the data for accessing the data for different purpose.

ADO.NET is a group of libraries provided by .NET framework which are used to create powerful database using various sources such as MS SQL, Microsoft Access, Oracle, XML, etc. it relies on various these classes to process requests and performs transition between a database system and the user. It provides classes for connecting with a data source, submitting queries, and processing results.

Thus ADO.NET is a large set of .NET classes which enables us to retrieve and manipulate data, and update data sources in many different ways.

Advantages Of Ado.Net:

1. **Connection with any database :**

ADO.NET can be used to connect with any type of database such as SQL Server, Oracle, and Access MySql etc. it can also connect with any third party database.

2. **Connected and Disconnected Architecture :**

It supports both connected and disconnected architecture. Connected architecture is supported by DAO, ADO, RDO or ODBC, disconnected is new feature provided by ADO.NET. you will study about the disconnected architecture in the future topics.

3. **XML Support :**

ADO.NET provides the new feature of XML supported, today XML is used widely for transfer data globally. XML is standard data transfer features for any type of data transfer. Using ADO.NET you can easily convert data from DataSet to XML and perform operation in secure and fast way.

4. **Fast, Scalable and Standard:**

ADO.NET is quite fats then other applications. It is quite scalable i.e. it can be expected. It also provides XML and other features thus it is quite standard used.

5. **Cross language Support:**

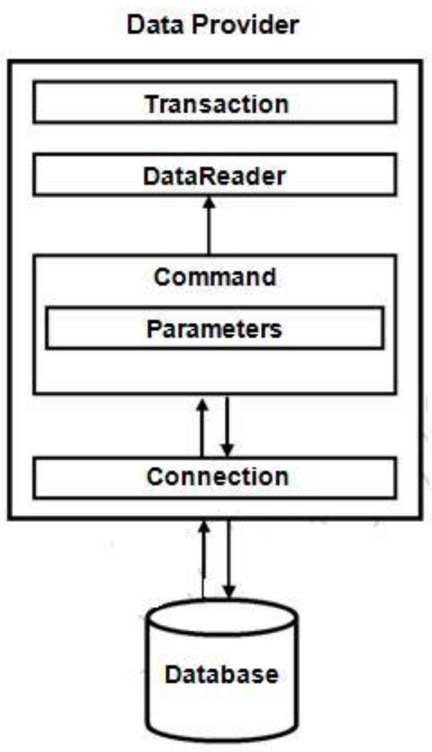
ADO.NET provides supports for many languages. You can use VB, C# or J# for ADO.NET programming. So it provides cross language support features.

Architecture Of Ado.Net (Connected and Disconnected):

ADO.NET is a data access technology from Microsoft .Net Framework, which provides communication between relational and non-relational systems through a common set of components. ADO.NET consists of a set of Objects that expose data access services to the .NET environment. ADO.NET is designed to be easy to use, and Visual Studio provides several wizards and other features that you can use to generate ADO.NET data access code. ADO.NET provides two types of architecture for accessing data via database: Connected and Disconnected Architecture .

Connected Architecture:

The architecture of ADO.NET in which connection to the database must be opened to access the data retrieved from database is called connected architecture. It is built on the classes called connection, command, datareader and transaction.



Components of Connected Architecture:

Connection: in connected architecture also the purpose of connection is to just establish a connection to database and itself will not transfer any data.

DataReader : DataReader is used to store the data retrieved by command object and make it available for .net application. Data in DataReader is read only and within the DataReader you can navigate only in forward direction and it also only one record at a time.

Command: command object is used to execute SQL queries against database. Using command object you can execute select, insert, update and delete SQL command.

Transaction: enables you to get the list of commands in transactions at the data source.

DISCONNECTED ARCHITECTURE:

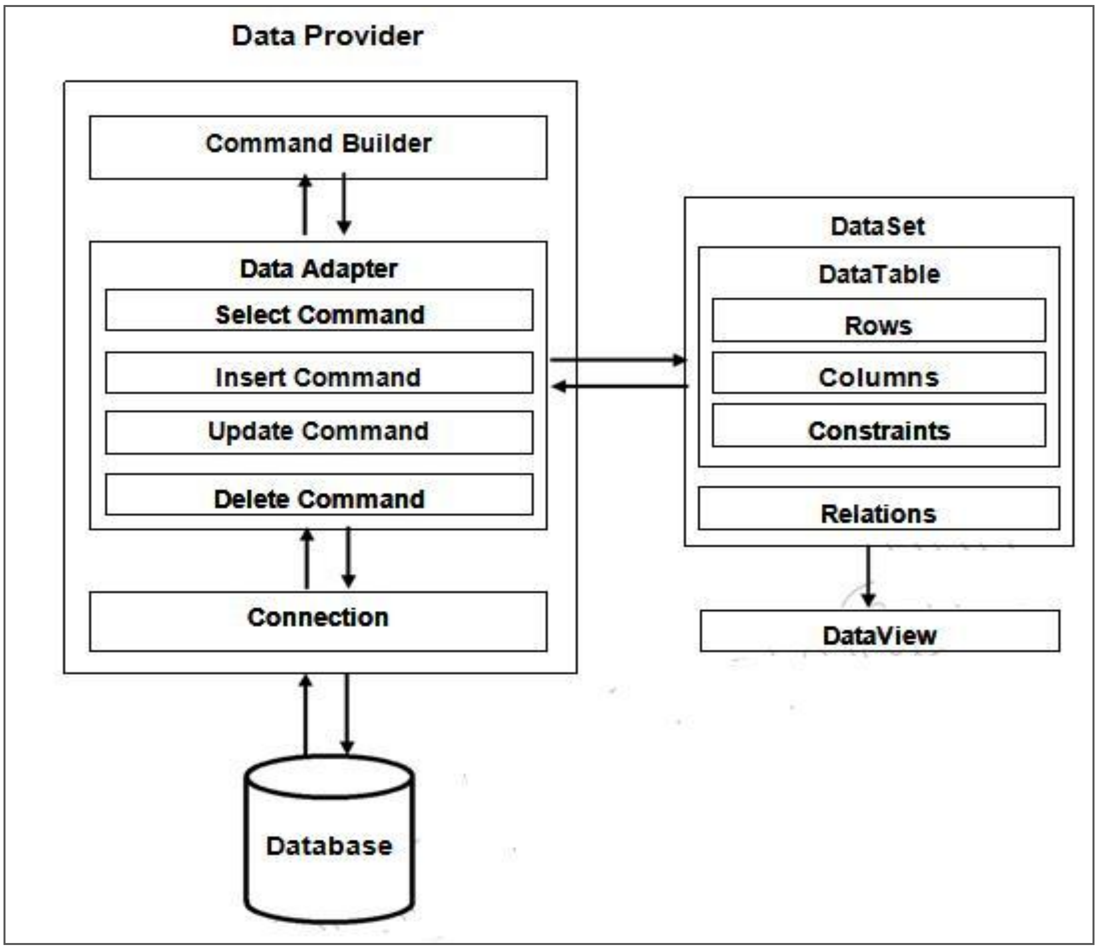
The architecture of ADO.net in which data retrieved from database can be accessed even when connection to database was closed is called as disconnected architecture. Disconnected architecture of ADO.net was built on classes connection, dataadapter, commandbuilder and dataset and dataview.

In disconnected architecture, the copy of the data is loaded from database and kept in local memory of client. Database is only connected during the copying of the database. After copying the databse the databse the connection to the databse is lost. You can perform insert, update, or select operation on the local of the database. The updated database is in our local memory only. Now when you connect to database again memory only. Now when you connect to database again then the local copy of the database is copied into the final database. In this way data consistency is maintained.

Components of DisConnected Architecture:

Connection : Connection object is used to establish a connection to database and connection it self will not transfer any data.

DataAdapter : DataAdapter is used to transfer the data between database and dataset. It has commands like select, insert, update and delete. Select command is used to retrieve data from database and insert, update and delete commands are used to send changes to the data in dataset to database. It needs a connection to transfer the data.



CommandBuilder : by default dataadapter contains only the select command and it doesn't contain insert, update and delete commands. To create insert, update and delete commands for the dataadapter, commandbuilder is used. It is used only to create these commands for the dataadapter and has no other purpose.

DataSet : Dataset is used to store the data retrieved from database by dataadapter and make it available for .net application.

To fill data in to dataset **fill()** method of dataadapter is used and has the following syntax.

Da.Fill(Ds);

When fill method was called, dataadapter will open a connection to database, executes select command, stores the data retrieved by select command in to dataset and immediately closes the connection.

DIFFERENCE BETWEEN CONNECTED AND DISCONNECTED ARCHITECTURE:

NO	CONNECTED ARCHITECTURE:	DISCONNECTED ARCHITECTURE:
1.	It is connection oriented architecture.	It is connection less architecture.
2.	It uses DataReader for communication between Database and application.	It uses datasets for communication between database and application.

3.	It gives faster performances the application is always connected with the database.	It gives low speed and performance as the database is not connected with the database at all time.
4.	Connected architecture can work with a single table at a time.	Disconnected architecture can work with multiple tables at a time.
5.	DataReader cannot persist the data.	DataSet can persist the data.
6.	It is only used Read only mode, you cannot update the data.	In this we can update the data, as the data is stored locally.
7.	More traffic would be creating if the number of users increases on the server side.	In this there is no concern on the traffic as the data is copied locally.
8.	Much memory of the server is used while processing connected architecture	Much memory of server is used once only when you copy the data into local memory, after that local memory of each user would be used.

DATA PROVIDERS IN ADO.NET

A set of libraries that is used to communicate with data source. Eg: SQL data provider for SQL, Oracle data provider for Oracle, OLE DB data provider for access, excel or mysql. Data providers are the one which does the work of communication between application and database. They are responsible for maintaining the connection to the database. Different classes are provided these data providers which help to perform different database operations either in connected or disconnected modes.

A data provider is a set of classes provided by ADO.NET that allows you to access a specific database, execute SQL commends, and retrieve data providers which allows you to do operations on any of databases. For different types of database, different types of data providers are provided by ADO.NET.

There are 4 types of data providers provided by Microsoft:

SQL Server Provider:

This provider is designed especially to work SQL server database.

Oracle Provider:

This provider deals with oracle database.

OLEDB provider:

This provider works with any database that has OLE DB driver. It provides a means to access global data. It can access any data from any type of database by using OLEDB driver.

ODBC provider:

This provider deals with any database which has ODBC driver. Almost all databases today come with ODBC driver inbuilt. Thus you can connect with any database even through drivers are not provided.

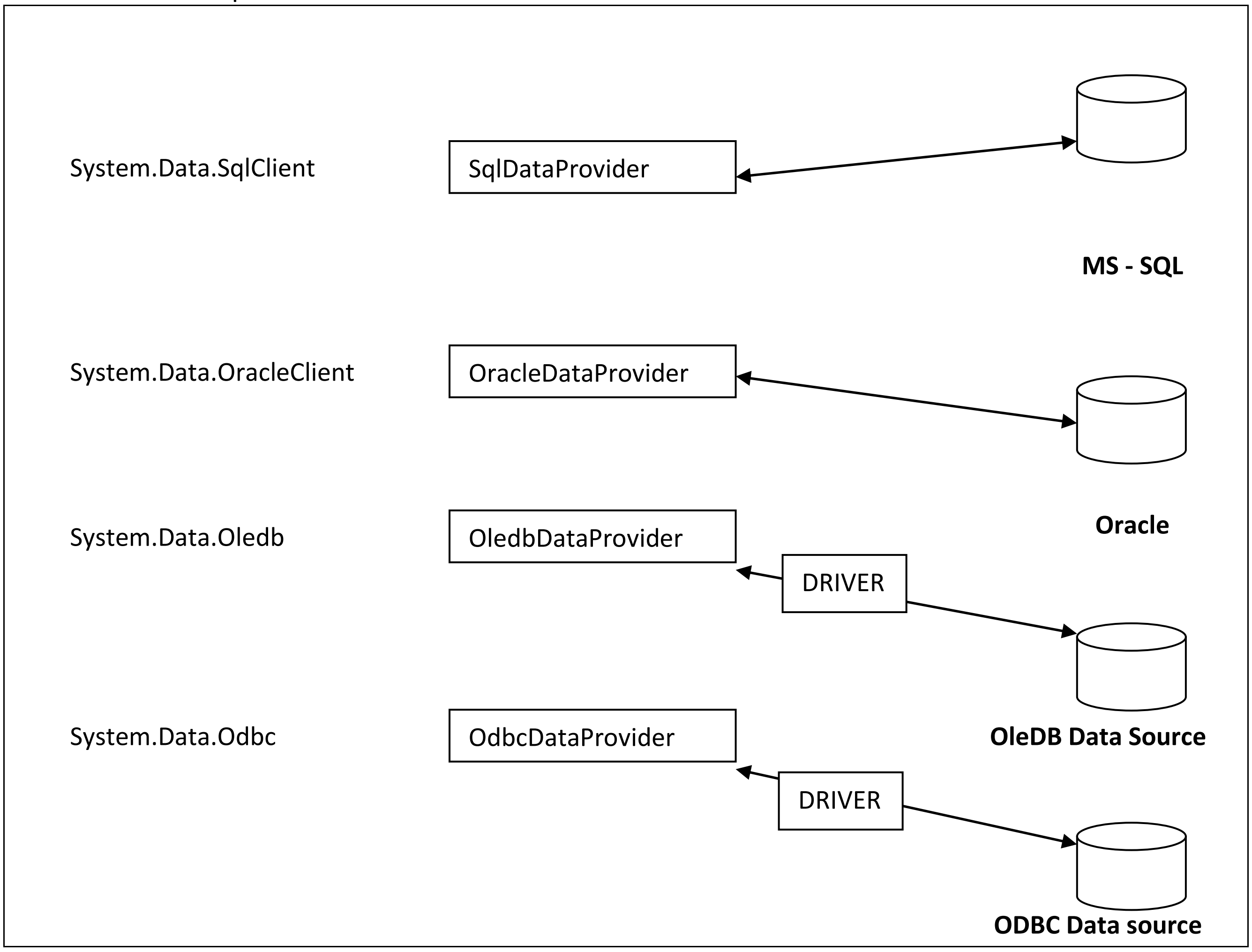
Here in the below diagram you can see SqldataProvider and OracledataProvider directly communicate with your database. It means if you are sure that you want to connect to SQL or Oracle then you can use this provider to get good performance.

For OleDbProvider and OdbcProvider, both require a driver between ADO.Net and your actual database. It is as these providers are meant to work any type of database so you require a driver to which works as a connection between both.

All these data providers provide common component which makes syntax easy to remember even though you change the provider.

The four main components provided by these data providers are:

1. **Connection**: Used for establishing connection with database.
2. **DataReader:** Used to read the data from the database.
3. **Command:** Used to specify the SQL query for retrieving data from the database.
4. **DataAdapter :** Used to fetch the data from database to local memory of client (i.e. DataSet) and also update the back to database.



DIFFERENCES BETWEEN DATASET AND DATAREADER

DATASET	DATAREADER
It is disconnected object and can provide access to data even when connection to database was closed.	It is connected object and cannot provide access to data when connection to database was closed.
It can store data from multiple tables.	It can store data from only one table.
It allows insert, update and delete on data	It is read only and it doesn't allow insert, update and delete on data.
It allows navigation between record either forward or backward.	It allows only forward navigation that also only to immediate next record.
It can contain multiple records.	It can contain only one record at a time.
All the data of a dataset will be on client system.	All the data of a DataReader will be on server and one record at a time is retrieved and stored in datareader when you call the Read() method of datareader.

COMMAND OBJECT IN ADO.NET

Command is used to execute almost any SQL command from within the .net application. The SQL command like insert, update, delete, select, create, alter, drop can be executed with command object and you can also call stored procedures with the command object. Command object has the following important properties.

- **Connection :** used to specify the connection to be used by the command object.
- **CommandType :** Used to specify the type of SQL command you want to execute. To assign a value to this property, use the enumeration CommandType that has the members Text, StoredProcedure and TableDirect. Text is the default and is set when you want to execute ant SQL command with command object. StoredProcedure is set when you want to call a stored procedure or function and TableDirect is set when you want to retrieve data from the table directly by specifying the table name without writing a select statement.
- **CommandText :** Used to specify the SQL statement you want to execute.
- **Transaction :** Used to associate a transaction object to the command object so that the changes made to the database with command object can be committed or rollback.

Command object has the following important methods.

- **ExecuteNonQuery() :** Used to execute an SQL statement that doesn't return any value like insert, update and delete. Return type of this method is int and it returns the no. of rows effected by the given statement.
- **ExecuteScalar() :** Used to execute an SQL statement and return a single value. When the select statement executed by executescalar() method returns a row and multiple rows, then the method will return the value of first column of first row returned by the query. Return type of this method is object.
- **ExecuteReader() :** Used to execute a select a statement and return the rows returned by the select statement as a DataReader. Return type of this method is DataReader.

CONNECTION OBJECT IN ADO.NET

The Connection Object is a part of ADO.NET Data Provider and it is a unique session with the Data Source. In .Net Framework the Connection Object is Handling the part of physical communication between the application and the Data Source. Depends on the parameter specified in the Connection String , ADO.NET Connection Object connect to the specified Database and open a connection between the application and the Database . When the connection is established, SQL Commands may be executed, with the help of the Connection Object, to retrieve or manipulate data in the Database. Once the Database activity is over , Connection should be closed and release the resources .

Some useful properties:

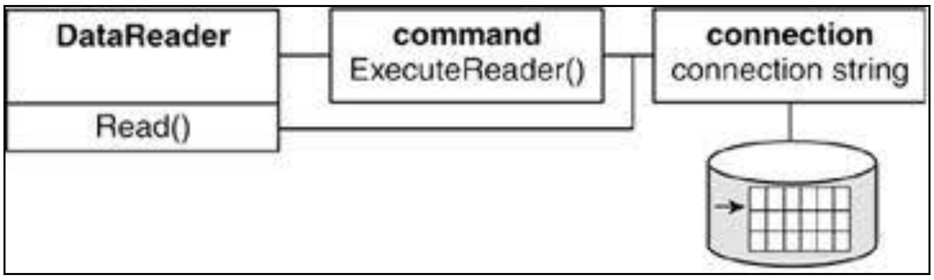
Property	Description
ConnectionString	Sets or returns the details used to create a connection to a data source
ConnectionTimeout	Sets or returns the number of seconds to wait for a connection to open
Provider	Sets or returns the provider name
State	Returns a value describing if the connection is open or closed
Version	Returns the ADO version number

Some useful methods:

Method	Description
BeginTrans	Begins a new transaction
Cancel	Cancels an execution
Close	Closes a connection
CommitTrans	Saves any changes and ends the current transaction
Execute	Executes a query, statement, procedure or provider specific text
Open	Opens a connection
OpenSchema	Returns schema information from the provider about the data source

DATAREADER:

the ADO.NET DataReader to retrieve a read-only, forward-only stream of data from a database. Results are returned as the query executes, and are stored in the network buffer on the client until you request them using the Read method of the DataReader. Using the DataReader can increase application performance both by retrieving data as soon as it is available, and (by default) storing only one row at a time in memory, reducing system overhead.



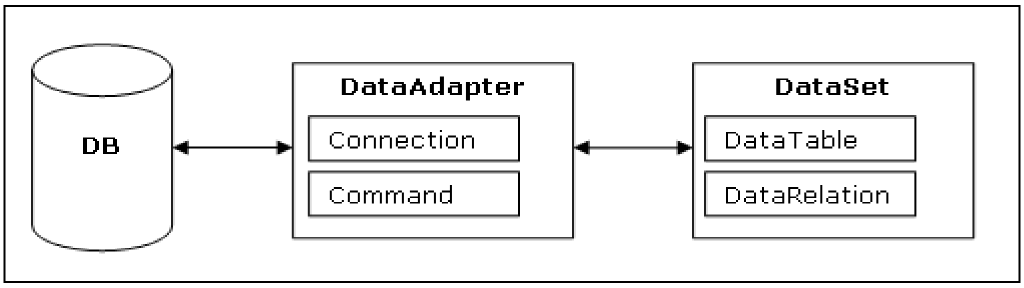
After creating an instance of the Command object, you have to create a DataReader by calling Command.ExecuteReader to retrieve rows from a data source.

```
SqlDataReader reader = cmd.ExecuteReader();
```

When the ExecuteReader method in the SqlCommand Object execute, it will instantiate a SqlClient.SqlDataReader Object. When we started to read from a DataReader it should always be open and positioned prior to the first record. The Read() method in the DataReader is used to read the rows from DataReader and it always moves forward to a new valid row, if any row exist . You should always call the Close method when you have finished using the DataReader object.

DATAADAPTER:

DataAdapter serves as a bridge between a DataSet and SQL Server for retrieving and saving data. We can use SqlDataAdapter Object in combination with Dataset Object. DataAdapter provides this combination by mapping Fill method, which changes the data in the DataSet to match the data in the data source, and Update, which changes the data in the data source to match the data in the DataSet.

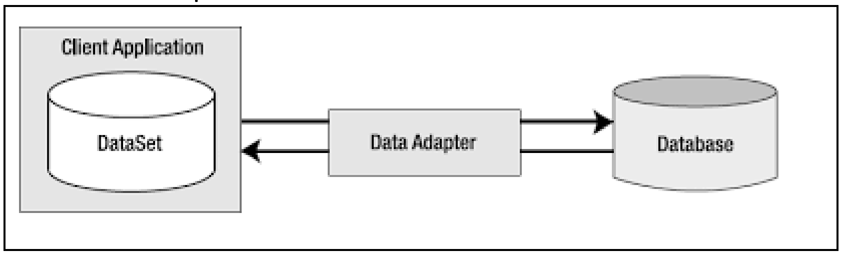


```
SqlDataAdapter adapter = new SqlDataAdapter(sql,connection );
adapter.Fill(ds);
```

The SqlDataAdapter Object and DataSet objects are combine to perform both data access and data manipulation operations in the SQL Server Database. When the user perform the SQL operations like Select , Insert etc. in the data containing in the Dataset Object , it won't directly affect the Database, until the user invoke the Update method in the SqlDataAdapter.

DATASET:

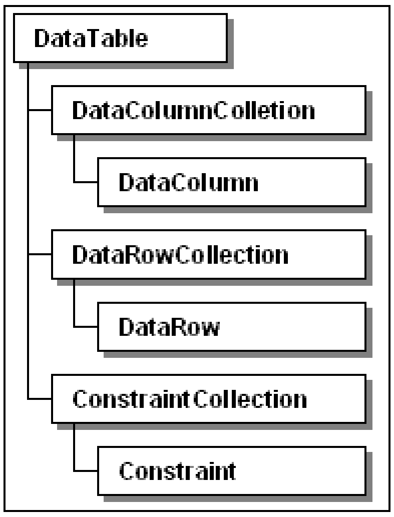
The ADO.NET DataSet contains DataTableCollection and their DataRelationCollection . It represents a complete set of data including the tables that contain, order, and constrain the data, as well as the relationships between the tables.



We can use Dataset in combination with DataAdapter Class . Build and fill each DataTable in a DataSet with data from a data source using a DataAdapter. The DataSet object offers disconnected data source architecture.

DATACOLUMN :

DataColumn is the generalized object which is not related to any specific data provider. It also provided by System.Data namespace.



dataColumn contain a single column of Datatable or DataSet. It has information regarding column like name, data type, default value, maximum length etc.

DATAROW:

In **C#.**Net, **DataRow** class provides the functionality to add a new row i.e. new record into the **DataTable**. **DataRow object** inherits the schema of C#.Net DataTable and allows you to add the values for each **DataColumn** according to the specified **DataType** for a data column of DataTable. You can use the zero based index number of DataColumn or column name to pass the value for specific column in the new DataRow. Index of the DataColumn depends upon the order of adding a DataColumn to the DataTable.

DATAVIEW:

A DataView enables you to create different views of the data stored in a DataTable, a capability that is often used in data-binding applications. Using a DataView, you can expose the data in a table with different sort orders, and you can filter the data by row state or based on a filter expression.

A DataView provides a dynamic view of data in the underlying DataTable: the content, ordering, and membership reflect changes as they occur. This behavior differs from the Select method of the DataTable, which returns a DataRow array from a table based on a particular filter and/or sort order: thiscontent reflects changes to the underlying table, but its membership and ordering remain static. The dynamic capabilities of the DataView make it ideal for data-binding applications.

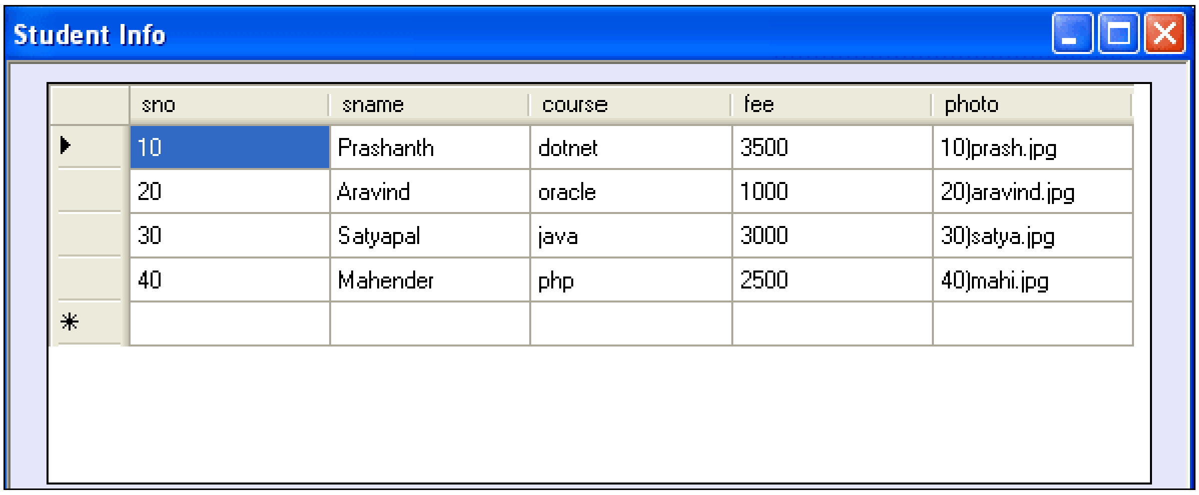
A DataView provides you with a dynamic view of a single set of data, much like a database view, to which you can apply different sorting and filtering criteria. Unlike a database view, however, a DataView cannot be treated as a table and cannot provide a view of joined tables. You also cannot exclude columns that exist in the source table, nor can you append columns, such as computational columns, that do not exist in the source table.

DATAGRIDVIEW CONTROL:

The GridView view mode is used to display a list of data items by bindings the data fields to columns and by displaying a column header to identify the field. Default GridView Style implements button as column headers. This column header is used for interaction capabilities for example for sorting GridView data according to a specific column.

Windows Forms provides the useful **DataGridView** control, which allows you to display structured data from memory or from any database in an automated way. In Windows Forms, you do not need to develop your own grid logic, but can simply plug in your data to the DataGridView.

```
DataSet DS = new DataSet();
DA.Fill(DS);
DataTable dt = new DataTable("DS");
dt = DS.Tables[0];
dataGridView1.DataSource = dt;
```



sno	sname	course	fee	photo
10	Prashanth	dotnet	3500	10)prash.jpg
20	Aravind	oracle	1000	20)aravind.jpg
30	Satyapal	java	3000	30)satya.jpg
40	Mahender	php	2500	40)mahi.jpg

Creating User Control with Property – Method -Event

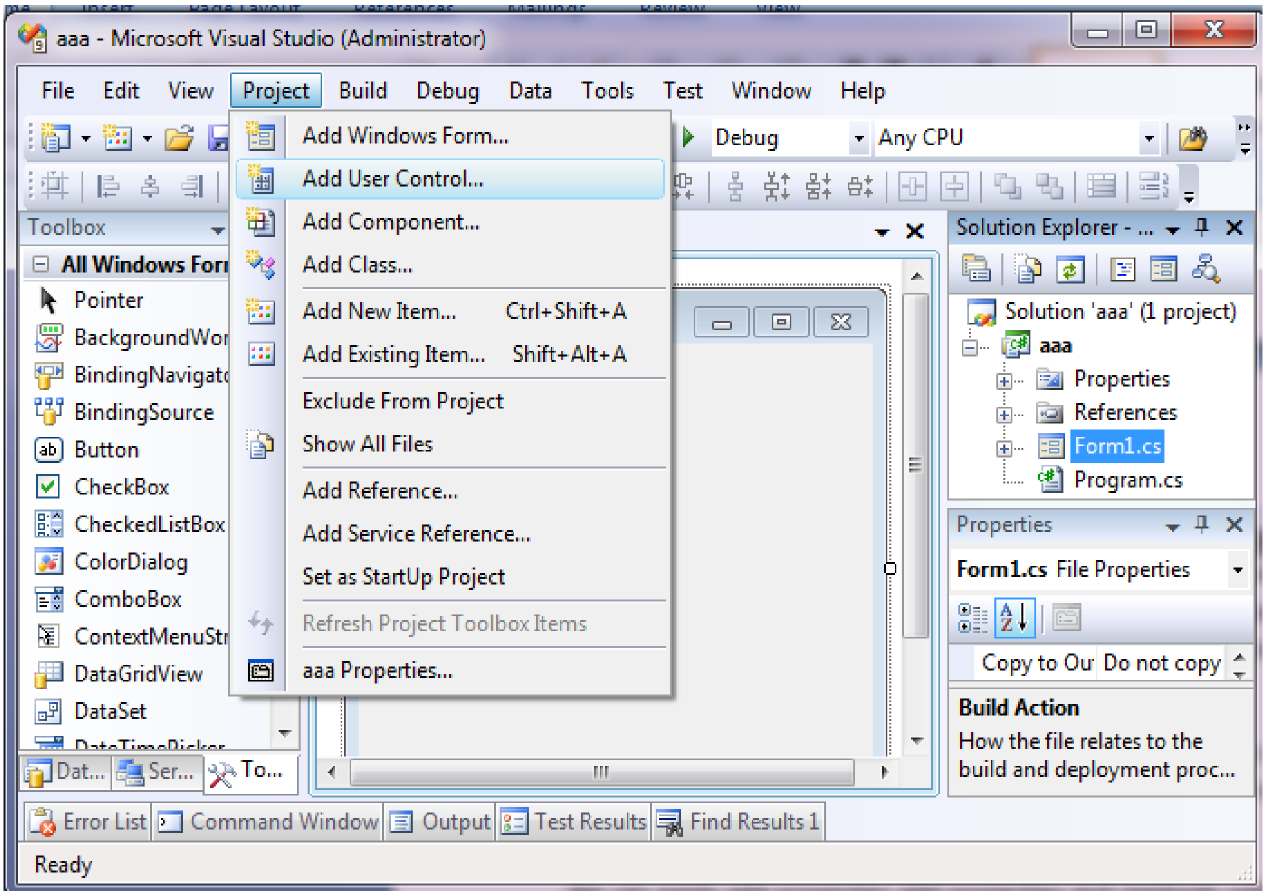
A custom control is user defined control. It is created by the user. It is derived from the UserControl class, the control class, or an existing Windows Forms control User control provides front-end functionality through its graphical interface, but provides back – end functionality through its methods, properties, and events.

Methods are the actions a user can tell your control to execute. Properties and member variable are data a user of your control can get or set. Events are notifications that something interesting has happened to your component.

We can easily add properties add properties and methods with any control, method public procedure you declare in your control class and a property can be declared using a property statement in visual basic, or implemented as a public member variable in C#.

To add to a .NET custom control project go through the following points.

1. Right click on solution explorer windows and select add.
2. Use Control and select user control you want to create or to add control to a user control just drag it on design surface with the same way you design windows form.



To create any control, through code, you simple instantiate the control similar to primitive data types (int, string, double, etc). However, you will instantiate it using the New keyword.

```
TextBox aTextBox = New TextBox();
aTextBox.ForeColor = Color.Red;
aTextBox.Location = New Point(50, 100);
private void Form1_Load(object sender, EventArgs e)
{
        TextBox atextbox = new TextBox();
        atextbox.BackColor = Color.Green;
        atextbox.BringToFront();
        atextbox.SelectAll();
        atextbox.Copy();
        atextbox.Location = new Point(50, 100);

        Button abutton = new Button();
        abutton.Location = new Point(45, 55);
        abutton.Text = "Click Me";

}
```

Creating Your Own Control vs. Using Microsoft Controls:

There are many reasons why you might want to create your own Controls even though Microsoft supplies a pretty rich set of controls right out of the box. The boxed controls may not solve all of your special business problems, so you'll want to customize their or create new controls to meet your specific needs. One of the most common uses that you will find for creating your own controls is for individualizing input masks and business rules for specific types of identifiers used in your applications across your enterprise.

A good example would be an employee number that is formatted in a special way and must be verified against a table in a database to make sure it exists. Instead of having each programmer create all of this logic from scratch, you might create a control that will perform the logic for validating the correct format and verifying its existence in database.

CREATING CRYSTAL REPORTS:

Crystal Report is a Reporting Application that can generate reports from various Data Sources like Databases, XML files etc.. The Visual Studio.NET Integrated Development Environment comes with Crystal Reports tools. The Crystal Reports makes it easy to create simple reports, and also has comprehensive tools that you need to produce complex or specialized reports in csharp and other programming languages.

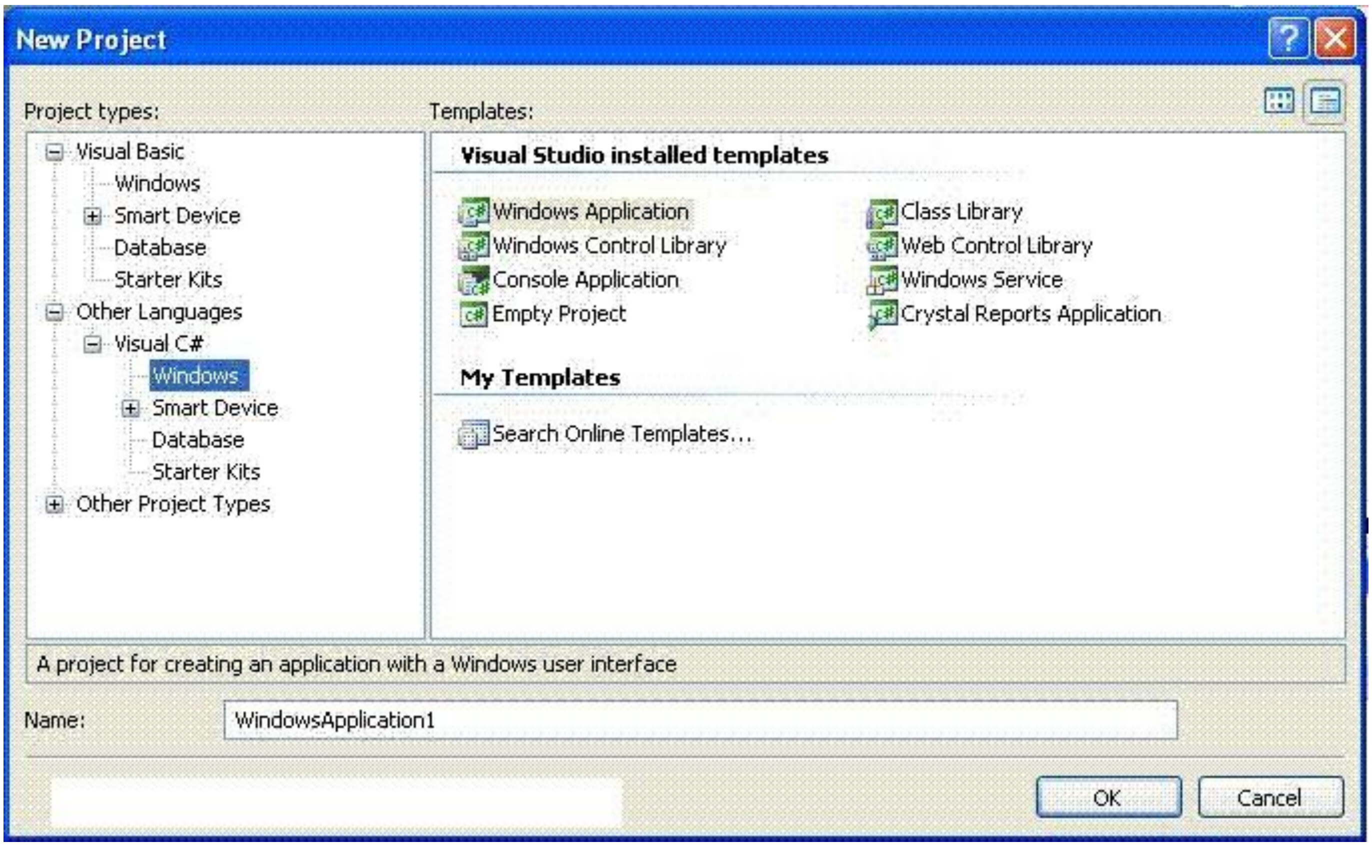
Crystal Reports is compatible with most popular development environments like **C#, VB.NET** etc. You can use the Crystal Reports Designer in Visual Studio .NET to create a new report or modify an existing report.

From the following sections you can find useful resources for generating customized reports from Crystal Reports with **C#**.

Let's see How to create a basic crystal report step by step using windows forms and C#. to create this you require only little knowledge about Database like MS – Access, SQL – Server and C#.Net .

Create a database.

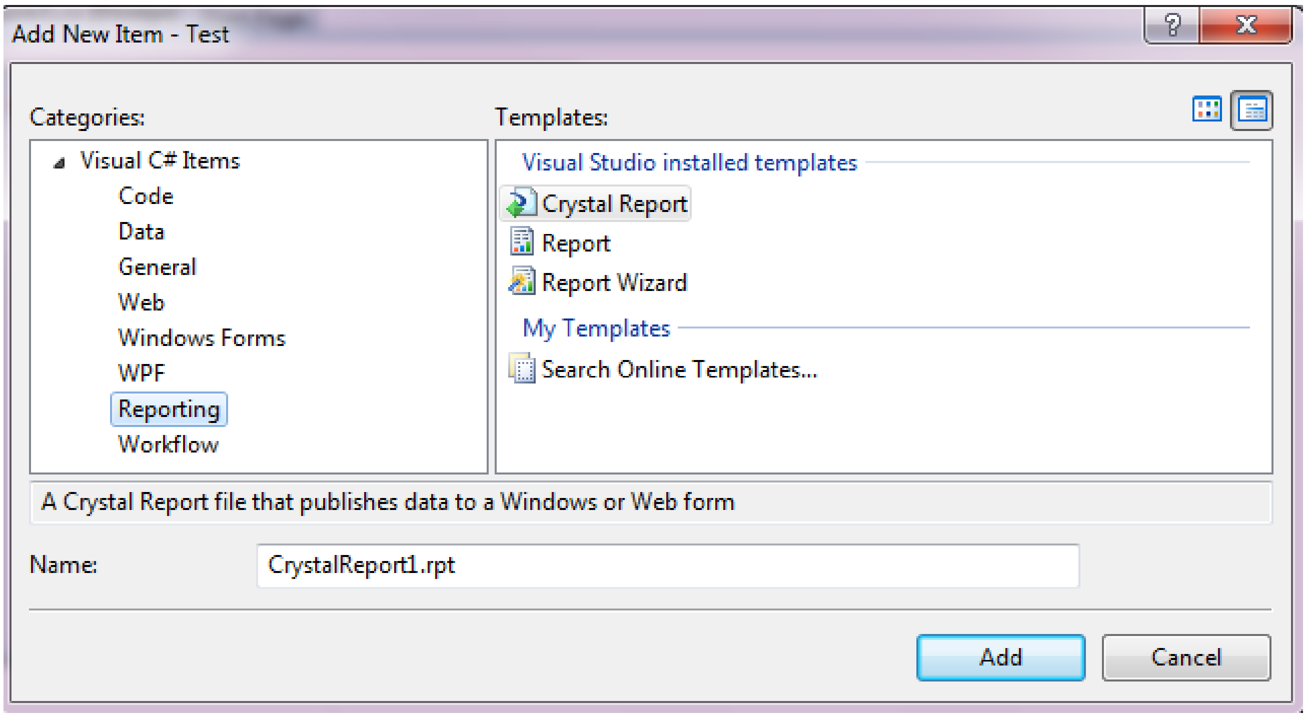
Create a table under that database.
Then move to Microsoft Visual Studio and create windows forms.



A project for creating an application with a Windows user interface

Name: WindowsApplication1

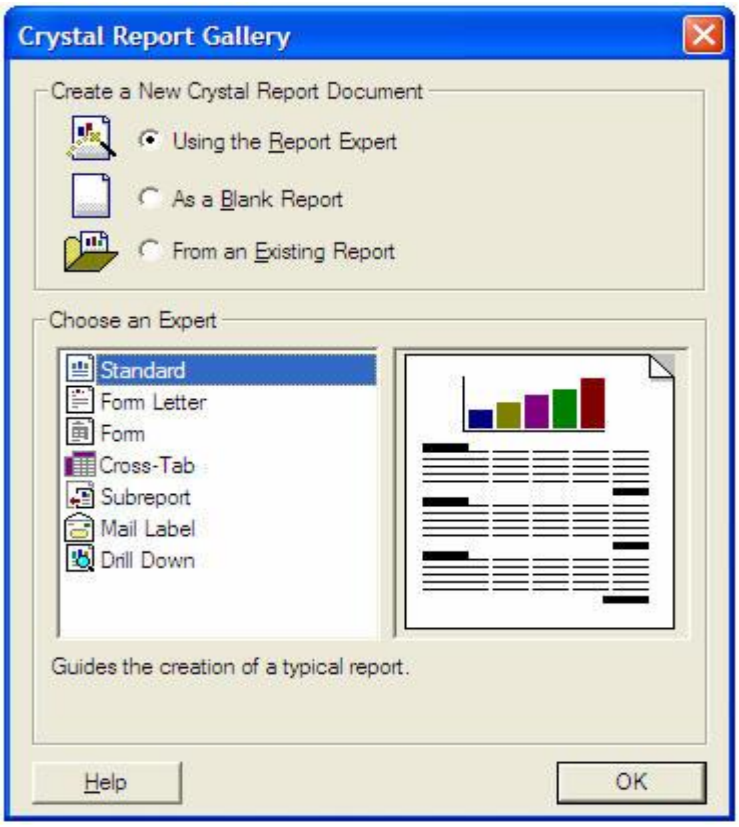
Now you will get the default Form1.cs from the main menu in visual studio C# project select PROJECT -> Add new Item. Then Add New Item dialogue will appear and select Crystal report from the dialogue box.



A Crystal Report file that publishes data to a Windows or Web form

Name: CrystalReport1.rpt

Select report type from Crystal Reports gallery.


Crystal Report Gallery

Create a New Crystal Report Document

(●) Using the Report Expert

( ) As a Blank Report

( ) From an Existing Report

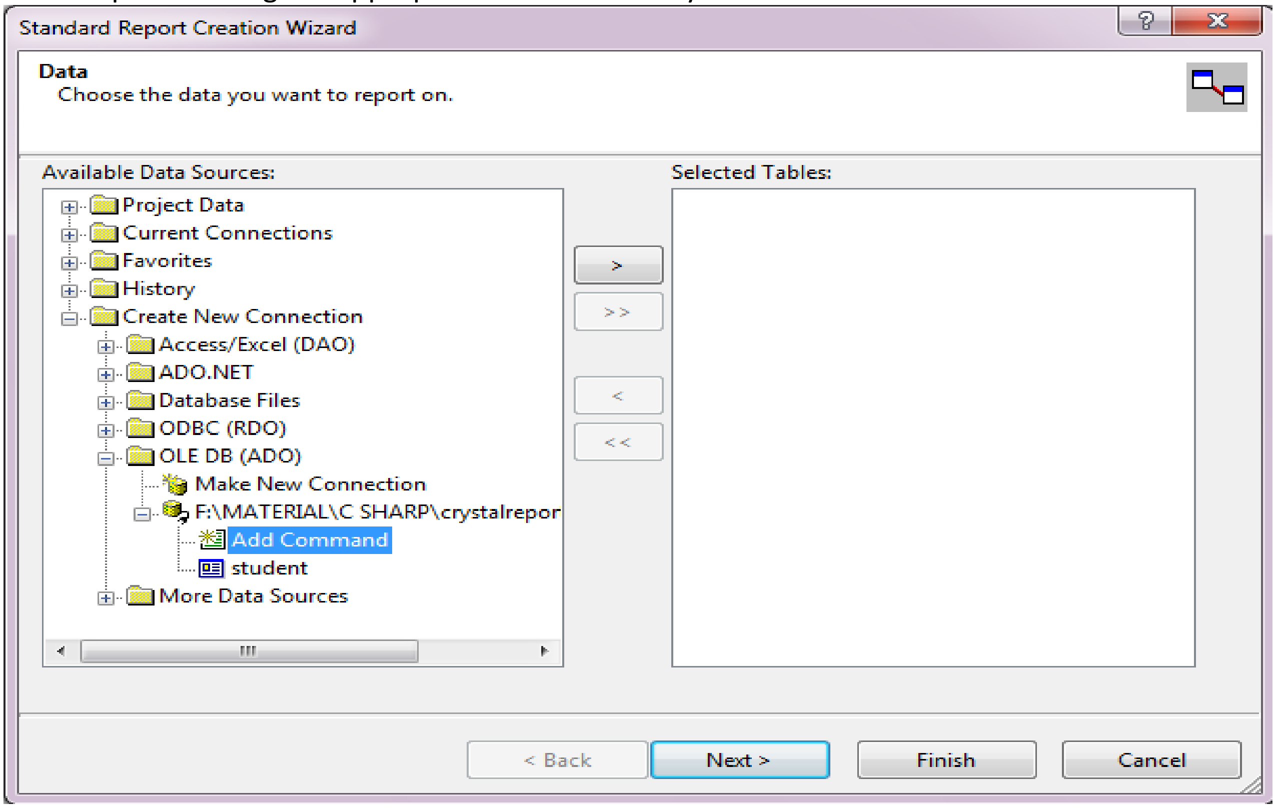
Choose an Expert

- Standard
- Form Letter
- Form
- Cross-Tab
- Subreport
- Mail Label
- Drill Down

Guides the creation of a typical report.

Accept the default setting and click OK.

Next step is selecting the appropriate connection to your database.


Standard Report Creation Wizard

Data

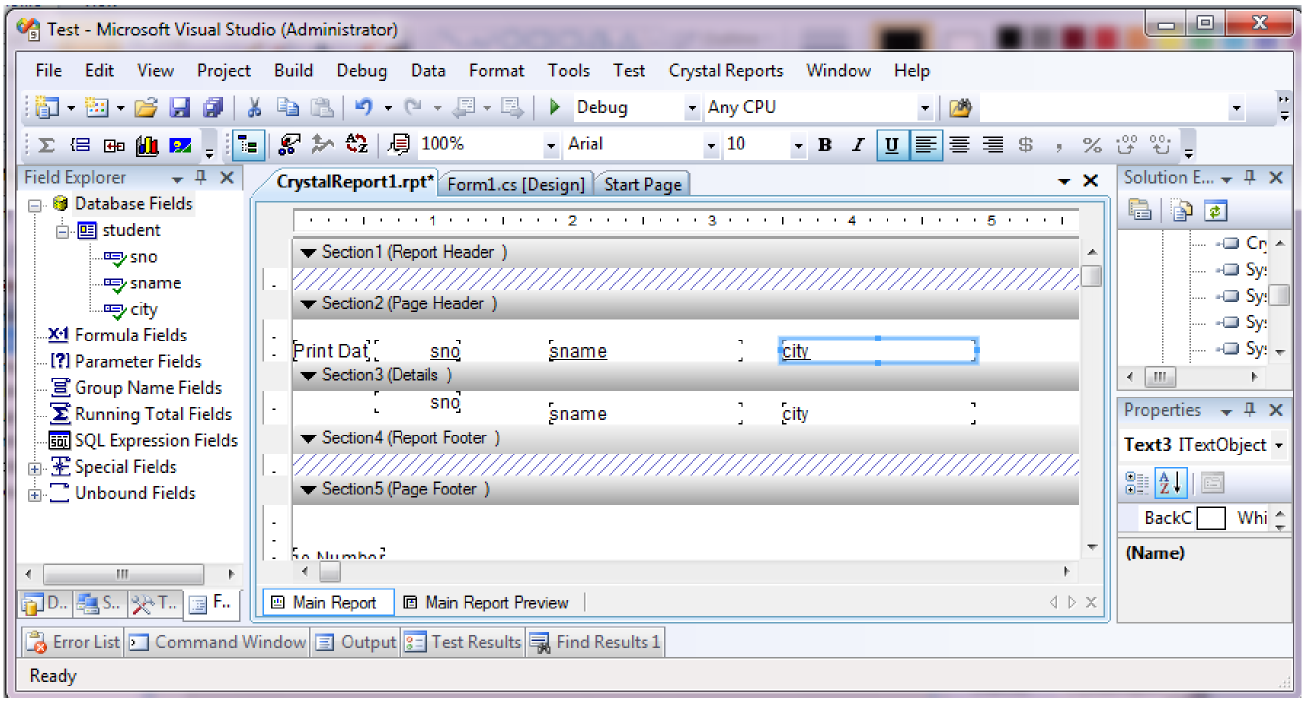
Choose the data you want to report on.

Available Data Sources:

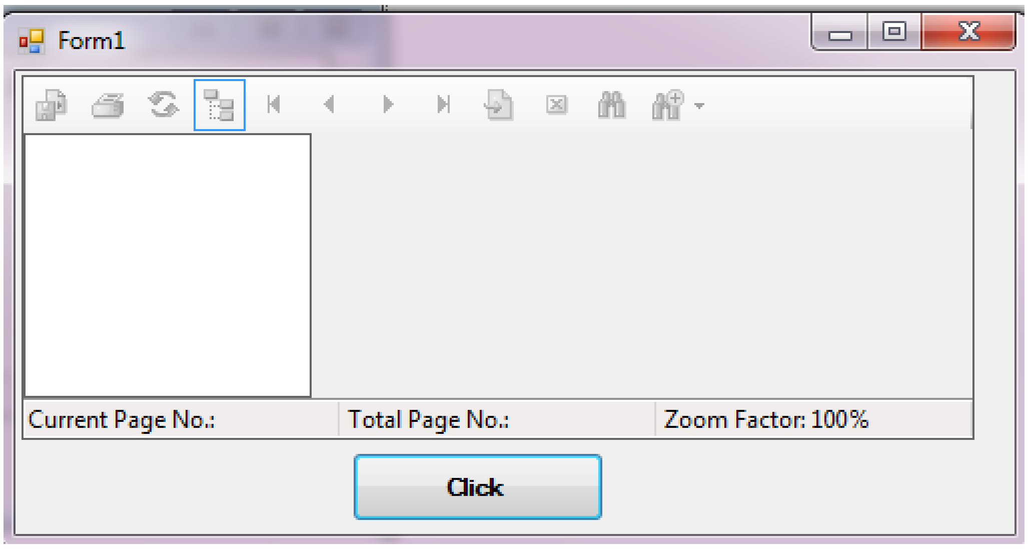
- Project Data
- Current Connections
- Favorites
- History
- Create New Connection
  - Access/Excel (DAO)
  - ADO.NET
  - Database Files
  - ODBC (RDO)
  - OLE DB (ADO)
    - Make New Connection
    - F:\MATERIAL\C SHARP\crystalrepor
      - Add Command
      - student
  - More Data Sources

Selected Tables:

After you click the finished button, the next windows you will get your oledbconnection, from there selected database name and click the tables then you can see all your tables from your database.


Test - Microsoft Visual Studio (Administrator)
File Edit View Project Build Debug Data Format Tools Test Crystal Reports Window Help
Field Explorer
Database Fields
student
sno
sname
city
Formula Fields
Parameter Fields
Group Name Fields
Running Total Fields
SQL Expression Fields
Special Fields
Unbound Fields
CrystalReport1.rpt* Form1.cs [Design] Start Page
Section1 (Report Header)
Section2 (Page Header)
Print Dat sno sname city
Section3 (Details)
sno sname city
Section4 (Report Footer)
Section5 (Page Footer)
Main Report Main Report Preview
Solution E...
Properties
Text3 TextObject
Error List Command Window Output Test Results Find Results 1
Ready

Click finished button. Then you can see the crystal reports designer windows in your C# project. In the crystal reports designer windows you can see the selected field from student table. You can also arrange the field objects and design of the screen according your requirements. After that your screen is look like the above picture.

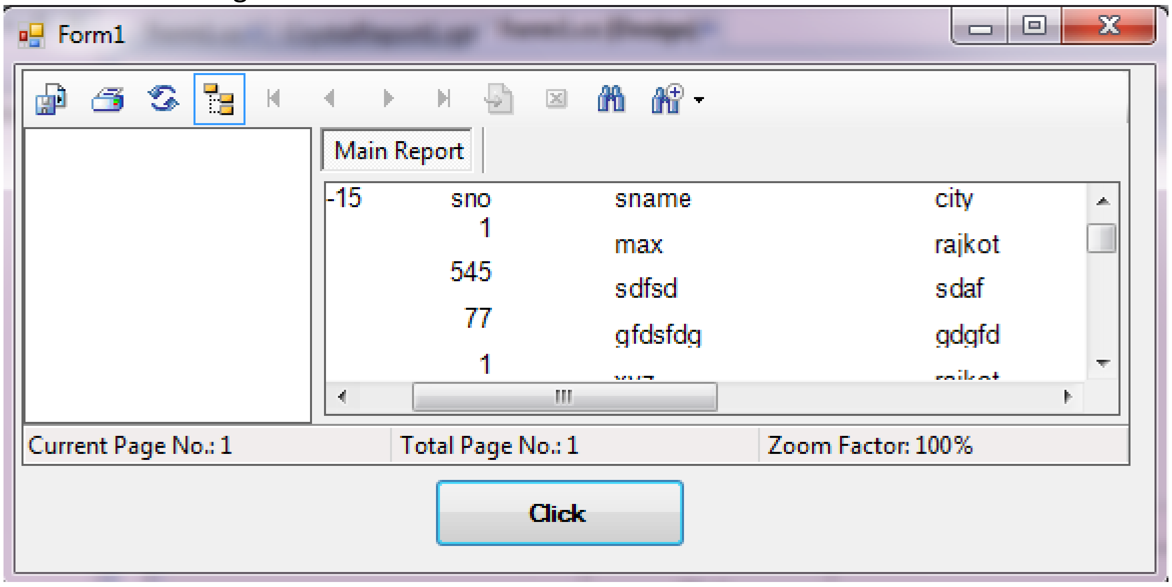


Now the designing part is over and the next step is to call the crysal rpoerts in your C# application and view it through crystal reports viewer control in c#. select the default form(form1.cs) you created in c# and drag a button and a crystalreportviewer control to your form.

Code

```
private void button1_Click(object sender, EventArgs e)
 {
     CrystalReport1 R1 = new CrystalReport1();
     R1.Load(@"D\crystalreport\Test\Test\CrystalReport1.rpt");
     crystalReportViewer1.ReportSource = R1;
     crystalReportViewer1.Refresh();
 }
```

Now add following source code in to form1.cs file.



```
-15      sno     sname      city
           1     max        rajkot
         545     sdfsd      sdaf
          77     gfdsfdg    gdgfd
           1
Current Page No.: 1    Total Page No.: 1    Zoom Factor: 100%
```

Now run the project and click on report button output will be like as above .

TYPES OF REPORTS IN CRYSTAL REPORTS

These are all used to create the crystal on Different ways. those ways are explained below...

Standard Report Creation Wizard:
Use this command to open the Standard Report Creation Wizard. This wizard guides you through the creation of a typical report through step by step. This is called Help of wizard. It will help you to create report step by step process. Most of the case it will help for beginners to create reports in crystal tool.

Blank Report Creation:
Use this command to create a new report without the help of a wizard. When you select this option, the Database Expert appears so that you can choose a data source. With option you can create a sample report with out data source. You can select database after you create a report.

Cross-Tab Report Creation Wizard:
Use this command to open the Cross-Tab Report Creation Wizard. This wizard guides you through the creation of a report that contains a summarized grid. So it will directly help to create cross tab in the report.

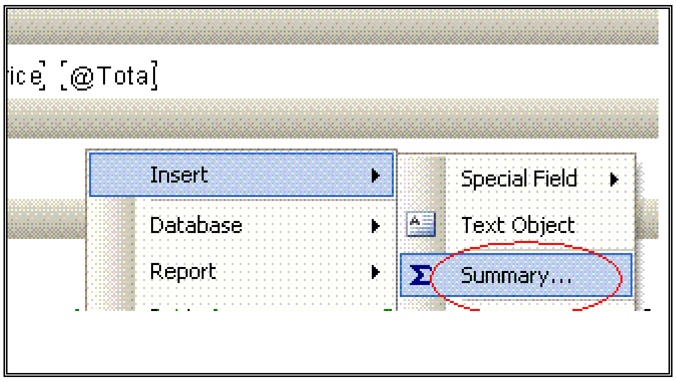
Mailing Labels Report Creation Wizard:
Use this command to open the Mailing Labels Report Creation Wizard. This wizard guides you through the creation of a report with multiple columns. The wizard is commonly used to create a sheet of address labels.

OLAP Report Creation Wizard:

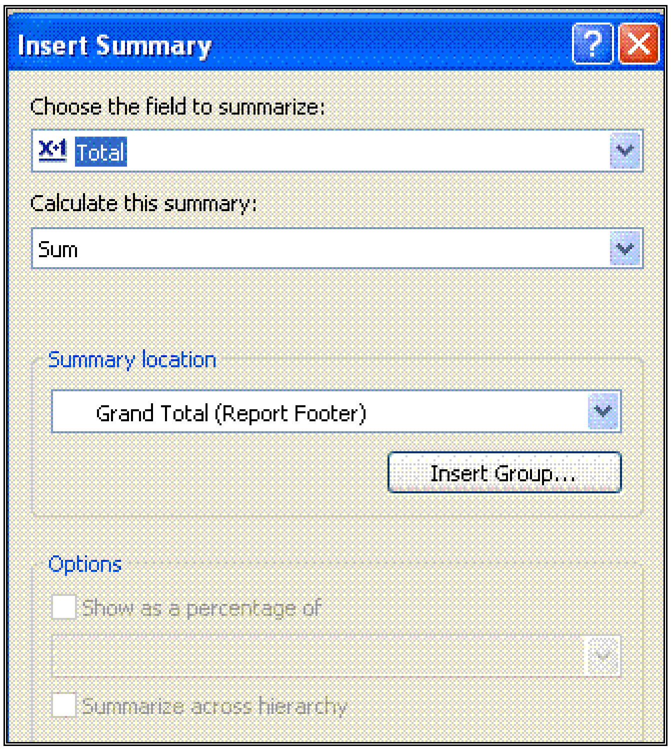
Use this command to open the OLAP Report Creation Wizard. This wizard guides you through the creation of a report that contains a summarized grid based on an OLAP data source. you can create the reports based on SQL server and Oracle OLAP data sources.

SUMMARY FIELD IN CRYSTAL REPORT:

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

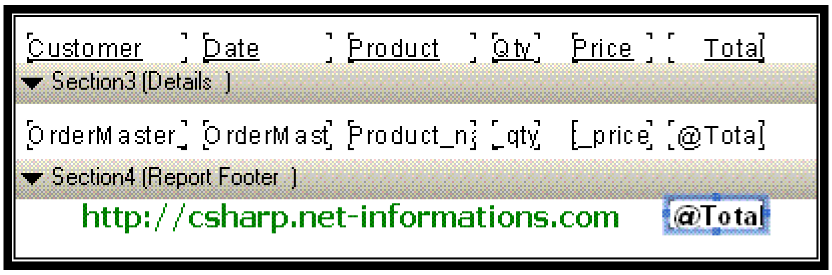

Insert Summary

Choose the field to summarize: Total

Calculate this summary: Sum

Summary location: Grand Total (Report Footer)

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 .

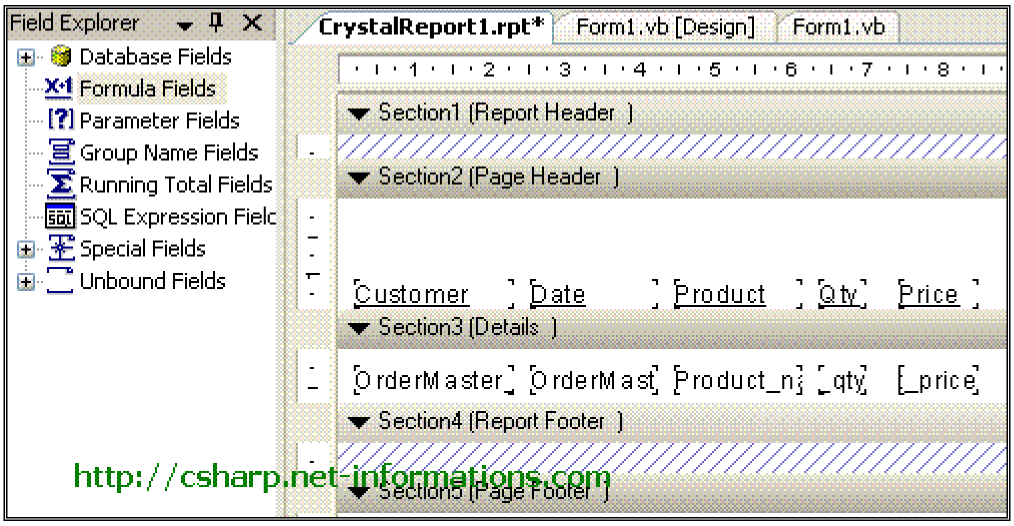
FORMULA FIELD IN CRYSTAL REPORT:

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.

In this C# Crystal Reports tutorial we are showing that all orders with qty and price and the total of each row , that means each in each row we are showing the total of qty and price.

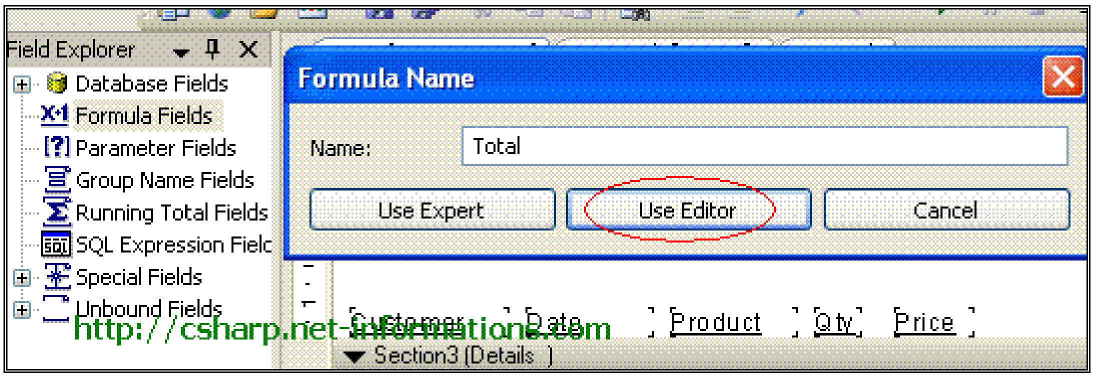
Before starting this tutorial Create a new Crystal Reports with fields CustomerName , Order Date , Product Name and Product Price . If you do not know how to create this report , just look the previous tutorial C# Crystal Reports from multiple tables . In that report selecting only four fields , here 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 :

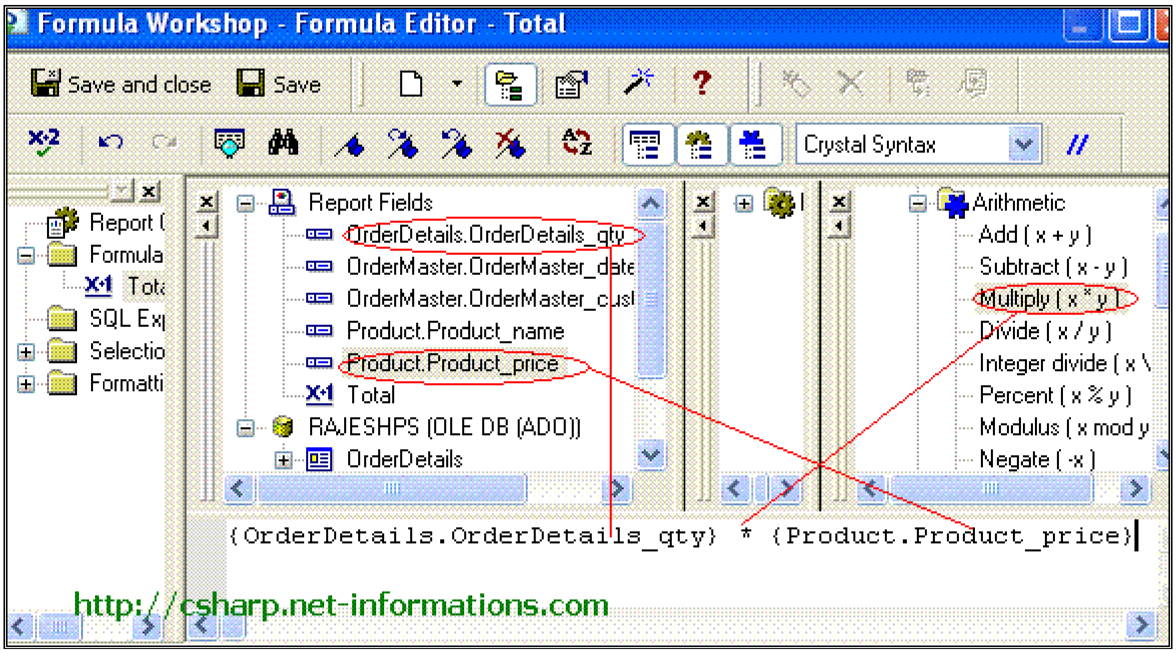


Next step is to create a Formula Field for showing the result of **Qty X Price** .

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.

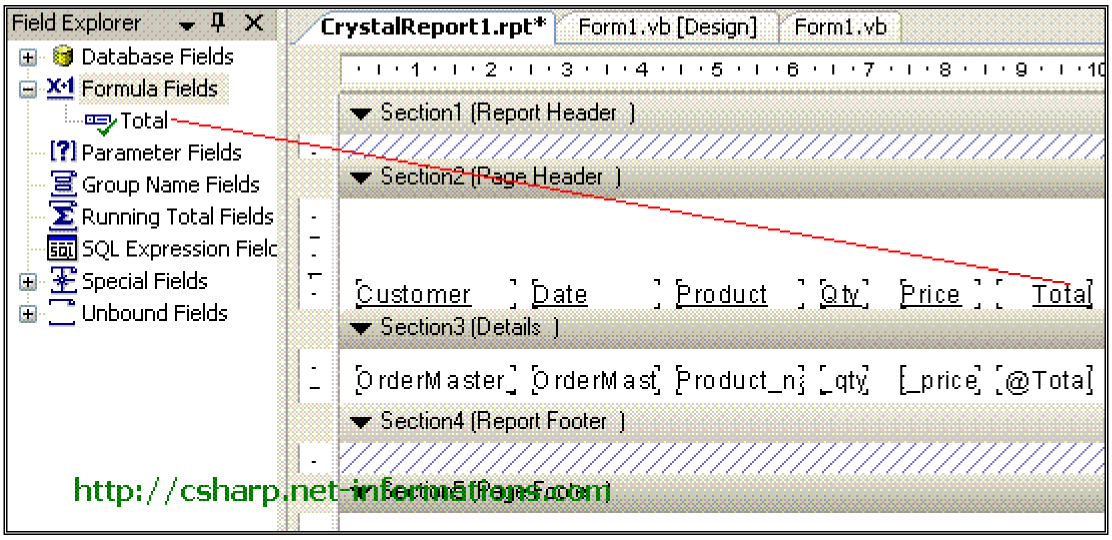


Now you can see the Formula Editor screen . Here you can enter which formula you want . Here we want the result of Qty X Price . For that we select OrderDetails.Qty , the multipy operator (*) and Product.Price . Double click each field for selection.



```
{OrderDetails.OrderDetails_qty} * {Product.Product_price}
```

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 .

Setup Projects

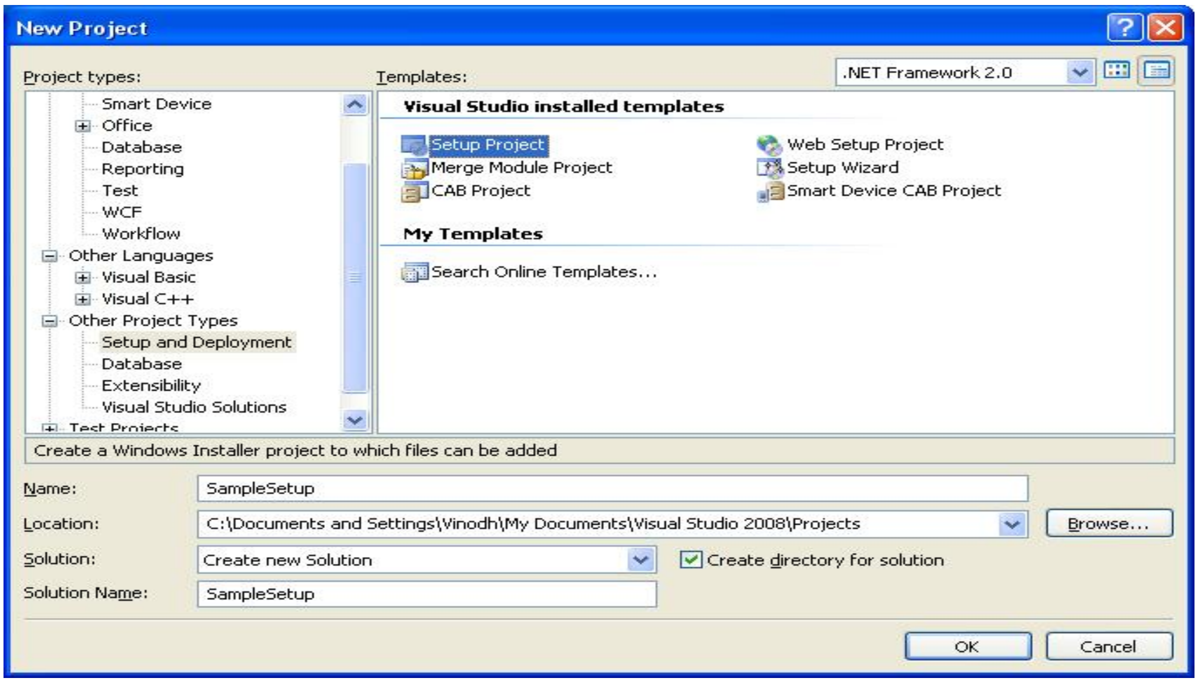
Visual studio provides templates for four types of deployment projects: merge module project, setup project, web setup project and CAB project. In addition, a setup wizard helps you create deployment projects. You can see the templates and the wizard in the new project dialog box. Expand the other project type's node, select the setup and deployment project node, and then click visual studio installer.

Types of Setup projects

Project type	**Purpose**
Merge Module Project	Packages components that can be shared by multiple windows-based applications. Merge module projects let you package files or components into a single module, which makes the module easier to share. The resulting .msm files can be included in any other deployment project.
Setup Project	Builds an installer for a windows – based application. The files for a set up project are installed into the program files directory on end-user computers.
Web Setup Project	Build an installer for a web application. The files for a web setup projects are installed into a virtual root directory on web servers.
CAB Project	Create a CAB files for downloading to an earlier web browser. You can use CAB projects to package ActiveX components that can be downloaded from a web server to a web browser.

Create a setup project:

To get started, create a console, windows forms, or WPF application that performs some simple function, such as displaying "Hello C#" to do this, and we just want to use the program so that we can build a setup application for it. For our current purposes, it doesn't matter what the program does.
To create the setup project, right click the solution, select new project, open the other project types branch, select setup and deployment, select setup wizard, name it samplesetup and click ok. You'll see the welcome screen and click next to move to the project type screen.


New Project

Project types:
- Smart Device
- Office
- Database
- Reporting
- Test
- WCF
- Workflow
- Other Languages
  - Visual Basic
  - Visual C++
- Other Project Types
  - Setup and Deployment
  - Database
  - Extensibility
  - Visual Studio Solutions
- Test Projects

Templates: .NET Framework 2.0

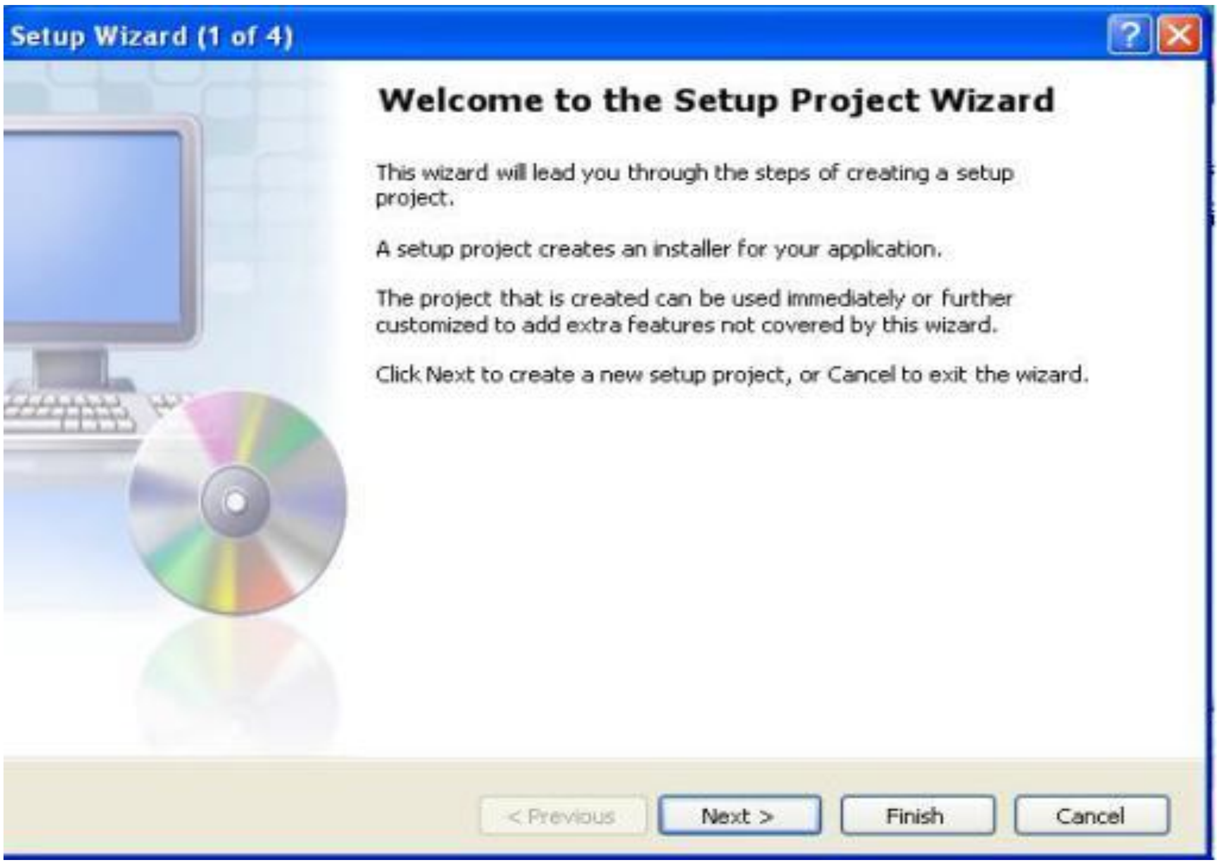
Visual Studio installed templates
- Setup Project
- Merge Module Project
- CAB Project
- Web Setup Project
- Setup Wizard
- Smart Device CAB Project

My Templates
- Search Online Templates...

Create a Windows Installer project to which files can be added

Name:	SampleSetup
Location:	C:\Documents and Settings\Vinodh\My Documents\Visual Studio 2008\Projects
Solution:	Create new Solution
Solution Name:	SampleSetup

[✓] Create directory for solution


Setup Wizard (1 of 4)

Welcome to the Setup Project Wizard

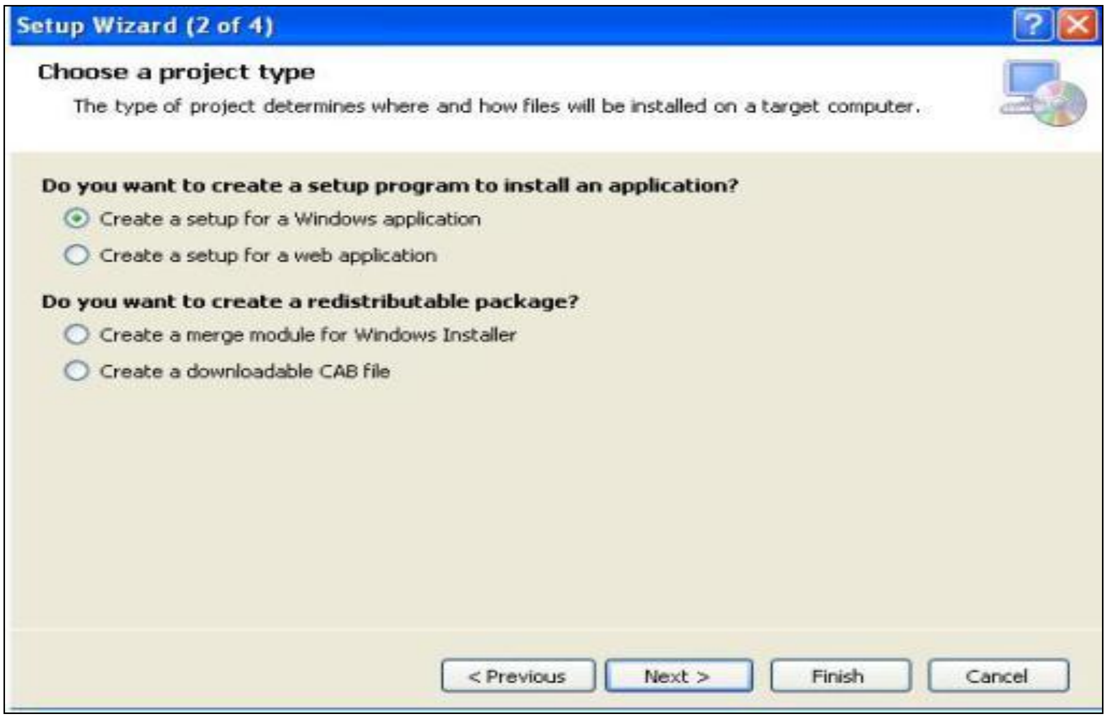
This wizard will lead you through the steps of creating a setup project.

A setup project creates an installer for your application.

The project that is created can be used immediately or further customized to add extra features not covered by this wizard.

Click Next to create a new setup project, or Cancel to exit the wizard.

Choose project type as per requirement; her windows application is selected. The click on next button.


Setup Wizard (2 of 4)

Choose a project type

The type of project determines where and how files will be installed on a target computer.

Do you want to create a setup program to install an application?

[✓ Create a setup for a Windows application]

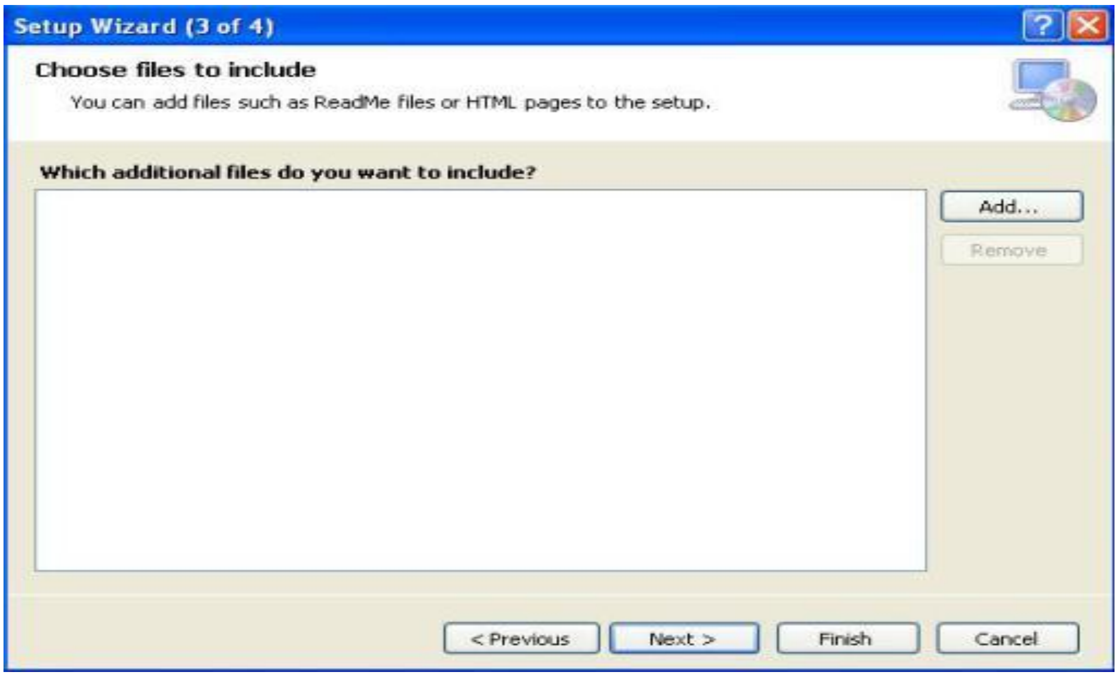
Create a setup for a web application

Do you want to create a redistributable package?

Create a merge module for Windows Installer

Create a downloadable CAB file

Next windows allow you to include setup files. And then click on next button.

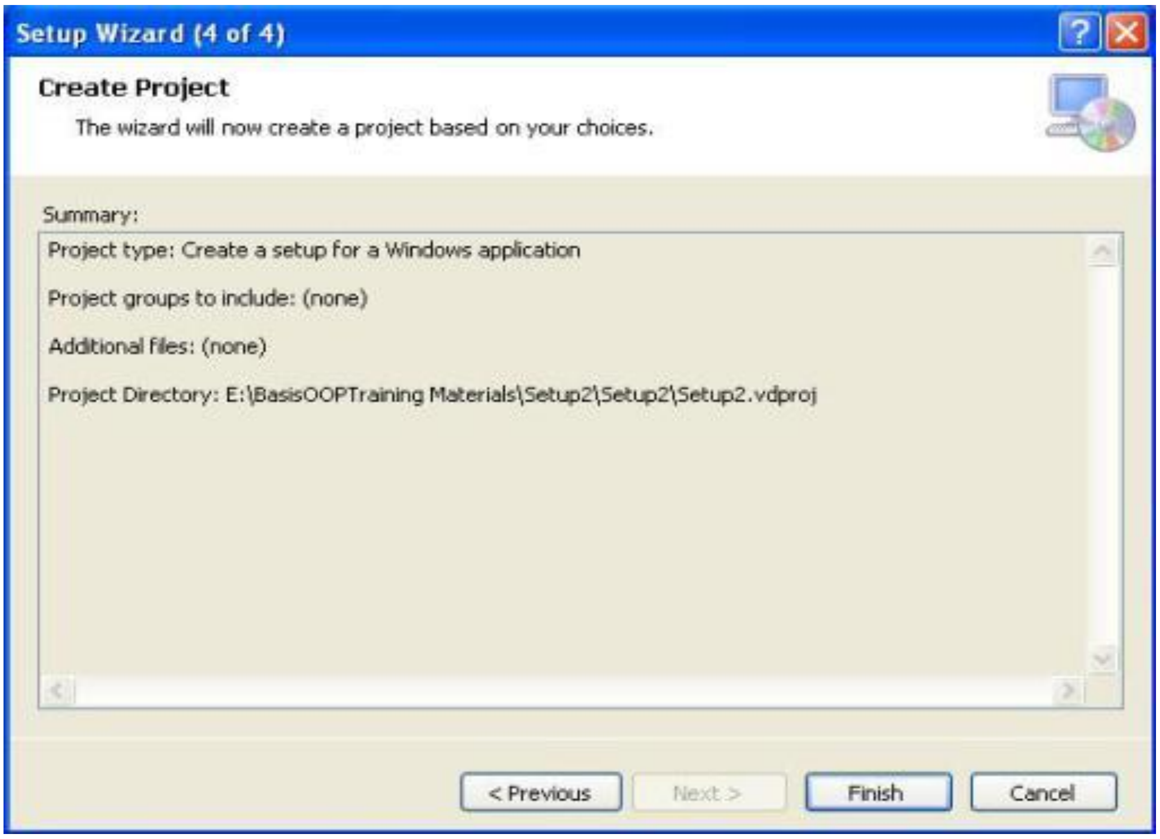

Setup Wizard (3 of 4)

Choose files to include

You can add files such as ReadMe files or HTML pages to the setup.

Which additional files do you want to include?

The create project window summarized s the choice you've made. Click the finished button to produce the new setup project, which appears as a project under your solution.


Setup Wizard (4 of 4)

Create Project

The wizard will now create a project based on your choices.

Summary:

Project type: Create a setup for a Windows application

Project groups to include: (none)

Additional files: (none)

Project Directory: E:\BasisOOPTraining Materials\Setup2\Setup2\Setup2.vdproj

File System Editor:

Adds project output and other files to a deployment project, to specify the locations where files be installed on a target computer, and to create shortcuts on a target computer. To access the file system editor, on the view menu, point to editor, and then click file system when a deployment project is selected in solution explorer.
The file system editor is divided into two parts: a navigation pane on the left and a details pane on the right. The navigation pane contains a hierarchical list of folders that represent the file system on a target computer. The folder names correspond to standard windows folders. For example, the application folder corresponds to a folder beneath the program files folder where the application will be installed. When a folder is selected in the navigation pane, any files and shortcuts that will be installed in that folder are displayed in the details pane.

User Interface Editor:

UI editor allows you to specify and edit dialog boxes that are displayed during installation on a target computer. To access the User Interface editor, on the View menu, point to Editor, and then click user interface when a deployment project is selected in solution explorer.
The user interface editor contains a single pane with hierarchical list of user interface dialog boxes. The list is divided into two sections for standard versus administrative installation, and each section contains start, progress and end nodes to represent the stages of installation.

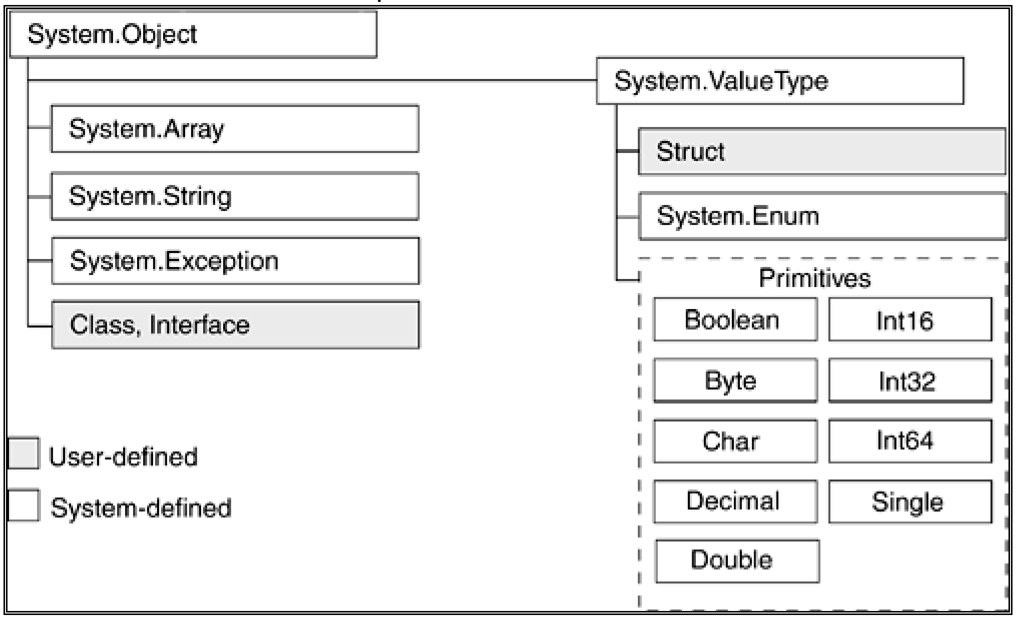
Launch Conditions Editor:

The launch conditions editors allows you to specify conditions that must be met in order to successfully run an installation. For example, you might want to check for a specific version of an operation system – if a user attempt to install on a system that does not meet the condition, the

installation will not occur. Searches can be performed on a target computer to determine if a particular file, registry key, or Microsoft windows installer component exists.
Predefined launch conations allows you to add a both search and a launch condition in a single step. The property for the search is automatically referenced in the condition property of the launch condition.

VALUE TYPE AND REFERENCE TYPE:

The Common Language Runtime (CLR) supports two kinds of types: reference types and value. Reference types include classes, arrays, interfaces, and delegates. Value types include the primitive data types such as int, char, and byte as well as struct and enum types. Value and reference types are distinguished by their location in the .NET class hierarchy and the way in which .NET allocates memory for each. Value types directly contain their data which are either allocated on the stack or allocated in-line in a structure. Reference types store a reference to the value's memory address, and are allocated on the heap.



Value types derive from System.ValueType. Reference types derive from System.Object.