What is .NET?

→.NET is not an operating system. .NET is not a programming language. ".NET is a framework".

→The .NET Framework is an environment or platform for developing, deploying and executing applications.

→It is a new, easy, and extensive programming platform. It is not a programming language, but it supports several programming languages. (.NET is a framework)

→By default .NET comes with few programming languages including C# (C Sharp), VB.NET, J# and managed C++. .NET is a common platform for all the .NET supported languages.

→ .NET framework class library is common for all the .net supported languages. So, developers need not learn many libraries when they switch to a different language. Only the syntax is different for each language.

Whether you write code in C# or VB.NET, you are calling methods in the same .NET class libraries. The same .NET framework executes the C# and VB.NET applications. So, there won't be any performance difference based on the language you write code. The common class library in .NET Framework can assist you in developing your applications in a faster, cheaper and easier manner.

→ In future versions of Windows, .NET will be freely distributed as part of operating system and users will never have to install .NET separately. The .NET Framework is included with Windows Server 2003, Windows Server 2008 and Windows Vista, and can be installed on older versions of Windows.

Version of .Net Framework :

Version	Release Date	Visual Studio	Default in Windows
.Net Framework 1.0	13-02-2002	Visual Studio .Net	
.Net Framework 1.1	24-04-2003	Visual Studio .Net 2003	Windows Server 2003
.Net Framework 2.0	07-11-2005	Visual Studio 2005	
.Net Framework 3.0	06-11-2006		Window Vista, Windows Server 2008
.Net Framework 3.5	19-11-2007	Visual Studio 2008	Windows 7, Windows Server 2008 R2
.Net Framework 4.0	12-04-2010	Visual Studio 2010	
.Net Framework 4.5	15-08-2012	Visual Studio 2012	Windows 8, Windows Server 2012

The .NET Framework consists of two main components:

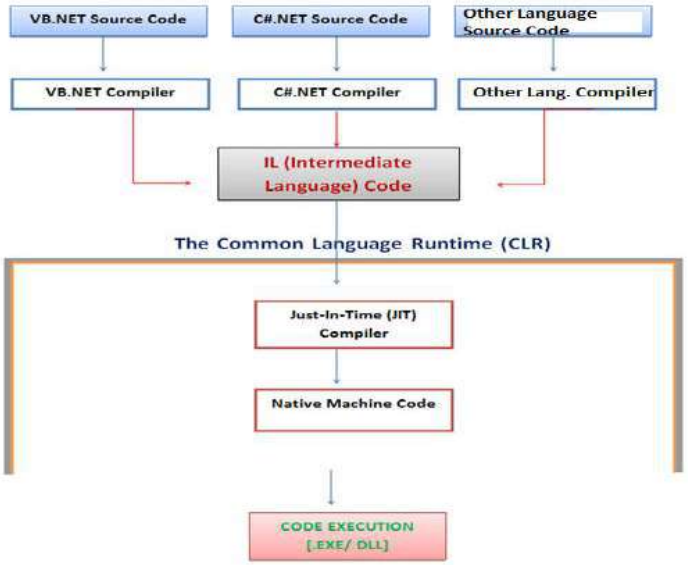
(1) Common Language Runtime (CLR)

(2) Class Libraries.Net framework class library (FCL) OR Base Class Library (BCL)

1) **Common Language Runtime** (CLR)
.Net Framework provides runtime environment called **Common Language Runtime** (CLR).It provides an environment to run all the .Net Programs. The code which runs under the CLR is called as **Managed Code**. Programmers need not to worry on managing the memory if the programs are running under the CLR as it provides memory management and thread management.

   Programmatically, when our program needs memory, CLR allocates the memory for scope and de-allocates the memory if the scope is completed.

The Compilation Divided in to Two Step.
In First step 1) Language Compilers (e.g. C#, VB.Net, J#) will convert the Code/Program to **Microsoft Intermediate Language** (MSIL) intern
In Second Step 2) this will be converted to **Native Code** by CLR JIT Compiler. See the below Fig.



There are currently over 15 language compilers being built by Microsoft and other companies also producing the code that will execute under CLR.

The main function of Common Language Runtime (CLR) is to convert the Managed Code into native code and then execute the Program.

The Managed Code compiled only when it needed, that is it converts the appropriate instructions when each function is called.

That means all .NET programming languages uses the same representation for common Data Types , so Common Language Runtime (CLR) is a language-independent runtime environment . The Common Language Runtime (CLR) environment is also referred to as a managed environment, because during the execution of a program it also controls the interaction with the Operating System.

The CLR has the following key features:

1) **Exception Handling** - Exceptions are errors which occur when the application is executed.

   Examples of exceptions are:
   - If an application tries to open a file on the local machine, but the file is not present.
   - If the application tries to fetch some records from a database, but the connection to the database is not valid.
2) **Garbage Collection** - Garbage collection is the process of removing unwanted resources when they are no longer required.

   Examples of garbage collection are
   - A File handle which is no longer required. If the application has finished all operations on a file, then the file handle may no longer be required.
   - The database connection is no longer required. If the application has finished all operations on a database, then the database connection may no longer be required.
3) **Type Safety: -** CLR provide type safety by the CLS in this it will check type safety feature
4) **Thread Management: -** CLR also support multi-threading environment.

2. **.Net Framework Class Library** (FCL)

This is also called as Base Class Library and it is common for all types of applications i.e. the way you access the Library Classes and Methods in VB.NET will be the same in C#, and it is common for all other languages in .NET.

The following are different types of applications that can make use of .net class library.

1. Windows Application.
2. Console Application
3. Web Application.
4. XML Web Services.
5. Windows Services.

In short, developers just need to import the BCL in their language code and use its predefined methods and properties to implement common and complex functions like reading and writing to file, graphic rendering, database interaction, and XML document manipulation.

The below table provides a list each class of the base class library and a brief description of what they provide.

Base Class Library Namespace	Brief Description
System	Contains the fundamentals for programming such as the data types, console, math and arrays, etc.
System.Collections	Contains Lists, stacks, hashtables and dictionaries
System.ComponentModel	Provides licensing, controls and type conversion capabilities
System.Configuration	Used for reading and writing program configuration data
System.Data	Is the namespace for ADO.NET
System.Diagnostics	Provides tracing, logging, performance counters, etc. functionality
System.Drawing	Contains the GDI+ functionality for graphics support
System.Globalization	Supports the localization of custom programs
System.IO	Provides connection to file system and the reading and writing to data streams such as files
System.Net	Provides access to network protocols such as SSL, HTTP, SMTP and FTP
System.Resources	Used when localizing a program in relation to language support on web or form controls
System.Text	Provides the StringBuilder class, plus regular expression capabilities
System.Web	Namespace for ASP.NET capabilities such as Web Services and browser communication.
System.Windows.Forms	Namespace containing the interface into the Windows API for the creation of Windows Forms programs.
System.Xml	Provides the methods for reading, writing, searching and changing XML documents and entities.

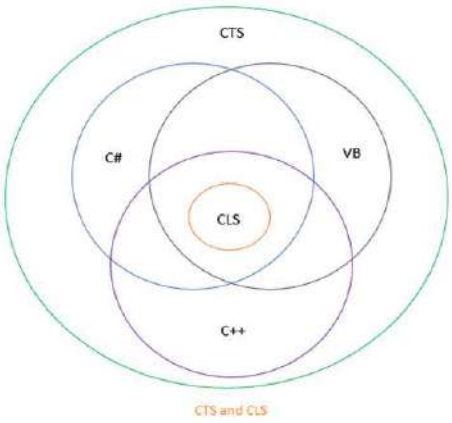
Common Type System (CTS) :

Common Type System (CTS) describes a set of data types, it ensure that objects written in different .Net supported languages can interact with each other.

The Common Language Runtime (CLR) can load and execute the source code written in any .Net language, only if the type is described in the Common Type System (CTS).

For example, C# has int Data Type and VB.Net has Integer Data Type. Hence a variable declared as int in C# or Integer in vb.net, finally after compilation, use the same structure Int32 from CTS.

Purpose of these is to support language independence in .NET. Hence it is called CTS.



4. Common Language Specification (CLS)

It is a sub set of CTS and it specifies a set of rules that needs to be satisfied by all language compilers targeting CLR. It helps in cross language inheritance and cross language debugging.

Common language specification Rules:

It describes the minimal and complete set of features to produce code that can be hosted by CLR. It ensures that products of compilers will work properly in .NET environment.

Sample Rules:

1. Representation of text strings
2. Internal representation of enumerations
3. Definition of static members and this is a subset of the CTS which all .NET languages are expected to support.
4. Microsoft has defined CLS which are nothing but guidelines that language to follow so that it can communicate with other .NET languages in a seamless manner.

Garbage Collection :

Garbage collection is the mechanisam to releasing memory from unused objects and components of the application. Language such as C++ do not have any garbage collection

system, therefor developers have to manually clean the memory. Visual Basic provides automatically Garbage collection system.

In .Net Framework, garbage collector is implemented as a separate thread. This thread will always be running at the back end.

Metadata :

Meatdata means data about data. Metadata in .Net is binary information which describes the characteristics of data. It contains information about description of the assembly, data types, members with their declaration, references, security permission, etc.

During the run time JIT uses both metadata & MSIL code and converts into native code.

Option Explicit Statement

When Option Explicit statement is set to ON, you must explicitly declare all variables using the Dim or ReDim statements. By default it in ON, so you need to declare all the variables before they are used.

When Option Explicit statement is set to OFF, all undeclared variables are treated as Object type.

Example :

```
' Force explicit variable declaration.
Option Explicit On
        ……
        ……
        Dim A As Integer
        A = 10
    ' The following assignment statement produces a COMPILER ERROR because
    ' the variable Square1 is not declared and Option Explicit is set to ON.
        Square1 = 10 ' It will causes ERROR
    'If Option Explicit is set to ON, then we must declare the variable before it is used.
```

Option Strict Statement

If Option Strict Statement is set to ON, Visual Basic prevents (disallows) implicit narrowing data type conversion. Visual Basic will not do automatically type conversion. Then Visual Basic will consider that an error.

If Option Strict Statement is set to OFF, Visual Basic will do automatically implicit narrowing data type conversion. By default it in OFF.

Example :

```
' Force explicit variable declaration.
Option Strict On
```

……
……
Dim I As Integer
Dim D As Double
D = 3.14159
I = D

‘A compilation ERROR occurs if such a narrowing conversion fails.
‘ It will disallows implicit conversions from Double To Integer
‘To solve above problem either you have to make **Option Strict Off** or you have to
‘do specific type conversion as follows.

I = Cint(D) ‘ Convert To Integer.

Option Compare Statement :

The Option Compare statement is used to specify the string comparison method, that is either Binary or Text for a class, module or structure. By default text comparison method is Binary.

Option Compare Binary
OR
Option Compare Text

Option Compare Binary, the following text sort order is produced..
A < B < E < Z < a < b < e < z
Option Compare Text, the following text sort order is produced.
(A=a) < (B=b) < (E=e) < (Z=z)

Example :

' Set the string comparison method to **Binary**

Option Compare Binary

MsgBox("AAA" < "aaa”) ‘It will return True, because A < a in Binary comparison

Example :

' Set the string comparison method to **Text**

Option Compare Text

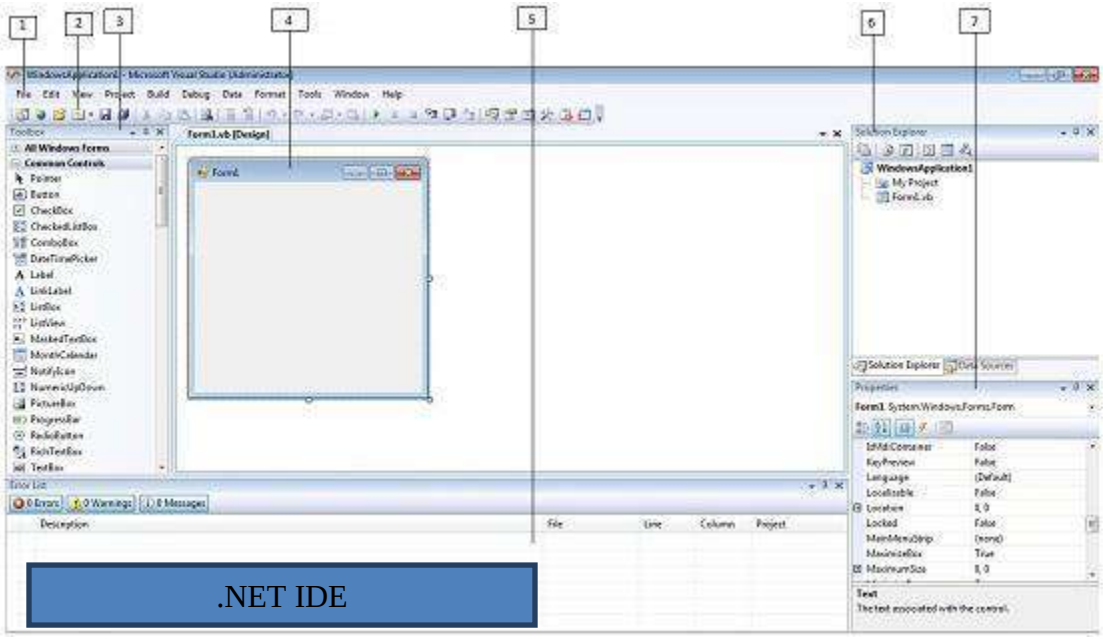
MsgBox("AAA" < "aaa") ‘It will return False, because A=a in Text comparison

UNIT-2 Programming in visual basic.net

IDE (integrated development environment):

An integrated development environment (IDE), also known as integrated design environment and integrated debugging environment, is a type of computer software suite that assists computer programmers to develop software.

The integrated development environment (IDE) is important in helping you create, run and debug any .Net programs or applications. You can consider VB.Net IDE as Microsoft Word and VB.Net programs as Word documents.



1. Menu Bar

It consists of menus that help you manipulate VB.Net programs in the project. The menus are listed from left to right as File, Edit, View, Project, Build, Debug, Tools, Window, and Help.

2. Standard Toolbar

Contains buttons that are shortcuts to some commonly used menu items.

3. Toolbox (Ctrl + Alt +X)

The window is very important in the VB.Net IDE. It contains control templates or components that are available for you to use. You can simply drag and drop any control from toolbox to your form

4. Forms Designer (Shift + F7)

We can drag and drop controls in this view of the form. We can also see some type of preview of our form

5. Output Window

The Output window is where many of the tools, including the compiler, send their output. Every time you start an application, a series of messages is displayed in the Output window. These messages are generated by the compiler, and you need not understand them at this point. If the Output window is not visible, choose View > Other Windows > Output from the menu.

6. Solution Explorer (Ctrl + Alt +L)

The window contains a Windows Explorer-like tree view of all the customizable forms and general code (modules) that make up a VB.Net application. The Solution Explorer provides you with an organized view of your project and program files associated with the project. Select the Solution Explorer on the View menu when you cannot find the Solution Explorer in your IDE.

7. Properties Window (F4)

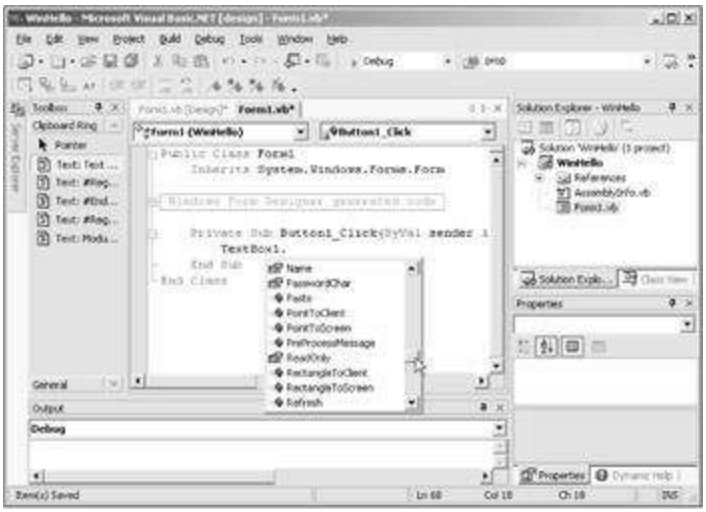
Window The properties window displays the properties for a form or a control. Properties describe attributes such as size, color, and font of a control. Each form or control has its own set of properties. When you click on a control or the form, the properties will be listed in the properties window. There are two columns in the properties window. The first column lists the property names and the second column shows the current value of the property. The value can be changed at the design phase of the form or through the program code. There are lots of properties associated with controls.

Title Bar -It shows the title of the VB.Net project you are currently working on. The default project title is the project name you have specified when you create a new project. If you would like to change the project name or title to other name, you can change it through Project -> Project Properties.

Code editor window (F7) –where we can write the coding of the form or class etc.

IntelliSense

One useful feature of VB .NET code designers is Microsoft's *IntelliSense*. IntelliSense is what's responsible for those boxes that open as you write your code, listing all the possible options and even completing your typing for you. IntelliSense is one of the first things you encounter when you use VB .NET, and you can see an example in Figure 1.25, where I'm looking at all the members of a text box object.

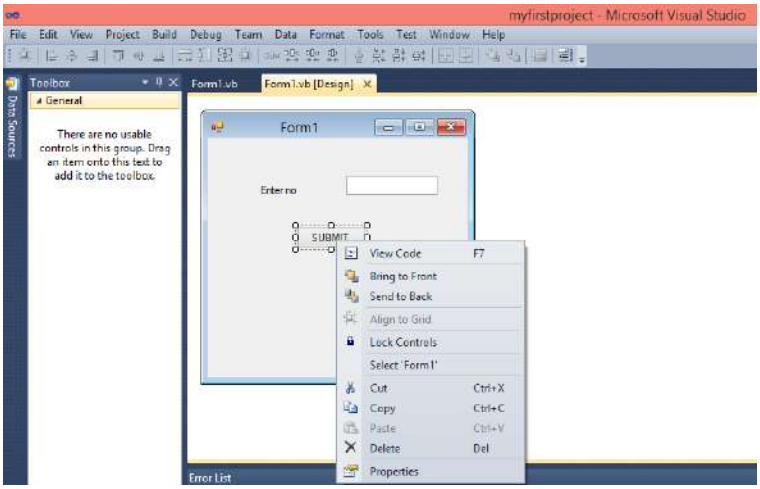


Context Menu

It contains shortcut for frequently performed actions.

To open context menu , select any object and click the right mouse button

Context menu will open according to the selected object



What is Auto Hide icon?
This is a new feature in Visual Studio that hides away the windows not currently in use.The reference to "not currently *using*" suggests that the windows or panels are not in focus; however, they are not closed down. Thus, as you change windows like going from Solution Explorer to Help the one you are leaving slides closed.

Variable:

A variable is something that is used in a program to store data in memory.

Variable in VB.Net has a specific type, which determines the size and layout of the variable's memory; the range of values that can be stored within that memory.

The **Dim** statement is used for variable declaration and storage allocation for one or more variables.

Example: **Dim** Counter **as** Integer

Data Types in VB.Net

VB.Net provides a wide range of data types. The following table shows all the data types available −

Data Type	**Storage Allocation**	**Value Range**
Boolean	Depends on implementing platform	**True** or **False**
Byte	1 byte	0 through 255 (unsigned)
Char	2 bytes	0 through 65535 (unsigned)
Date	8 bytes	0:00:00 (midnight) on January 1, 0001 through 11:59:59 PM on December 31, 9999
Decimal	16 bytes	0 through +/- 79,228,162,514,264,337,593,543,950,335 (+/- 7.9...E+28) with no decimal point; 0 through +/- 7.9228162514264337593543950335 with 28 places to the right of the decimal
Double	8 bytes	-1.79769313486231570E+308 through -4.94065645841246544E-324, for negative values 4.94065645841246544E-324 through 1.79769313486231570E+308, for positive values
Integer	4 bytes	-2,147,483,648 through 2,147,483,647 (signed)
Long	8 bytes	-9,223,372,036,854,775,808 through 9,223,372,036,854,775,807(signed)
Object	4 bytes on 32-bit platform 8 bytes on 64-bit platform	Any type can be stored in a variable of type Object
SByte	1 byte	-128 through 127 (signed)
Short	2 bytes	-32,768 through 32,767 (signed)

Single	4 bytes	-3.4028235E+38 through -1.401298E-45 for negative values; 1.401298E-45 through 3.4028235E+38 for positive values
String	Depends on implementing platform	0 to approximately 2 billion Unicode characters
UInteger	4 bytes	0 through 4,294,967,295 (unsigned)
ULong	8 bytes	0 through 18,446,744,073,709,551,615 (unsigned)
User-Defined	Depends on implementing platform	Each member of the structure has a range determined by its data type and independent of the ranges of the other members
UShort	2 bytes	0 through 65,535 (unsigned)

Data Type Conversion Function

CBool(expression)- Converts the expression to Boolean data type.

```
Dim a, b, c As Integer
Dim check As Boolean
a = 5
b = 5
' The following line of code sets check to True.
check = CBool(a = b)
c = 0
' The following line of code sets check to False.
check = CBool(c)
```

CByte(expression)- Converts the expression to Byte data type.

```
Dim aDouble As Double
```

```
Dim aByte As Byte
aDouble = 125.5678
' The following line of code sets aByte to 126.
aByte = CByte(aDouble)
```

CChar(expression) - Converts the expression to Char data type.

```
Dim aString As String
Dim aChar As Char
' CChar converts only the first character of the string.
aString = "BCD"
' The following line of code sets aChar to "B".
aChar = CChar(aString)
```

CDate(expression) - Converts the expression to Date data type

```
Dim aDateString, aTimeString As String
Dim aDate, aTime As Date
aDateString = "February 12, 1969"
aTimeString = "4:35:47 PM"
' The following line of code sets aDate to a Date value.
aDate = CDate(aDateString)
' The following line of code sets aTime to Date value.
aTime = CDate(aTimeString)
```

CDbl(expression)- Converts the expression to Double data type.

```
Dim aDec As Decimal
Dim aDbl As Double
```

```
' The following line of code uses the literal type character D to make aDec a Decimal.
aDec = 234.456784D
' The following line of code sets aDbl to 1.9225456288E+1.
aDbl = CDbl(aDec * 8.2D * 0.01D)
```

CDec(expression)- Converts the expression to Decimal data type.

```
Dim aDouble As Double
Dim aDecimal As Decimal
aDouble = 10000000.0587
' The following line of code sets aDecimal to 10000000.0587.
aDecimal = CDec(aDouble)
```

CInt(expression)- Converts the expression to Integer data type.

```
Dim aDbl As Double
Dim anInt As Integer
aDbl = 2345.5678
' The following line of code sets anInt to 2346.
anInt = CInt(aDbl)
```

CLng(expression)- Converts the expression to Long data type.

```
Dim aDbl1, aDbl2 As Double
Dim aLng1, aLng2 As Long
aDbl1 = 25427.45
aDbl2 = 25427.55
' The following line of code sets aLng1 to 25427.
aLng1 = CLng(aDbl1)
```

```
' The following line of code sets aLng2 to 25428.
aLng2 = CLng(aDbl2)
```

CObj(expression)- Converts the expression to Object type.

```
Dim aDouble As Double
Dim anObject As Object
aDouble = 2.7182818284
' The following line of code sets anObject to a pointer to aDouble.
anObject = CObj(aDouble)
```

CShort(expression)- Converts the expression to Short data type.

```
Dim aByte As Byte
Dim aShort As Short
aByte = 100
' The following line of code sets aShort to 100.
aShort = CShort(aByte)
```

CSng(expression)- Converts the expression to Single data type.

```
Dim aDouble1, aDouble2 As Double
Dim aSingle1, aSingle2 As Single
aDouble1 = 75.3421105
aDouble2 = 75.3421567
' The following line of code sets aSingle1 to 75.34211.
aSingle1 = CSng(aDouble1)
' The following line of code sets aSingle2 to 75.34216.
```

```
aSingle2 = CSng(aDouble2)
```

CStr(expression)- Converts the expression to String data type.

```
Dim aDouble As Double
Dim aString As String
aDouble = 437.324
' The following line of code sets aString to "437.324".
aString = CStr(aDouble)
```

CTYPE Function

It used to convert one type to another type. Instead of remember all conversion function, remember only CTYPE function.

Syntax: CType(expression,typename)

Example:

```
dim a as integer
‘dim b as integer

  b=66.7
 a=CType(b,Integer)
```

Operators:

Operators	Description
Arithmetic Operators	+,-,*,/,\,MOD,^
Assignment Operators	=,^=,*=,/=,\=,+=,-=,<<=,>>=,&=
Comparison Operators	>,<,=,<>,>=,<=
Concatenation Operators	+,&
Logical/Bitwise Operators	And,Or,Not,Xor
Bit Shift Operators	>>,<<

Boxing And Unboxing

Boxing

- Boxing is a mechanism in which value type is converted into reference type.
- It is implicit conversion process in which object type (super type) is used.
- In this process type and value both are stored in object type

Unboxing

- Unboxing is a mechanism in which reference type is converted into value.
- It is explicit conversion process.

```
Dim i As Integer = 10
    Dim j As Integer
    ' boxing
    Dim o As Object
    o = i
    'unboxing
    j = CInt(o)
```

Constants and Enumerations

The **constants** refer to fixed values that the program may not alter during its execution. These fixed values are also called literals.

Constants can be of any of the basic data types like an integer constant, a floating constant, a character constant, or a string literal. There are also enumeration constants as well.

In VB.NET, **const** is a keyword that is used to declare a variable as constant. The Const statement can be used with module, structure, procedure, form, and class.

Syntax:

```
Const constname As datatype = value
```

Example: Const num As Integer = 10

Enumerations

Enum is a keyword known as Enumeration. **Enumeration** is a user-defined data type used to define a related set of constants as a list using the keyword **enum** statement. It can be used with module, structure, class, and procedure. For example, month names can be grouped using Enumeration.

Syntax:

```
Enum enumeration name [ As Data type ]
' enumeration data_list or Enum member list
End Enum
```

In the above syntax, the **enumeration name** is a valid identifier name, and the data type can be mentioned if necessary, the enumeration **data_list** or **Enum member list** represents the set of the related constant member of the Enumeration.

Declaration of Enumerated Data

Following is the declaration of enumerated data using the Enum keyword in VB.NET Programming language:

```
Enum Fruits
Banana
Mango
Orange
Pineapple
Apple
End Enum
```

Here, **Fruits** is the name of enumerated data that contains a list of fruits called **enumeration** lists.

When we are creating an Enumerator constant data list, by default, it assigns a value 0 to the first index of the enumerator. Each successive item in the enumerator list is increased by 1.

```
Enum Fruits
Banana = 5
Mango
Orange
Pineapple
Apple
End Enum
```

```
Msgbox(fruits.mango)
```

Statements

A **statement** is a complete instruction in Visual Basic programs. It may contain keywords, operators, variables, literal values, constants and expressions.

Statements could be categorized as −

- **Declaration statements** − these are the statements where you name a variable, constant, or procedure, and can also specify a data type.
- **Executable statements** − these are the statements, which initiate actions. These statements can call a method or function, loop or branch through blocks of code or assign values or expression to a variable or constant. In the last case, it is called an Assignment statement.

The Option Statement

There is one statement that is very important to know when making an application- Option statement

There are three option statements, these are as follows:

(a) Option Explicit:

It has two modes:

1. On
2. Off

Any VB.net program has 'On' mode of option explicit by default. If program has 'Option explicit On' statement than it requires all variables have proper deceleration otherwise it gives compile time error and another mode is 'Off', if we use 'Option explicit Off' statement than vb.net create variable declaration automatically and program does not give an error

We can take an example for better understanding:

```
Option Explicit On
Public Class Form1
   Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) H
andles MyBase.Load
      TempString = "Hello"
      MessageBox.Show(TempString)
   End Sub
End Class
```

Above program gives an error '**Name TempString is not declared**'

```
Option Explicit Off
Public Class Form1
   Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) H
andles MyBase.Load
      TempString = "Hello"
      MessageBox.Show(TempString)
   End Sub
End Class
```

Above program run continue without error

Note: For better coding it is recommended to declare variables with Dim keyword and data type.

(b) Option Compare:

This statement also has two modes

1. Binary
2. Text

Option Compare is Binary by default; we can change string comparison method by set the text or Binary see example:

```
Public Class Form1
```

```
   Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) H
andles MyBase.Load
      If "Hello" = "HELLO" Then
         MessageBox.Show("True")
      Else
         MessageBox.Show("False")
      End If
   End Sub
End Class
```

In above program if we use Option Compare Binary,messagebox show 'false' and if we use 'Text' mode than messagebox show 'True. that means when we set Option Compare to Text we can able compare string with Case insensitive comparision.

(c) Option Strict:

It has also two modes:

1. On
2. Off

Option Strict Off is the default mode. When you assign a value of one type to a variable of another type ,Visual Basic will consider that an error if this option is on and there is any possibility of data loss.

```
Option Strict On
Public Class Form1
   Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) H
andles MyBase.Load
      Dim tempInt As Integer = 2010
      MessageBox.Show(tempInt)
   End Sub
End Class
```

4)Option Infer Statement

Enables the use of local type inference in declaring variables.

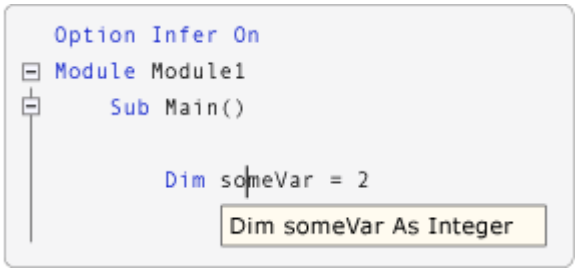
Syntax

```
Option Infer { On | Off }
```

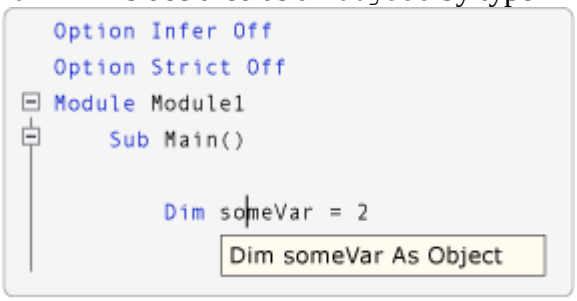
When you set `Option Infer` to `On`, you can declare local variables without explicitly stating a data type. The compiler infers the data type of a variable from the type of its initialization expression.

`Option Infer` is turned on. The variable in the declaration `Dim someVar = 2` is declared as an integer by type inference.

The following screenshot shows IntelliSense when Option Infer is on:



In the following illustration, `Option Infer` is turned off. The variable in the declaration `Dim someVar = 2` is declared as an `Object` by type inference. In this example,



[!NOTE] When a variable is declared as an `Object`, the run-time type can change while the program is running. Visual Basic performs operations called *boxing* and *unboxing* to convert between an `Object` and a value type, which makes execution slower.

Type Checking Function:

VB.NET provides number of data verification or data type checking function as below:

IsDate()

- Returns True if the value of variable is date value; Otherwise, it returns False

```
DimMyVar, MyCheck

MyVar = "04/28/2014"' Assign valid date value.
MyCheck = IsDate(MyVar)    ' Returns True.
MsgBox(MyCheck)

MyVar = "April 28, 2014"' Assign valid date value.
MyCheck = IsDate(MyVar)    ' Returns True.
MsgBox(MyCheck)

MyVar = "13/32/2014"' Assign invalid date value.
MyCheck = IsDate(MyVar)    ' Returns False.
MsgBox(MyCheck)

MyVar = "04.28.14"' Assign valid time value.
MyCheck = IsDate(MyVar)    ' Returns True.
```

```
MsgBox(MyCheck)

    MyVar = "04.28.2014"' Assign invalid time value.
    MyCheck = IsDate(MyVar)    ' Returns False.
MsgBox(MyCheck)
```

IsNothing()

- Returns True if the object variable that currently has no assigned value; Otherwise, it returns False

```
  Dim objtemp As Object
    Dim bolans As Boolean

bolans = IsNothing(objtemp)
MsgBox(bolans)  ' return true

objtemp = "dolly"
bolans = IsNothing(objtemp)
MsgBox(bolans)  ' return false
```

IsNumeric()

- Returns True if the value is numeric; Otherwise returns false

```
  DimMyVar, MyCheck
  MyVar = "53"' Assign value.
  MyCheck = IsNumeric(MyVar)    ' Returns True.
MsgBox(MyCheck)

  MyVar = "459.95"' Assign value.
  MyCheck = IsNumeric(MyVar)    ' Returns True.
MsgBox(MyCheck)

  MyVar = "45 Help"' Assign value.
  MyCheck = IsNumeric(MyVar)    ' Returns False.
MsgBox(MyCheck)
```

IsArray()

- Tests whether an object variable points to an array

```
Dim s() As Integer = {1, 2}
Dim t As Object
t = s
```

```
MsgBox(IsArray(t)) ' return true

Dim strArr() As String
Console.WriteLine(IsArray(strArr)) ' return false An uninitialized array
```

String Functions

String functions are mainly used to manipulate the string .

1. The Len Function

The length function is used to find out the number of characters in any given string.

Syntax

Len(*string*)

2. The Mid function

The mid function is used to Return a substring containing a specified number of characters from a string.

Syntax

Mid (string, start[, length])

string - String expression from which characters are returned.

start - Long. Character position in string at which the part to be taken begins.

length - Length is Optional. Number of characters to return.

3. The Left Function

The Left function extract the left portion of a string.

Syntax

Left("string", n)

4. The Right Function

The Right function extract the right portion of a string.

SyntaxRight("string", n)

5. The Replace Function

The replace function is used to replacing some text in a string with some other text.

Syntax

Replace(string, searchtext, replacetext)

6. The Trim function

The trim function trims the empty spaces on both side of the String.

Syntax

Trim ("String")

7. The Ltrim Function

The Ltrim function trims the empty spaces of the left portion of the string.

Syntax

Ltrim("string")

8. The Rtrim Function

The Rtrim function trims the empty spaces of the Right portion of the string.

Syntax

Rtrim("string")

9.The Ucase and the Lcase Functions

The Ucase function converts all the characters of a string to capital letters. On the other hand, the Lcase function converts all the characters of a string to small letters.

Syntax

Ucase("string")

Lcase(“string”)

Date Functions

DateAdd()

It is used to returns a date with a date,time value added with a specified time intreval.

Syntax: DateAdd(interval, number, date)

A date is added after an intreval of 10 days to the current date value.

DateDiff()

It is used to return a long value specifying the number of time intrevals between the specified date values.

Syntax: Datediff(interval,date1,date2)

The time intrevals between two same dates of different years are found using the 'DateDiff()'.

DatePart()

returns an integer value containing the specified component of the Date value.

Syntax: DatePart(interval,date)

the date value entered is in the **mm:dd:yy** format

DateSerial()

returns an date value for the specified year, month and day with the time set to the midnight.

If the month value is '0' or '-1' the month december or november of the previous year is taken.

If the month value is '1', january month of the calculated year is taken, if '13' january of the following year is taken.

If the Day value is '1' refers to the first day of the calculated month, '0' for the last day of previous month, '-1' the penultimate day of the previous month.

Syntax:DateSerial(Year,Month,Day)

DateValue()

returns an date value containing the date information as a string

Syntax:DateValue(Date)

the date information alone is displayed as a String using the **DateValue** function

IsDate()

checks if the given expression is a valid date and returns a boolean true or false.

Syntax:IsDate(Expession)

If the date given as the argument is valid, the **IsDate** function returns **True**.

Today()

Return today's date

Syntax:Today()

Now()

Return today's date with time

Syntax:now()

Month()

returns the month of the year as an integer value in the range of 1-12.
Syntax:Month(Date)
the month for the given date is returned using the Month function.

TimeSerial()
returns an date value with the specified hour, minute, second with the date information is set to January 1 of year 1.
Syntax:Timeserial(hour, minute, second)

Format()

Returns a string formatted according to instructions contained in a format string expression.

Syntax: format(date,format)

Example:

DateAdd

```
MsgBox("10 Days after the current date is::" &DateAdd(DateInterval.Day, 10, Now))
```

DateDiff

```
    Dim d1 As Date = #2/4/2009#
    Dim d2 As Date = #2/4/2010#
    Dim res As Long
    res = DateDiff(DateInterval.Day, d1, d2)
MsgBox("The number of time intrevals between the dates is::" & res)
```

DatePart

```
MsgBox("The date part of '01/10/2010' is::" &DatePart("d", "01/10/2010"))
```

DateSerial

```
    Dim a As Date
    a = DateSerial(2010, 2, 21)
MsgBox(a)
```

DateValue

```
MsgBox("Date information as a String is::"&DateValue("5/10/2010 12:00:01 AM"))
```

TimeSerial

```
MsgBox("Time displayed using Timeserial() is:: " &TimeSerial(4, 30, 23))
```

isdate

```
    Dim curdat As Date
curdat = "5/31/2010"
MsgBox("Is '5/31/2010' a valid date::" &IsDate(curdat))
```

Today

```
    Dim curdat As Date
curdat = Today()
MsgBox(curdat)
```

now

```
    Dim curdat As Date
curdat = Now()
MsgBox(curdat)
```

Month

```
    Dim dat As Date
dat = "6/23/2010"
MsgBox("Month value of the given date is::" & Month(dat))
```

Format

```
MsgBox(Format(Now, "M-d-yy"))
MsgBox(Format(Now, "MM-dd-yyyy"))
MsgBox(Format(Now, "MMMM-d-yyy-  dddd"))
MsgBox(Format(Now, "hh:mm:sstt"))
```

Methods

Methods are simply member procedures built into the class. In Visual Basic .NET there are two types of methods Functions and Sub Procedures. Methods help us to handle code in a simple and organized fashion. Functions return a value, but Sub Procedures does not return any value. Methods are basically a series of statements that are executed when called.
Module:

A `Module` statement defines a reference type available throughout its namespace. A *module* (sometimes called a *standard module*) is similar to a class but with some important distinctions. Every module has exactly one instance and does not need to be created or assigned

to a variable. Modules do not support inheritance or implement interfaces. Notice that a module is not a *type* in the sense that a class or structure is — you cannot declare a programming element to have the data type of a module.

You can use `Module` only at namespace level. This means the *declaration context* for a module must be a source file or namespace, and cannot be a class, structure, module, interface, procedure, or block. You cannot nest a module within another module, or within any type.

```
Public Module thisModule
   Sub Main()
      Dim userName As String = InputBox("What is your name?")
      MsgBox("User name is " & userName)
   End Sub
   ' Insert variable, property, procedure, and event declarations.
End Module
```

A module has the same lifetime as your program. Because its members are all `Shared`, they also have lifetimes equal to that of the program.

Modules default to Friend access. You can adjust their access levels with the access modifiers. All members of a module are implicitly `Shared`.

Rules

- **Modifiers.** All module members are implicitly Shared. You cannot use the Shared keyword when declaring a member, and you cannot alter the shared status of any member.
- **Inheritance.** A module cannot inherit from any type other than Object, from which all modules inherit. In particular, one module cannot inherit from another.

  You cannot use the Inherits Statement in a module definition, even to specify Object.

- **Default Property.** You cannot define any default properties in a module. For more information, see Default.

Procedure

A procedure is a group of statements that together perform a task when called. After the procedure is executed, the control returns to the statement calling the procedure. VB.Net has two types of procedures −

- Sub procedures or Subs
- Functions

Functions return a value, whereas Subs do not return a value.

1. **Sub procedures or Subs or Sub Routine**
   - A Sub procedure is a series of Visual Basic statements enclosed by the Sub and End Substatements. The Sub procedure performs a task and then returns control to the calling code, but it does not return a value to the calling code.
   - Each time the procedure is called, its statements are executed, starting with the first executable statement after the Sub statement and ending with the first End Sub, Exit Sub, or Return statement encountered.
   - You can define a `Sub` procedure in modules, classes, and structures. By default, it is `Public`, which means you can call it from anywhere in your application that has access to the module, class, or structure in which you defined it

     A `Sub` procedure can take arguments, such as constants, variables, or expressions, which are passed to it by the calling code.

The **Sub** statement is used to declare the name, parameter and the body of a sub procedure. The syntax for the Sub statement is −

```
[Modifiers] Sub SubName [(ParameterList)]
  [Statements]
End Sub
```

Where,

- ***Modifiers*** − specify the access level of the procedure; possible values are - Public, Private, Protected, Friend, Protected Friend and information regarding overloading, overriding, sharing, and shadowing.
- ***SubName*** − indicates the name of the Sub
- ***ParameterList*** − specifies the list of the parameters

Sub tellOperator(ByVal task As String) Dim stamp AsDate stamp = TimeOfDay() MsgBox("Starting "& task &" at "&CStr(stamp)) EndSub	tellOperator("file update")

2. Functions

- A Function procedure is a series of Visual Basic statements enclosed by the Function and End Function statements. The Function procedure performs a task and then returns control to the calling code. When it returns control, it also returns a value to the calling code.
- Each time the procedure is called, its statements run, starting with the first executable statement after the Function statement and ending with the first End Function, Exit Function, or Returnstatement encountered.

- You can define a Function procedure in a module, class, or structure. It is Public by default, which means you can call it from anywhere in your application that has access to the module, class, or structure in which you defined it.
- A Function procedure can take arguments, such as constants, variables, or expressions, which are passed to it by the calling code.

The Function statement is used to declare the name, parameter and the body of a function. The syntax for the Function statement is −

```
[Modifiers] Function FunctionName [(ParameterList)] As ReturnType
   [Statements]
End Function
```

Where,

- ***Modifiers*** − specify the access level of the function; possible values are: Public, Private, Protected, Friend, Protected Friend and information regarding overloading, overriding, sharing, and shadowing.
- ***FunctionName*** − indicates the name of the function
- ***ParameterList*** − specifies the list of the parameters
- ***ReturnType*** − specifies the data type of the variable the function returns

FunctionFindMax(ByVal num1 AsInteger, ByVal num2 AsInteger) AsInteger
' local variable declaration */
Dim result AsInteger
 	
If (num1 > num2) Then
result = num1
Else
result = num2
EndIf
FindMax = result
EndFunction
	Calling function
	
	MsgBox("the maximum number is "&FindMax(10, 5))

Optional argument

Sub notify(ByVal company AsString, OptionalByValdesgAsString = "manager") Ifdesg = "manager"Then MsgBox("i am manager")	Call notify("abc") Call notify("abc", "worker")

```
Else
MsgBox("i am not manager")
EndIf
EndSub
```

Function overloading

Function overloading is where two or more functions can have the same name but different parameters.

Function overloading can be considered as an example of compile time polymorphism feature in Oop's.

```
PublicOverloadsFunction add(ByVal a AsInteger, ByVal b AsInteger)
MsgBox("You are in function add(a,b)")
Return a + b
EndFunction

PublicOverloadsFunction add(ByVal a AsInteger, ByVal b AsInteger, ByVal c AsInteger)
MsgBox("You are in function add(a, b, c)")
Return a + b + c
EndFunction
```

```
MsgBox(add(4, 2))

MsgBox(add(4, 5, 1))
```

ByVal&ByRef Methods for passing arguments to Subroutine & Functions

- The **ByVal** keyword indicates that an argument is passed in such a way that the called procedure or property cannot change the value of a variable passed as the argument in the calling code.
- The **ByRef** keyword indicates that an argument is passed in such a way that the called procedure can change the value of a variable passed as the argument in the calling code.

```
Dim value AsInteger = 1
```

Pass by value

```
Sub Example1(ByVal test AsInteger)
    test = 10
EndSub
```

```
' The integer value doesn't change here when passed ByVal.

    Example1(value)
MsgBox("by val "& value)
```

Pass by reference

```
Sub Example2(ByRef test AsInteger)
    test = 10
EndSub
```

```
' The integer value DOES change when passed ByRef.

    Example2(value)
MsgBox("by ref "& value)
```

Output : 1	Output :10

VB.Net Arrays

An array is a data structure used to store elements of the same data type. The elements are ordered sequentially with the first element being at index 0 and the last element at index n-1, where n is the total number of elements in the array.

Declaring an Array

To declare an array in VB.NET, you need to specify the data type of the elements and the size of the array. Here's an example of declaring an array of integers:

```
Dim numbers(4) As Integer
```

Initializing an Array

After declaring an array, you can initialize it with values using various methods.

Inline initialization:

```
Dim numbers() As Integer = {1, 2, 3, 4, 5}
```

Types of Arrays in VB.net

There are two **types of Arrays in VB.net** namely:

1. Fixed Size Array in VB.net

A **Fixed Size Array In VB.net** is used to store a set number of elements in memory. This indicates that the array's size cannot be altered because the number of elements we specified in the array declaration will not change during the defining of the elements.
For example, we need to hold only 5 names in an array; it can be defined and initialized in the array such as:

```
Dim names( 0 to 4) As String
names(0) = "Adones"
names(1) = "Glenn"
names(2) = "Jaymar"
names(3) = "Paul"
names(4) = "Jude"
```

The above representation of the **fixed array** is that we have defined a **string array** named **0** to **4**, which stores all the elements in the array from **0** to index **4**.

2.Multidimensional Array

In VB.NET, a multidimensional array is useful for storing more than one dimension in a tabular form, such as rows and columns. The multidimensional array support two or three dimensional in VB.NET.

Declaration of Multidimensional Array
Declaration of two-dimensional array
Dim twoDimenArray As Integer(,) = New Integer(3, 2) {}
Or Dim arr(5, 3) As Integer
Representation of Three Dimensional array
Dim arrThree(2, 4, 3) As Integer
Or Dim arr1 As Integer(, ,) = New Integer(5, 5, 5) { }

In the above representation of multidimensional, we have created 2-dimensional array **twoDimenArray with 3 rows and 2 columns and** 3-dimensional array with three dimensions 2, 4, and 3.

Initialization of Multidimensional Array

The following ways to initialize the multidimensional array:

```
' Initialization of Two Dimensional Array
Dim intArray As Integer( , ) = New Integer( 3, 2) { {4, 5}, {2, 3}, {6, 7} }
Dim intArray( , ) As Integer = {  {5, 4}, {3, 2}, {4, 7} }
' Initialization of Three Dimensional Array
Dim threeDimen(3, 3, 2 ) As Integer = { {{1, 3, 2}, {2, 3, 4}}, {{5, 3, 6}, {3, 4, 5}}, {{1, 2, 2},{
5, 2, 3} }}
```

2.Dynamic Arrays In VB.net

Dynamic Arrays In VB.net are those that can be **resized** and **dimensioned** in accordance with the requirements of the software.
Using the **ReDim statement**, a **dynamic array** can be declared.
Syntax for **ReDim statement**:
ReDim [Preserve] arrayname(subscripts)
Example program of **Dynamic Arrays in VB.net**:

```
Module arrayApl
  Sub Main()
    Dim marks() As Integer
    Dim i As Integer

    ReDim marks(2)
    marks(0) = 85
    marks(1) = 75
    marks(2) = 90

    ReDim Preserve marks(10)
    marks(3) = 80
    marks(4) = 76
    marks(5) = 92
    marks(6) = 99
```

```
      marks(7) = 79
      marks(8) = 75

      For i = 0 To 10
        Console.WriteLine(i & vbTab & marks(i))
      Next i
      Console.ReadKey()
    End Sub
End Module
```

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

```
0 85
1 75
2 90
3 80
4 76
5 92
6 99
7 79
8 75
9 0
10 0
```

Jagged Array in VB.net

A **Jagged Array in VB.net** is an **array** whose components are arrays of various shapes and sizes.

A jagged array can store arrays instead of a specific data type value and is occasionally referred to as an "**array of arrays**".

The following code shows declaring a **Jagged Array in VB.net** named ***scores*** of Integers:

```
'Declaring a Jagged Array'

Dim scores As Integer()() = New Integer(5)(){}
```

Example Program of Jagged Array in VB.net

The following example illustrates using a jagged array:

```
Module arrayApl
  Sub Main()
    'a jagged array of 5 array of integers
    Dim a As Integer()() = New Integer(4)() {}
    a(0) = New Integer() {0, 0}
    a(1) = New Integer() {1, 2}
    a(2) = New Integer() {2, 4}
    a(3) = New Integer() {3, 6}
    a(4) = New Integer() {4, 8}
    Dim i, j As Integer
```

```
      ' output each array element's value

      For i = 0 To 4
        For j = 0 To 1
          Console.WriteLine("a[{0},{1}] = {2}", i, j, a(i)(j))
        Next j
      Next i
      Console.ReadKey()
    End Sub
End Module
```

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

```
a[0,0] = 0
a[0,1] = 0
a[1,0] = 1
a[1,1] = 2
a[2,0] = 2
a[2,1] = 4
a[3,0] = 3
a[3,1] = 6
a[4,0] = 4
a[4,1] = 8
```

Conditional statement

Visual Basic allows you to test conditions and perform different operations depending on the results of that test. You can test for a condition being true or false, for various values of an expression, or for various exceptions generated when you execute a series of statements. The decision statements supported by Visual Basic include:

- **If...Then**

if *condition* then
 	
Statement1
 	
Statement2
end if
	Dim x AsInteger
	x = 10
	If (x Mod 2) = 0 Then
	MsgBox("x is an even number")
	EndIf

- **If...Then...Else**

if *condition* then
 	
Statement1
 	
statement2
	Dim x AsInteger
	x = 10
	If (x Mod 2) = 0 Then
	MsgBox("x is an even number")
	Else
	MsgBox("x is an odd number")

else
Statement3
statement4
end if
	EndIf

- **If...Then...ElseIf…Then..Else**

If *condition1* then
 	
Statements…
 	
Else If *condition*2 then
 	
Statements…
 	
Else
Statements…
 	
End If
	
	Dim x AsInteger
	x = InputBox(x)
	If x > 0 Then
	MsgBox("It is positive.")
	
	ElseIf x < 0 Then
	MsgBox("It is negative.")
	Else
	MsgBox("It is zeo.")
	EndIf

- **Select...Case**

Select case variable case val1 statements case val2 statements case val3 statements …… case else statements End select	 Dim x AsInteger x = InputBox("Please enter your number from 1 to 3", x) Select Case x Case 1 MsgBox("You entered 1", x) Case 2 MsgBox("You entered 2") Case 3 MsgBox("You entered 3") Case Else MsgBox("Invalid!") EndSelect

Loop Constructs

Loop structures allow you to execute one or more lines of code repetitively. You can repeat the statements until a condition is true, until a condition is false, a specified number of times, or once for each object in a collection. The loop structures supported by Visual Basic include:

- **While**

While condition [statements] [Continue While] [statements] [Exit While] [statements] End While	Dim index AsInteger = 0 While index <= 10 MsgBox(index.ToString&" ") index += 1 EndWhile ' Output: 0 1 2 3 4 5 6 7 8 9 10

- **Do...Loop**

Do [statements] [Continue Do] [statements] [Exit Do] [statements] Loop { While | Until } condition	Dim index AsInteger = 0 Do MsgBox(index.ToString&" ") index += 1 LoopUntil index > 10 ' Output: 0 1 2 3 4 5 6 7 8 9 10

- **For...Next**

For counter [As datatype] = start To end [Step step] [statements] [Continue For] [statements] [Exit For] [statements] Next [counter]	For index AsInteger = 1 To 5 MsgBox(index.ToString&" ") Next ' Output: 1 2 3 4 5

- **For Each...Next**

For Each element [As datatype] In group [statements] [Continue For] [statements]	' Create a list of strings by using a ' collection initializer. DimlstAsNewList(OfString) _ From {"abc", "def", "ghi"}

[Exit For] [statements] Next [element]	' Iterate through the list. ForEach item AsStringInlst MsgBox(item &" ") Next 'Output: abcdefghi

What is an Exit Statement

The **Exit** statement allows you to exit directly from any decision structure, loop, or procedure. It immediately transfers execution to the statement following the last control statement. The syntax for the **Exit** statement specifies which type of control statement you are transferring out of. The following versions of the **Exit** statement are possible:

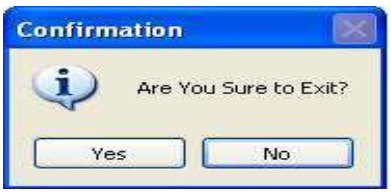
- **Exit Select**
- **Exit Try**
- **Exit Do**
- **Exit While**
- **Exit For**

You can also exit directly from a **Function**, **Sub**, or **Property** procedure; the syntax is similar to that of **Exit For** and **Exit Do**:

- **Exit Sub**
- **Exit Function**
- **Exit Property**

Message Box

The show method of MessageBox is used to display User Specific message in a Dialog Box and waits for the user to click a button.It returns an integer value indicating which button is click by user.
MessageBox is shown in the figure below:



MessageBox.Show (Text As String, Caption As String, Buttons As System.Windows.Forms.MeesageBoxButtons, Icon As System.Windows.Forms.MessageBoxIcon) As System.Windows.Forms.DialogResult
Here,
(1) Text is a compulsory argument. The String that you specify as a Text will display as a message in the Dialog Box.
(2) Caption is a compulsory argument. The String that you specified as a caption will be display in the title bar of the Dialog Box.
(3) Buttons is used to specify type of buttons to display in the message box.
(4) Icon is used to specify type of icon to display in the message box.

Possible values for Button argument are

MessageBoxButtons.OKOnly	Display OK Button.
MessageBoxButtons.OKCancel	Display OK and Cancel Button.
MessageBoxButtons.AbortRetryIgnore	Display Abort, Retry and Ignore Button.
MessageBoxButtons.YesNoCancel	Display Yes, No and Cancel Button.
MessageBoxButtons.YesNo	Display Yes and No Button.
MessageBoxButtons.CancelRetry	Display Cancel and Retry Button.

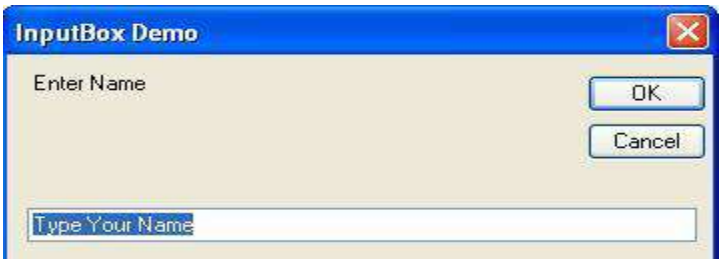
Possible values for Icon argument are:

MessageBoxIcon.Critical	Display Critical icon.
MessageBoxIcon.Question	Display Question icon.
MessageBoxIcon.Exclamation	Display Exclamation icon.
MessageBoxIcon.Information	Display Information icon.

InputBox

InputBox function display a prompt in the form of dialog box as shown below and waits for the user to input some value in textbox or click a button.

If user clicks on OK button then it will return content of textbox in the form of string. If user clicks on Cancel Button then it will return a blank value.



InputBox (Prompt As String,[Title As String=""], [DefaultResponse As String=""], [XPos As Integer = -1], [YPos As integer = -1]) As String

Here,

(1) Prompt is a compulsory argument. The String that you specify as a Prompt will display as a message in the InputBox Dialog.

(2) Title is an optional Argument. The String that you specify as a Title will display in the title bar of the InputBox. If you skip this argument then name of the application will display in the title bar.

(3) DefaultResponse is an Optional Argument. The String that you specify as a DefaultResponse will display as a default value in the textbox of the InputBox. If you skip this argument then Textbox of the InputBox is displayed empty.

(4) XPos is an Optional Argument. It Specify the distance (in pixel) of the left edge of the Input box from the left edge of the screen.

(5) YPos is an Optional Argument. It Specify the distance (in pixel) of the upper edge of the Input box from the top edge of the screen.

Note: If you skip XPos and YPos then InputBox will display in the centre of the screen.

What's difference between MsgBox() function and MessageBox.Show() in VB.net?

MsgBox() displays normal messagebox, but when using MessageBox.Show(), it displays a messagebox with Windows Classic Title Bar

Collections

- VB.NET Collections are data structures that holds data in different ways for flexible operations .
- The important datastructres in the Colletions are ArrayList ,HashTable, Sorted list,Stack and Queue etc.
- The main advantages of collections are:

1 .differentdatatype can be under one collection

2 .individual data can be deleted from the collection

3. Collection provides the features of searching,sorting and adding the data as and when needed

ArrayList

- is one of the most flixible data structure from.
- contains a simple list of values.
- easily we can add , insert , delete , view etc..
- It represents ordered collection of an object that can be **indexed** individually.
- very flexible because it grow dynamically and also shrink .
- Commonly used methods are:
  - Add(item) : add elements in arraylis
  - Insert(index, item) : add element to particular index
  - Remove(item) : remove item from array
  - RemoveAt(item_index) : remove item from particular index
  - Sort () : - sort elements in ascending order
  - Copy() : - copy one arraylist into an another arraylist
  - Binary search(item) : search data using binary search technique so needed only sorted data
  - Indexof(item) : search the first occurance of data in given array list. Returns position
  - LastIndexOf (item):search the last occurance of data in given array list. Returns position

```
DimarAsNewArrayList

ar.Add(10)
ar.Add(3)
ar.Add(15)
ar.Add(4)

ar.Sort()
MsgBox(ar.BinarySearch(15))     →3
MsgBox(ar.IndexOf(4))           →1
ar.Insert(2, 8)
MsgBox(ar.Contains(8))
→true
```

HashTable

- stores a (Key, Value) pair type collection of data .
- keys are unique and value can be of any datatype.

- We can retrive items from hashTable to provide the key .
- Both key and value are Objects.
- Commonly used method are:
  - Add(key, Value) : adds keys and value to hash table
  - Item(key) : gives value of the specified key from hash table
  - Contains(key): check wheather the given key is in hash table or not
  - Remove(key) : remove the specified key from hash table
  - Keys() : retrives all the key from the hash table
  - Clear() : remove all the element from the hash table
  - Count() : returns the number of element in the hash table
  - Values () : retrives all the value from hash table

```
DimhtAsNewHashtable
ht.Add("amit", 70)
ht.Add("priya", 80)
ht.Add("chintan", 90)
ht.Add("madhu", 90)

MsgBox(ht.Count)                    →4
MsgBox(ht.Item("amit"))             →70
ht.Remove("priya")
MsgBox(ht.Contains("madhu"))→true
ForEach t Inht.Keys
MsgBox(t &ht.Item(t) &ht(t))
Next
ForEach t Inht.Values
MsgBox(t)
Next
```

Sorted List

- It uses a **key** as well as an **index** to access the items in a list.
- A sorted list is a combination of an array and a hash table. It contains a list of items that can be accessed using a key or an index. If you access items using an index, it is an ArrayList, and if you access items using a key, it is a Hashtable. The collection of items is always sorted by the key value.
  - Add(key, Value) : adds keys and value to sorted list. Key must be unique
  - Containskey(key): check wheather the given key is in sorted list or not
  - Getkeylist() : retrives all the key from the sorted ist
  - Clear() : remove all the element from the sorted list
  - Count() : returns the number of element in the sorted list

```
Dim s AsNewSortedList
s.Add("red", 10)
```

```
s.Add("green", 5)
s.Add("blue", 5)
s.Add("black", 4)

ForEach x Ins.GetKeyList()
MsgBox(x)                    →
black,blue,green,red
Next

MsgBox(s.ContainsKey("blue"))
→true
s.Clear()

ForEach x Ins.GetKeyList()
MsgBox(x)
Next
```

UNIT-3:Introduction to Windows controls

A control is an object that can be drawn on tothe Form to enable or enhance user interaction with the application.

Examples of these controls, TextBoxes,Buttons, Labels, Radio Buttons, etc.

All these Windows Controls are based on theControl class, the base class for all controls.

VB.NET allows us to work with controls in twoways: at design time and at runtime.

working with controls at design time means,controls are visible to us and we can work with them by dragging and dropping them from the Toolbox and setting their propertiesin the properties window.

Working at runtime means, controls are notvisible while designing, are created and assigned properties in code and are visible only when the application is executed.

The Control class is in the System.Windows.Forms namespace.

It is a base class for the Windows Controls.

VB.NET Controls are the pillars that help in creating a GUI Based Applications in VB.Net quickly and easily. These are objects that you can drag to the Form using the Control toolbox in the IDE. Each VB.NET Control has some properties, events, and methods

Properties describe the object
Methods are used to make the object do something
Events describe what happens when the user/Object takes any action.

Common properties:

Name: This property uniquely Identifycontrol/object on form.

Enabled: Return /sets a value that determineswhether an object can respond to user- generated events like click, double click etc.Its value is either true or false.Default value: “true”If value is set to false, then object will not responduser-generated events.

Visible: Return/sets a value that determines whether object is visible or hidden.Its value is either true or false.Default value: True.Set to false to hide object.

Backcolor: Return/set background color used to display teat and graphics in object.

Forecolor: Return/sets foreground color used to display text and graphic in an object.

Font: Return font object.

Size: Return/sets the height and width of object.

Left: Return/sets the distance between internal left edge of object and left edge of itscontainer.

Top: Returns/sets distance between internal topedge of object and Top edge of its container.

Form Controls

Form:

A **Form** is used in VB.NET to create a form-based or window-based application. Using the form, we can build a attractive user interface. It is like a container for holding different control that allows the user to interact with

an application. The controls are an object in a form such as buttons, Textboxes, Textarea, labels, etc. to perform some action. However, we can add any control to the runtime by creating an instance of it.

A Form uses a **System.Windows.Form** namespace, and it has a wide family of controls that add both forms and functions in a Window-based user interface

Properties	Description
BackColor	It is used to set the background color for the form.
BackgroundImage	It is used to set the background image of the form.
Cursor	It is used to set the cursor image when it hovers over the form.
AllowDrop	Using the AllowDrop control in a form, it allows whether to drag and drop on the form.
Font	It is used to get or set the font used in a form.
Locked	It determines whether the form is locked or not.
FormBorderStyle	It is used to set or get border style in a form.
Text	It is used to set the title for a form window.
MinimizeBox	MinimizeBox It is used to display the minimum option on the title bar of the form.
IsMDIChild	It is used to authenticate whether a form is a container of a Multiple Document Interface (MDI) child form.
Autoscroll	It allows whether to enable auto-scrolling in a form.
MaximizeBox	It is used to display the maximum option on the title bar of the form.
MaximumSize	It is used to set the maximum height and width of the form.
Language	It is used to specifies the localized language in a form.
AcceptButton	It is used to set the form button if the enter key is pressed.
Top, Left	It is used to set the top-left corner coordinates of the form in pixel.
Name	It is used to define the name of the form.
MinimumSize	It is used to set the minimum height and width of the form.
Enabled	It uses the True or False value to enable mouse or keyboard events in the form.
TopMost	It uses a Boolean value that represents whether you want to put the window form on top of the other form. By default, it is False.

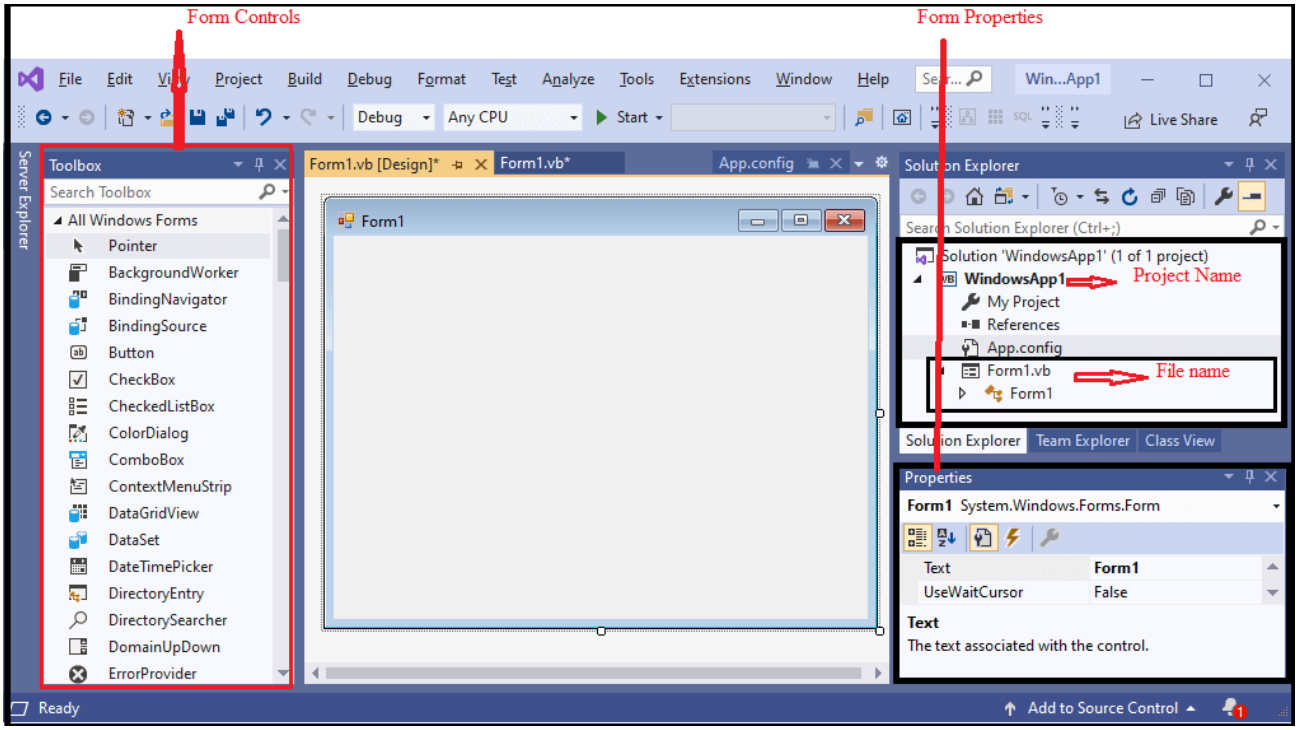
The following are the most important list of events related to a form.

Events	Description
Activated	An activated event is found when the user or program activates the form.
Click	A click event is active when the form is clicked.
Closed	A closed event is found before closing the form.
Closing	It exists when a form is closing.
DoubleClick	DoubleClick The DoubleClick event is activated when a user double clicks on the form.
DragDrop	A DragDrop event is activated when a drag and drop operation is performed.
MouseDown	A MouseDown event is activated when the mouse pointer is on the form, and the mouse button is pressed.
GotFocus	A GotFocus event is activated when the form control receives a focus.
HelpButtonClicked	It is activated when a user clicked on the help button.
KeyDown	A KeyDown event is activated when a key is pressed while focussing on the form.
KeyUp	A KeyUp event is activated when a key is released while focusing on the form.
Load	The load event is used to load a form before it is first displayed.
LostFocus	It is activated when the form loses focus.
MouseEnter	A MouseEnter event is activated when the mouse pointer enters the form.
MouseHover	A MouseHover event is activated when the mouse pointer put on the form.
MouseLeave	A MouseLeave event is activated when the mouse pointer leaves the form surface.
Shown	It is activated whenever the form is displayed for the first time.
Scroll	A Scroll event is activated when a form is scrolled through a user or code.
Resize	A Resize event is activated when a form is resized.
Move	A Move event is activated when a form is moved.

For creating a Windows Forms application in VB.NET, we need to follow the following steps in Microsoft Visual Studio.

1. GoTo File Menu.
2. Click on New Project.
3. Click on Windows Forms App or Application

And finally, click on the 'Create' button to create your project, and then, it displays the following window form with a name Form1.



Label Control

In VB.NET, a label control is used to display descriptive text for the form in control. It does not participate in user input or keyboard or mouse events. Also, we cannot rename labels at runtime. The labels are defined in the class System.Windows.Forms namespace.

Button Control

Button control is used to perform a click event in Windows Forms, and it can be clicked by a mouse or by pressing Enter keys. It is used to submit all queries of the form by clicking the submit button or transfer control to the next form. However, we can set the buttons on the form by using drag and drop operation.

BackColor	It is used to set the background color of the Button in the Windows form.
BackgroundImage	It is used to set the background image of the button control.
ForeColor	It is used to set or get the foreground color of the button control.
Image	It is used to set or gets the image on the button control that is displayed.

Text	It is used to set the name of the button control in the windows form.

Button Events:

Click: A Click event is found in the button control when the control is clicked.

TextBox Control:

TextBoxes are used to accept input from the useror used to display text.

By default we can enter up to 2048 characters ina TextBox but if the Multiline property is set to True we can enter up to32KB of text.

Properties:

Enabled: Default value is True. To disable, set theproperty to False.

Multiline: Setting this property to True makes theTextBox multiline which allows to accept multiplelines of text. Default value is False.

PasswordChar: Used to set the password character. The text displayed in the TextBox will be the character set by the user. Say, if you enter the text that is entered in the TextBox is displayed as.

ReadOnly: Makes this TextBox readonly. It doesn'tallow to enter any text.

TextAlign: Allows to align the text from three possible options. The default value is left and you can set the alignment of text to right or center.

Scrollbars: Allows to add a scrollbar to a Textbox. Very useful when the TextBox is multiline. You have four options with this property. Options areNone, Horizontal, Vertical and Both.

WordWrap: The WordWrap properties validate whether the multiline Textbox control automatically wraps words to the beginning of the next line when necessary.

RichTextBox Control:

The **RichTextBox** control allows the user to enter and edit text. It also provides more advanced formatting features than the standard TextBoxcontrol. It provides the following features.

Text can be assigned directly to the control, or can be loaded from a Rich

Text Format (RTF) or plain text file.

The text within the control can be assigned character and paragraphformatting.

The **RichTextBox** control provides a number of properties you can use to apply formatting to any portion of text within the control. To change the formatting of text, it must first be selected. Only selected text can be assigned character and paragraph formatting. Once a setting has been made to a selected section of text, all text entered after the selection is also formatted with the same settings until a setting change is made or adifferent section of the control's document is selected.

Properties:

AcceptsTab: Gets or sets a value indicating whether pressing the TAB key in a multiline text box control types a TAB character in the control instead of moving the focus to the next control in the tab order.

Dock: Gets or sets which control borders are docked to its parent control and determines how a control is resized with its parent.

SelectionFont: Gets or sets the font of the current text selection or insertion point.
SelectionColor: Gets or sets the text color of the current text selection or insertion point.
SelectedText: Gets or sets the selected text within the RichTextBox
SelectionBullet: Gets or sets a value indicating whether the bullet style is applied to the current selection or insertion point.

SelectionIndent: Gets or sets the length, in pixels, of the indentation of the line where the selection starts.

SelectionRightIndent: The distance (in pixels) between the right edge of the RichTextBox control and the right edge of the text that is selected or added at the current insertion point.

SelectionHangingIndent: Gets or sets the distance between the left edge of the first line of text in the selected paragraph and the left edge of subsequent lines in the same paragraph.

Methods

Clear(): Clears all text from the text box control.
Copy(): Copies the current selection in the text box to the **Clipboard**.
Cut(): Moves the current selection in the text box to the **Clipboard**.
Paste(): Replaces the current selection in the text box with the contents of the **Clipboard**.
Focus(): Sets input focus to the control.
Undo(): Undoes the last edit operation in the text box.
Redo(): Reapplies the last operation that was undone in the control.

Events

Click: Occurs when the text box is clicked.
DoubleClick: Occurs when the control is double-clicked.
TextChanged: Occurs when the Text property value changes.

BackColorChanged: Occurs when the value of the BackColor property changes.

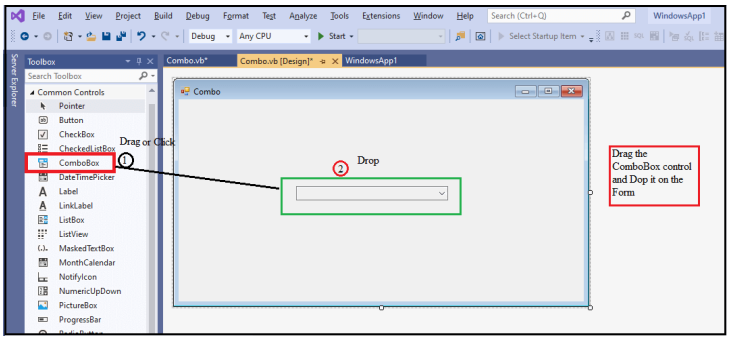
The **RichTextBox** is typically used to provide text manipulation and display features similar to word processing applications suchas Microsoft Word.

ComboBox Control

The **ComboBox** control is used to display more than one item in a drop-down list. It is a combination of **Listbox** and **Textbox** in which the user can input only one item. Furthermore, it also allows a user to select an item from a drop-down list.

Let's create a ComboBox control in the VB.NET Windows by using the following steps.

Step 1: We need to drag the combo box control from the toolbox and drop it to the Windows form, as shown below.



Step 2: Once the ComboBox is added to the form, we can set various properties of the ComboBox by clicking on the ComboBox control.

ComboBox Properties

There are following properties of the ComboBox control.

Items: Gets an object representing the collection of the items contained in this ComboBox.
SelectedIndex :Gets or sets the index specifying the currently selected item
SelectedItem :Gets or sets currently selected item in the ComboBox.
SelectedText: Gets or sets the text that is selected in the editable portion of a ComboBox.
Sorted: Gets or sets a value indicating whether the items in the combo box are sorted.
Text: Gets or sets the text associated with this control.
MaxDropDownItems: Gets or sets the maximum number of items to be displayed in the drop-down part of the combo box.

Events

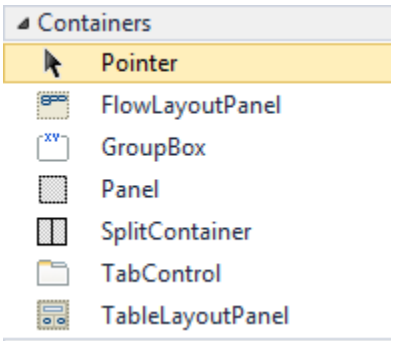
DropDown: Occurs when the drop-down portion of a combo box is displayed.
SelectedIndexChanged: Occurs when the SelectedIndex property of a ComboBox control has changed.

Methods

FindString
Finds the first item in the combo box that starts with the string specified as an argument.

Container Controls

Container controls are a misunderstood concept for many a new programmer. It is as if newbies do not realize the true power of a container control.



GroupBox and Panel:

A GroupBox control is a container control that is used to place Windows Forms child controls in a group. The purpose of a GroupBox is to define user interfaces where we can categories related controls in a group.

The **GroupBox** is the perfect control to use when you have to separate different controls based on their purpose for your app.

To create a GroupBox control at design-time, you simply drag and drop a GroupBox control from Toolbox to a Form in Visual Studio. After you drag and drop a GroupBox on a Form, the GroupBox looks like Figure 1. Once a GroupBox is on the Form, you can move it around and resize it using mouse and set its properties and events.

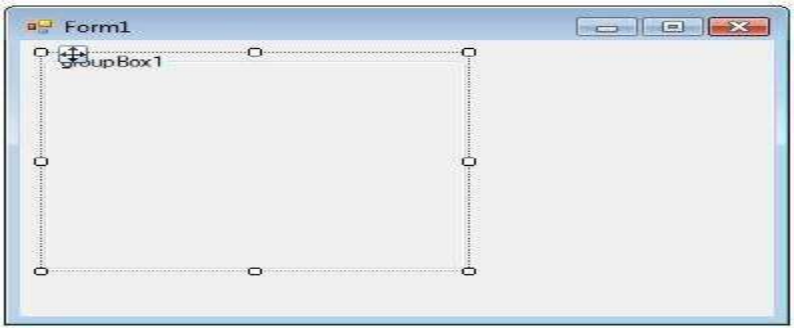


Figure 1

Creating a GroupBox control at run-time is merely a work of creating an instance of GroupBox class, set its properties and add GroupBox class to the Form controls.

First step to create a dynamic GroupBox is to create an instance of GroupBox class. The following code snippet creates a GroupBox control object.

```
Dim authorGroup As New GroupBox
```

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

```
authorGroup.Name = "GroupBox1"
authorGroup.BackColor = Color.LightBlue
authorGroup.ForeColor = Color.Maroon
authorGroup.Text = "Author Details"
authorGroup.Font = New Font("Georgia", 12)
```

Once a GroupBox control is ready with its properties, next step is to add the GroupBox control to the Form. To do so, we use Form.Controls.Add method. The following code snippet adds a GroupBox control to the current Form.

```
Me.Controls.Add(authorGroup)
```

Panels

The Panel control is a container control that is used to host a group of similar child controls. One of the major uses I found for a Panel control when you have to show and hide a group of controls.

Instead of show and hide individual controls, you can simply hide and show a single Panel and all child controls.

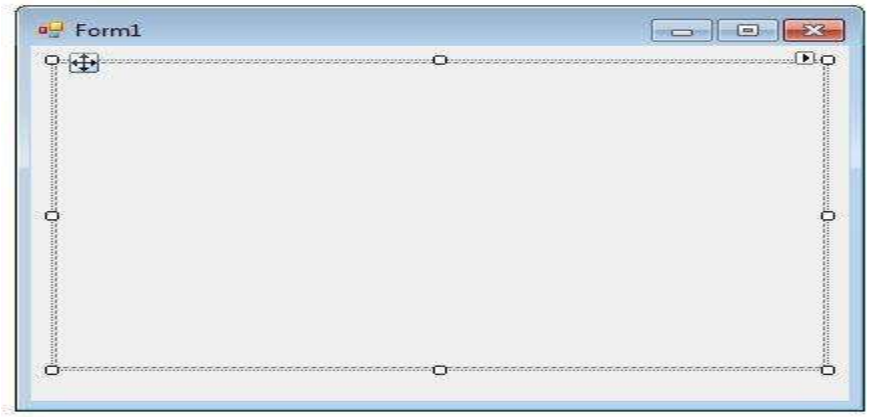
are those controls which contain othercontrols, for example, a set of radio buttons, checkboxes, etc.

Creating a Panel

We can create a Panel control using the Forms designer at design-time or using the Panel class in code at run-time.

Design-time

To create a Panel control at design-time, you simply drag and drop a Panel control from Toolbox to a Form in Visual Studio. After you drag and drop a Panel on a Form, the Panel looks like Figure 1. Once a Panel is on the Form, you can move it around and resize it using mouse and set its properties and events.



Run-time

Creating a Panel control at run-time is merely a work of creating an instance of Panel class, set its properties and adds Panel class to the Form controls.

First step to create a dynamic Panel is to create an instance of Panel class. The following code snippet creates a Panel control object.

```
Dim dynamicPanel As New Panel()
```

In the next step, you may set properties of a Panel control. The following code snippet sets location, size and Name properties of a Panel.

```
dynamicPanel.Location = New System.Drawing.Point(26, 12)
dynamicPanel.Name = "Panel1"
dynamicPanel.Size = New System.Drawing.Size(228, 200)
dynamicPanel.BackColor = Color.LightBlue
```

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

Controls.Add(dynamicPanel)

Panels are similar to Groupboxes but the difference, Panels cannot display captions where as GroupBoxes can and Panels can havescrollbars where as GroupBoxes can't.

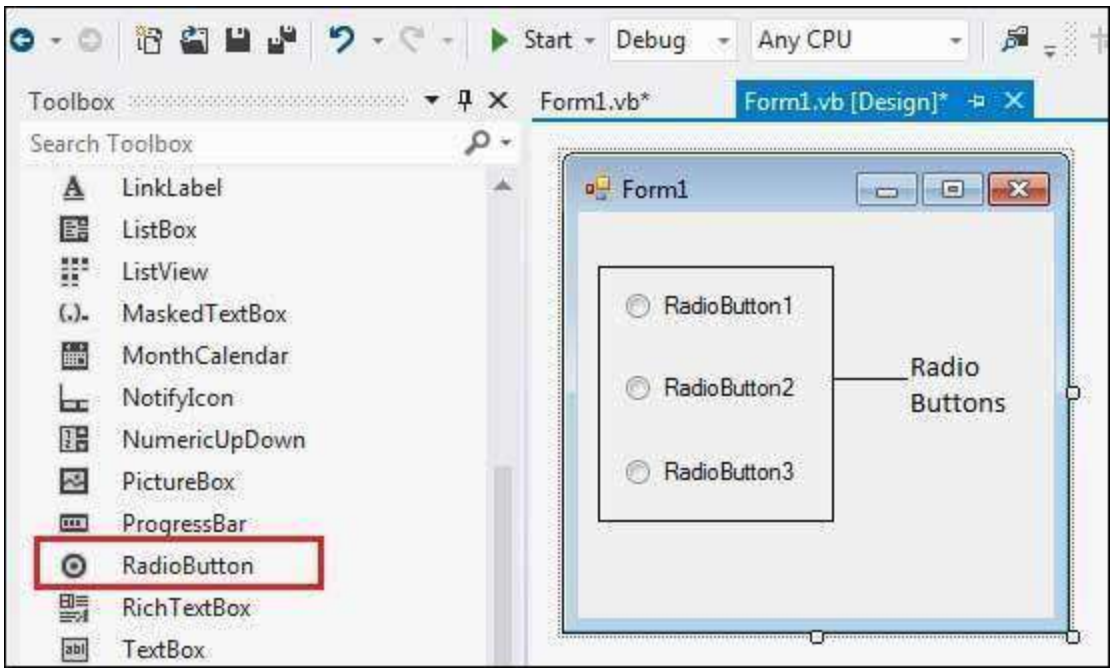
If the Panel's Enabled property is set to False then the controls which the Panel contains are also disabled. Panels are based on the ScrollableControl class.

Notable property of the Panel control in theappearance section is the BorderStyle property.

Notable property in the layout section isthe AutoScroll property.

RadioButton Control:

The RadioButton control is used to provide a set of mutually exclusive options.
The user can select one radio button in a group. If you need to place more than one group of radio buttons in the same form, you should place them in different container controls like a GroupBox control.
Create three radio buttons by dragging RadioButton controls from the Toolbox and dropping on the form.



The ***Checked*** property of the radio button is used to set the state of a radio button. You can display text, image or both on radio button control. You can also change the appearance of the radio button control by using **the *Appearance* property.**

Properties of the RadioButton Control
Text: Gets or sets the caption for a radio button.

Events of the RadioButton Control
CheckedChanged: Occurs when the value of the Checked property of the RadioButton control is changed.

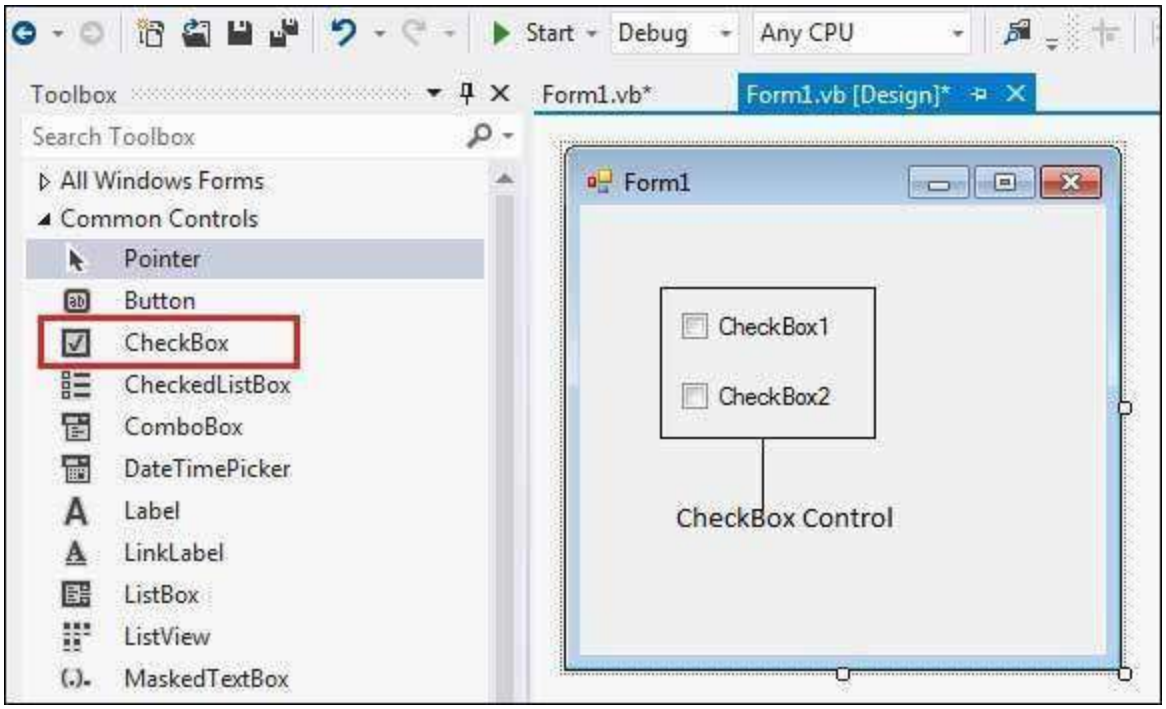
CheckBox Control :

The CheckBox control allows the user to set true/false or yes/no type options. The user can select or deselect it. When a check box is selected it has the value True, and when it is cleared, it holds the value False.

CheckBoxes are those controls which gives us anoption to select, say, Yes/No or True/False.

When a checkbox is selected a check (a tick mark)appears indicating a selection.

The CheckBox control is based on the TextBoxBaseclass which is based on the Control class.



The CheckBox control has three states, **checked**, **unchecked** and **indeterminate**. In the indeterminate state, the check box is grayed out. To enable the indeterminate state, the *ThreeState* property of the check box is set to be **True**.

Properties of the CheckBox Control
CheckAlign:Gets or sets the horizontal and vertical alignment of the check mark on the check box.
Checked: Gets or sets a value indicating whether the check box is selected.
CheckState: Gets or sets the state of a check box.

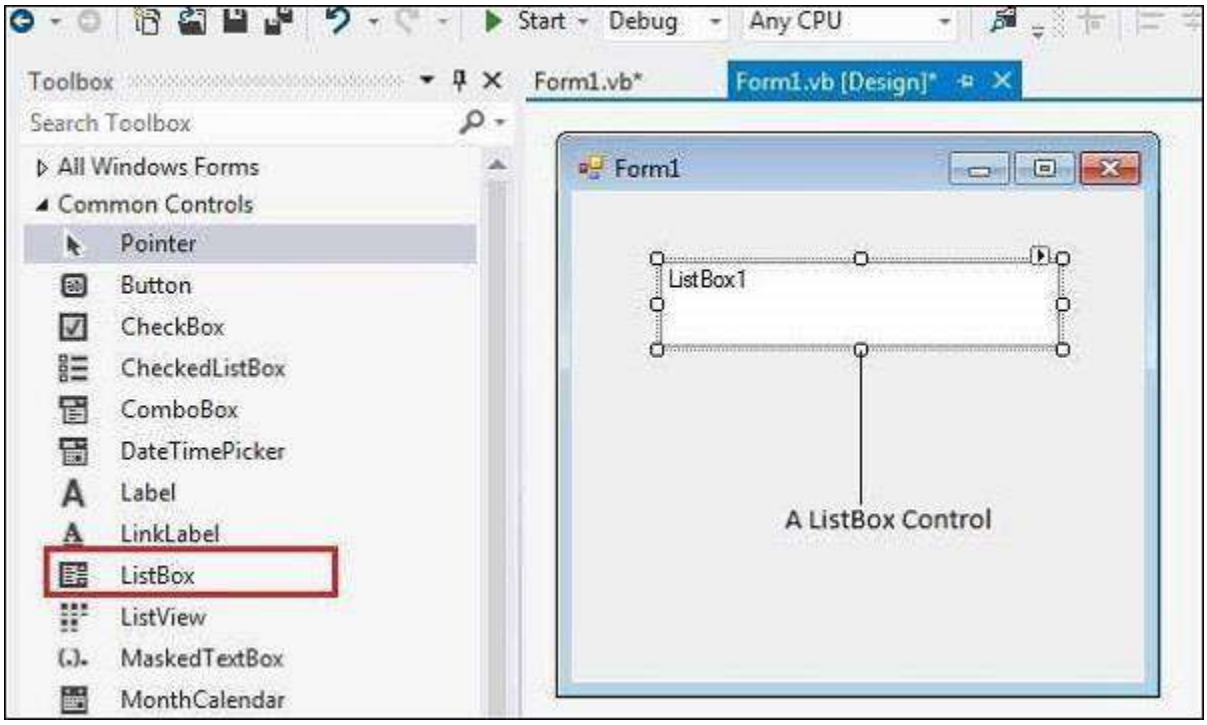
Methods of the CheckBox Control
OnClick: Raises the OnClick event.

Events of the CheckBox Control
CheckedChanged:Occurs when the value of the Checked property of the CheckBox control is changed.

ListBox Control:

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

Let's create a list box by dragging a ListBox control from the Toolbox and dropping it on the form.



Properties of the ListBox Control
Items: Gets the items of the list box.
SelectedItem: Gets or sets the currently selected item in the list box.
SelectedIndex: Gets or sets the zero-based index of the currently selected item in a list box.
SelectionMode:Gets or sets the method in which items are selected in the list box. This property has values −

- None
- One
- MultiSimple
- MultiExtended

MultiColumn property displays items in multiple columns instead of a straight vertical list of items. This allows the control to display more visible items and prevents the need for the user to scroll to an I tem.

Sorted property is used to sort the data's displayed in the listbox.
SelectedIndices
Gets a collection that contains the zero-based indexes of all currently selected items in the list box.

Methods of the ListBox Control

GetSelected
Returns a value indicating whether the specified item is selected.
SetSelected
Selects or clears the selection for the specified item in a ListBox.
FindString
Finds the first item in the ListBox that starts with the string specified as an argument.

ClearSelected: Unselects all items in the ListBox.

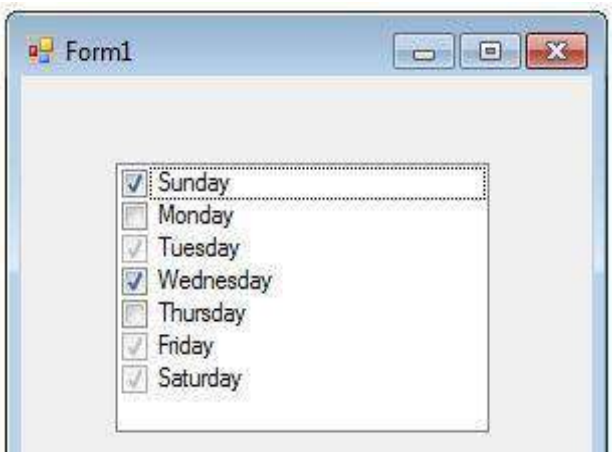
Events of the ListBox Control

Click: Occurs when a list box is selected.

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

Checked ListBox Control :

The CheckedListBox control in VB.Net provides all the features and functionality of a standard ListBox control, with the added capability of displaying a check mark alongside each item in the list. This additional functionality allows users to select multiple items from the list by checking the corresponding checkboxes.



Properties	Description
HorizontalScrollbar	Property used to Get or set if a checked list box should display a horizontal scroll bar.
Item Height	Property returns the height of an item.
Items	Property returns the items of this control.
MultiColumn	Property to get or set the if the checked list box allows multicolumn.
SelectedIndex	Property used to set or get the selected item in the control.
SelectedItem	Property used to get or set the selected item.
SelectedItems	Property used to get or set the selected items.

Locked	Prevents the control being moved at design time.

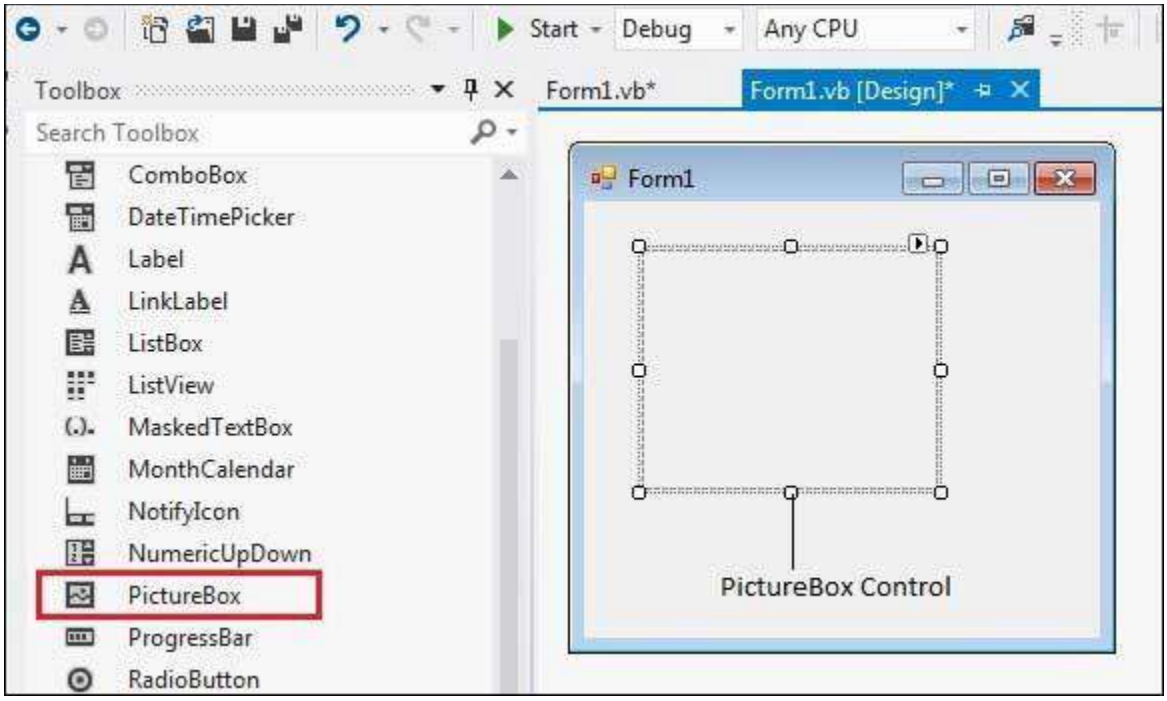
Methods:

ClearSelected(): It is used to unselect all the selected items in the CheckedListBox.

Sort():A Sort() method is used to sort or organize all the items available in CheckedListBox.

PictureBox control :

It is used to display the images on Windows Form. The PictureBox control has an image property that allows the user to set the image at runtime or design time.



Once the PictureBox is added to the form, we can set various properties of the image by clicking on the PictureBox control.

Properties of the PictureBox Control
Image: Gets or sets the image that is displayed in the control.
SizeMode: Determines the size of the image to be displayed in the control. This property takes its value from the PictureBoxSizeMode enumeration, which has values −

- **Normal** − the upper left corner of the image is placed at upper left part of the picture box
- **StrechImage** − allows stretching of the image
- **AutoSize** − allows resizing the picture box to the size of the image
- **CenterImage** − allows centering the image in the picture box
- **Zoom** − allows increasing or decreasing the image size to maintain the size ratio.

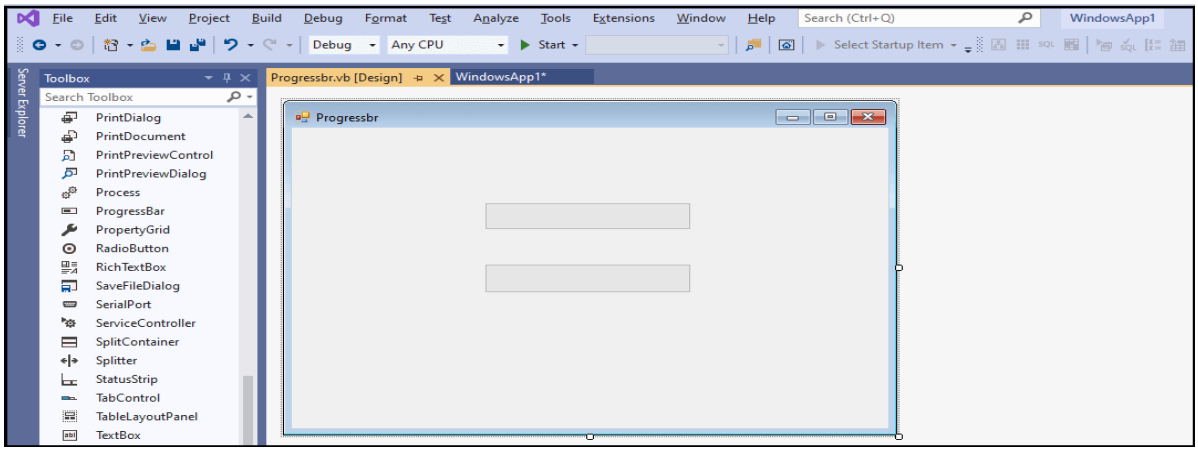
BorderStyle: It is used to set the border style for the picture box in the windows form

Events of the PictureBox Control
Resize: The resize event occurs when the picture box control is changed.

Methods of the PictureBox Control
Load
Displays an image in the picture box

ProgressBar Control :

The Window ProgressBar control is used by the user to acknowledge the progress status of some defined tasks, such as downloading a large file from the web, copying files, installing software, calculating complex results, and more.



Once the ProgressBar is added to the Form, we can set various properties of the ProgressBar by clicking on the ProgressBar control.

Properties of the ProgressBar Control

Minimum specifies the minimum value with which the progressbar willstart.

Maximum specifies the maximum value with which the progressbar willterminate.

Value the current value of the progressbar

Step the value by which every time the bar in progressbar shouldincrement.

Methods of the ProgressBar Control

ToString: The ToString method is used to display the progress bar control by returning the string.

Events of the ProgressBar Control

BackgroundImageChanged: When the background property changes to the progress bar control, the BackgroundImageChanged event changes.

TextChanged: It occurs when the property of the text is changed in the progress bar control.

Treeview:

The TreeView control is used to display a hierarchy of nodes (both parent,child).

You can expand and collpase these nodes by clicking them. In addition, the root node can be contracted or expanded by clicking on the plus sign (+) button. It is also useful to provide the full path of the root node to the child node.

This control is similar to Windows Explorer which displays a tree view in it's left pane to list all the folders on the hard disk.

Notable Properties of TreeView

Bounds: Gets the actual bound of the tree node

Checked: Gets/Sets whether the tree node is checked

FirstNode: Gets the first child tree node

FullPath: Gets the path from the root node to the current node

ImageIndex: Gets/Sets the image list index of the image displayed for a

nodeIndex: Gets the location of the node in the node collection

IsEditing: Gets whether the node can be edited

IsExpaned: Gets whether the node is expaned

IsSelected: Gets whether the node is selected

LastNode: Gets the last child node

NextNode: Gets the next sibling node

NextVisibleNode: Gets the next visible node

NodeFont: Gets/Sets the font for nodes

Nodes: Gets the collection of nodes in the currentnode

Parent: Gets the parent node of the current node

PrevNode: Gets the previous sibling node

PrevVisibleNode: Gets the previous visible node

Methods of the TreeView Control

Sort():A Sort method is used to sort the tree nodes that are available in the tree view control.

ToString: ToString method is used to return the name of the string that is in the tree view control.

Tooltip Control :

A tooltip is a small pop-up window that displays some information when you rollover on a control.

This class allows providing help to users when they place the mouse cursorover a control. The ToolTip class is typically used to alert users to the intended use of a control.

This class allows providing help to users when they place the mouse cursorover a control. The ToolTip class is typically used to alert users to the intended use of a control. For example, to specify ToolTip text for a TextBox control that accepts a name, specifying the format of the name totyped into the control. In order for ToolTip text to be displayed when the user moves the mouse cursor over a control, the ToolTip text to be displayed must be associated with the control within an instance of the ToolTip class. To associate ToolTip text with a control, use the SetToolTip method.

AutomaticDelay property enables to set a single delay value which is thenused to set the values of the AutoPopDelay, InitialDelay, and ReshowDelayproperties. Each time the AutomaticDelay property is set, the following values are set by default.

Property	Default Value
AutoPopDelay	10 times the AutomaticDelay property value.
InitialDelay	Equal to the AutomaticDelay property value.
ReshowDelay	1/5 of the AutomaticDelay property value.

AutopopDelay -The period of time (in milliseconds) that the ToolTip remains visible when the mouse pointer is stationarywithin a control. The default value is 5000.

InitialDelay - The period of time (in milliseconds) that the mouse pointer must remain stationary within a control beforethe ToolTip window is displayed.

ReshowDelay - The length of time that must transpire beforesubsequent ToolTip windows appear as the mouse pointer moves from one control to another.

MaskedTextbox :

MaskedTextBox control is a specialized form of the Textbox control. You can specify the format (the Mask property) of thedata required of the user. For example, you can select a mask for a ZIP code, a date, a phone number, or a social security number.

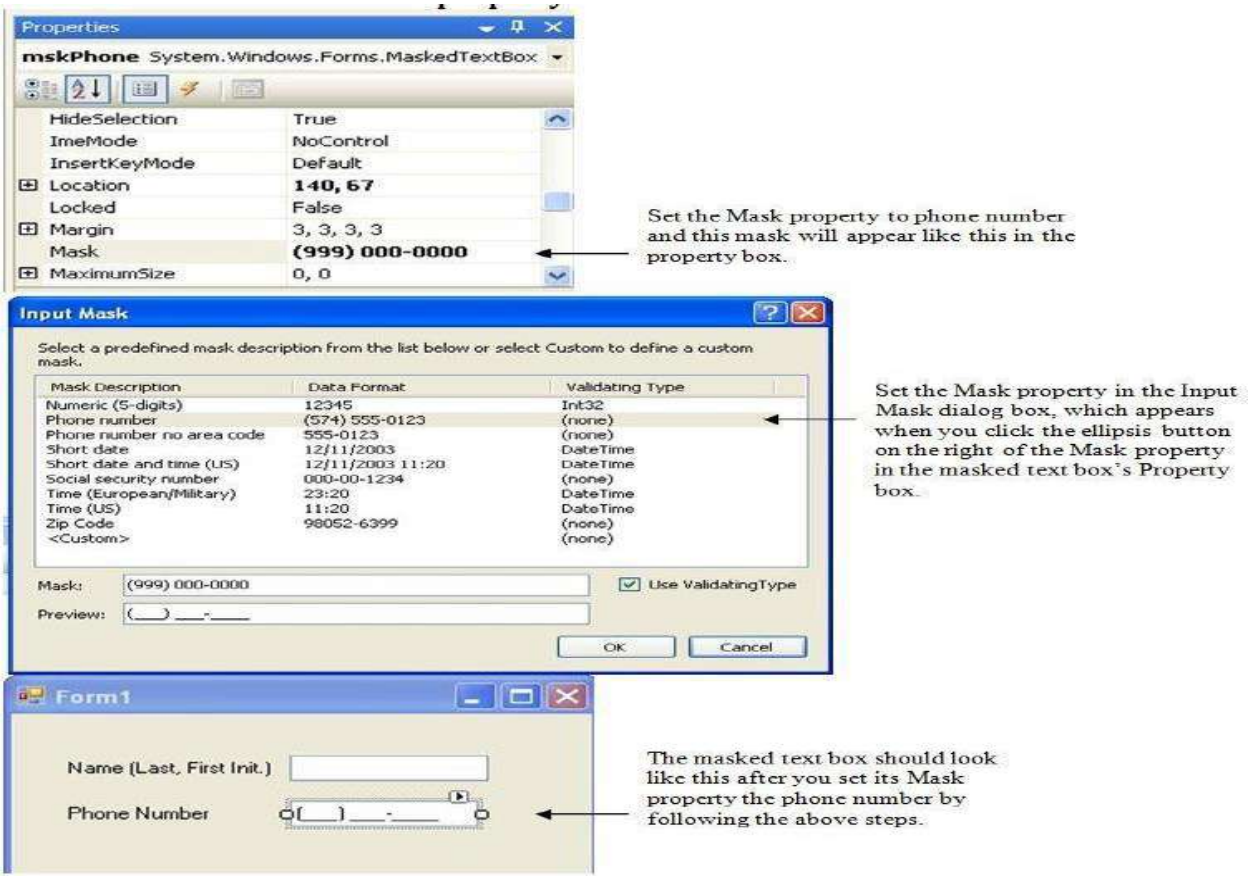
Masked TextBox Control in .NET is used to restrict the inputfrom user .

A MaskedTextBox control provides validation mechanism foruser input on a Form. For example, if you want a TextBox to accept date in mm/dd/yyyy format, you can set masking in the MaskedTextBox.

MaskedTextBox accepts text input of a specific format. We often require phone numbers to have their area code and also the correct number of digits. To solve this problem, we use the MaskedTextBox control in Windows Forms.

Properties	Description
TextAlign	Text alignment is set using this property

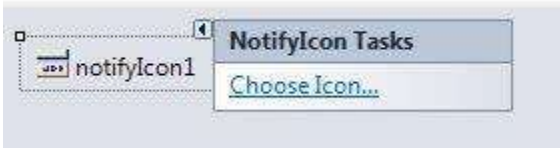
Mask	This property is used to set the predefined maskdescription.
TextMaskFormat	This property is used to gets or sets a value thatdetermines the literals, prompt character to beincluded in the formatted string.
HidePromptOnLeave	Property used to hide the mask characters, literalswhen the focus is lost from the control.
AllowPromptAsInput	Property used to set the input character as per thespecified by moving character.
TextMaskFormat	Property used to specify the formatting for the text.
RejectInputOnFirstFailure	Property used to gets or sets a value indicating whether to stop user input after the first invalidcharacter is entered.



NotifyIcon :

Icons in the status area are short cuts to processes that are running in the background of a computer, such as a virus protection program or a volume control. These processes do not come with their own user interfaces. The NotifyIcon classprovides a way to program in this functionality.

Icon property defines the icon that appears in the status area.



ContextMenu Pop-up menus for an icon are addressed withthe **ContextMenu** property.

Text property assigns ToolTip text. In order for the icon toshow tool tip bring the mouse to the icon.

LinkLabel :

LinkLabel is similar to a Label but they display ahyperlink.

Even multiple hyperlinks can be specified in thetext of the control and each hyperlink can perform a different task within the application.

They are based on the Label class which is basedon the Control class.

Notable properties of the LinkLabel control arethe ActiveLinkColor, LinkColor and LinkVisited which are used to set the link color.

Component

Image list, error provider, Help provider, Timer

ImageList Control :

The ImageList is a simple control that stores images used by other controls at runtime.

For example, a TreeView control can use icons to identify its nodes.

The simplest and quickest method of preparing these images is to create an ImageList control and add to it all the icons you need for decorating the TreeView control's nodes. The ImageList control maintains a series of bitmaps in memory that the TreeView control can access quickly at runtime.

ErrorProvider control:

The errorprovider control provides a user interface to indicate to the user that a control on a form has an error associated with it, or The ErrorProvider component provides an easy way to set validation errors. It allows us to set an error message for any control on the form when the input is not valid. When an error message is set, an icon indicating the error will appear next to the control and the error message is displayed as Tool Tip when the mouse is over the control.

Errorprovider properties:

Icon - Icon property are used to indicate an error.

ContainerControl - ContainerControl property are used to parent control, usually the form, that contains the data-bound controls on which the errorprovider can display error icon.

BlinkStyle - Controls whether the error icon blinks when an error is set.

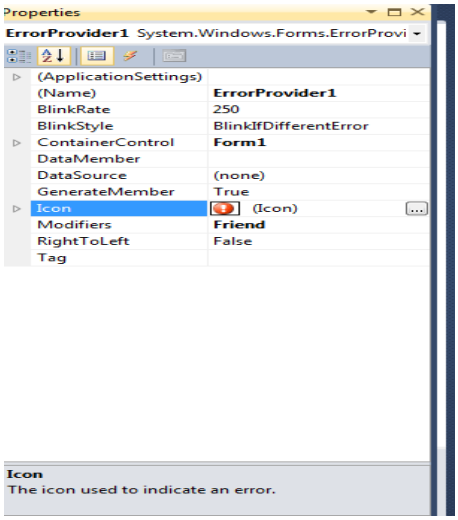


Figure 1.

Working with errorprovider:

Drag a errorprovider, TextBox and a Button from the Toolbox. Form looks like this.

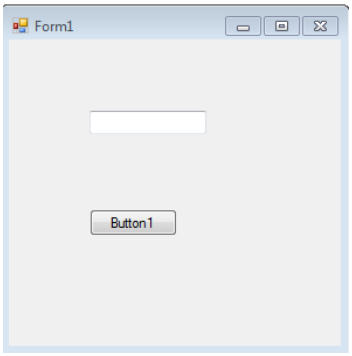
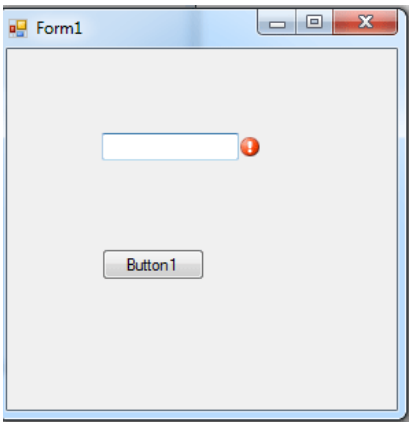


Figure 2.

Now double click on the form and write this code.

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.L
oad
        ErrorProvider1.SetError(TextBox1, " Please enter the text ")
    End Sub
```

Now run the application. Textbox displays an error icon.



HelpProvider control :

This control provides pop-up or on line help for controls. HelpProvider control Provides help with your application is very useful as it allows your users to understand it more easily, thereby increasing productivity and saving some money. Support for help in Visual Basic exists and you can display HTML files that can contain a set of linked topics.

Help class:

The Help class allows us to display HTML help to users. The Help class provides two methods: ShowHelp and ShowHelpIndex.

ShowHelp - ShowHelp is used to display a help file for a particular control and requires that control to be displayed along with the help file. The URL you specify for the help file can be in the form of F:\myHelp (local machine) or http:\\www.vbheaven.com\help.htm (Web).

ShowHelpIndex -ShowHelpIndex method is used to display the index of a specified help file. You call the ShowHelpIndex method just like you call the ShowHelp method.

Example:

Drag a button and level on the form.

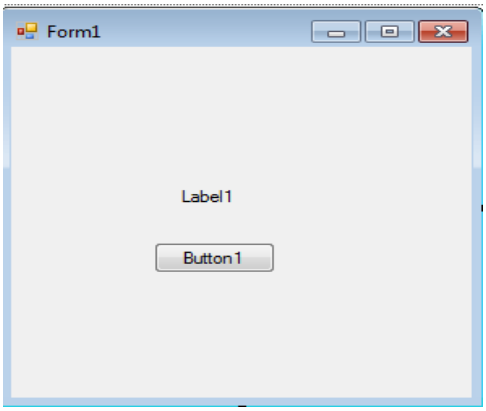


Figure 1.

Now double click on the button and add following code displays a help file and have a named myhelp.htm in the F: drive of your machine.

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.
Click
        Help.ShowHelp(Label1, "F:/myhelp.htm")
    End Sub
```

Now click on button myhelp.htm file will be display or select label control press F1 for help.

HelpProvider Component:

The HelpProvider component provides these additional properties to each control on the form. These properties are:

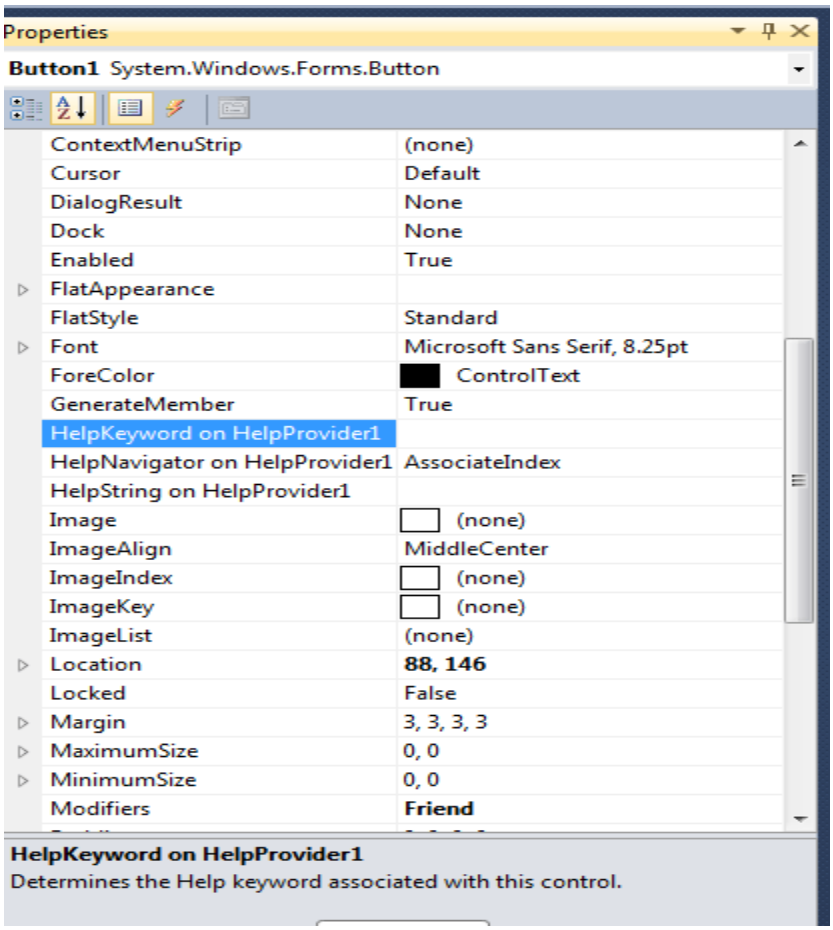


Figure 2.

HelpString - Determines the help string associated with a control.

HelpKeyWord - Determines the help keyword associated with a control.

HelpNavigator - Determines the kind of help associated with a control. Provides six values: TableOfContents, Find, Index, Topic, AssociatedIndex and KeywordIndex.

ShowHelp - Determines if help should be displayed for this control.

Example:

Using the property display a help on the button when we press F1.

Firstly set two property of the The HelpProvider component provides these additional properties to button.

ShowHelp on Helpprovider1 -> True

HelpString on HelpProvider1 -> write some string which will be display for help string such as please press the button.

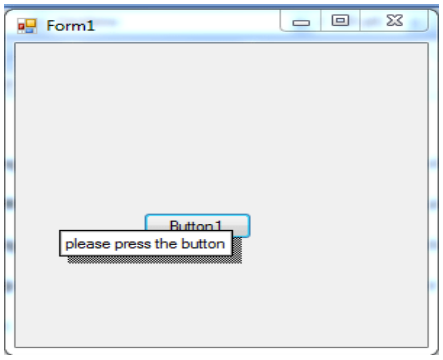
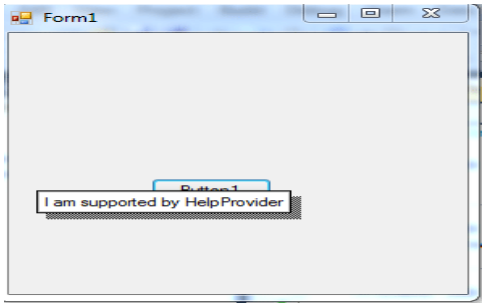


Figure 3.

The following code displays a help string when Button2 has the focus and F1 key is pressed.

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.L
oad
    HelpProvider1 .SetHelpString (Button1 , "I am supported by HelpProvider");
End Sub
```

The image below displays output from code above.



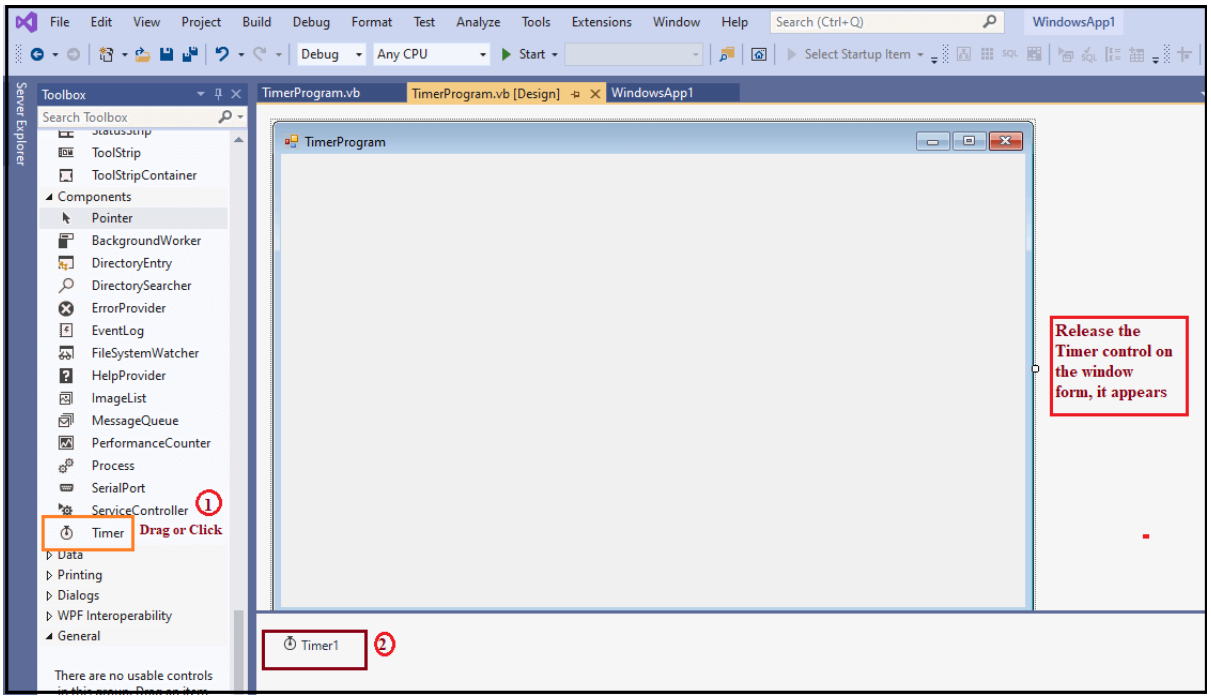
Timer Control :

The timer control is a looping control used to repeat any task in a given time interval. It is an important control used in Client-side and Server-side programming, also in Windows Services.

Furthermore, if we want to execute an application after a specific amount of time, we can use the **Timer Control**. Once the timer is enabled, it generates a tick event handler to perform any defined task in its time interval property. It starts when the start() method of timer control is called, and it repeats the defined task continuously until the timer stops.

Let's create a Timer control in the VB.NET Windows form by using the following steps.

Step 1: Drag and drop the Timer control onto the window form, as shown below.



Step 2: Once the Timer is added to the form, we can set various properties of the Timer by clicking on the Timer control.

Timer Control Properties

There are following properties of the VB.NET Timer control.

Properties	Description
Name	The Name property is used to set the name of the control.
Enabled	The Enables property is used to enable or disable the timer control. By default, it is True.
Interval	An Interval property is used to set or obtain the iteration interval in milliseconds to raise the timer control's elapsed event. According to the interval, a timer repeats the task.
AutoReset	The AutoReset property is used to obtain or set a Boolean value that determines whether the timer raises the elapsed event only once.
Events	Events property are used to get the list of event handler that is associated with Event Component.
CanRaiseEvents	It is used to get a value that represents whether the component can raise

	an event.

Events of Timer Control

Events	Description
Disposed	When control or component is terminated by calling the Dispose method, a Dispose event occurs.
Elapsed	When the interval elapses in timer control, the Elapsed event has occurred.
Tick	A tick event is used to repeat the task according to the time set in the Interval property. It is the default event of a timer control that repeats the task between the Start() and Stop() methods.

Methods of Timer Control

Methods	Description
BeginInt()	The BeginInt() method is used to start run time initialization of a timer control used on a form or by another component.
Dispose()	The Dispose() method is used to free all resources used by the Timer Control or component.
Dispose(Boolean)	It is used to release all resources used by the current Timer control.
Close()	The Close() method is used to release the resource used by the Timer Control.
Start()	The Start() method is used to begin the Timer control's elapsed event by setting the Enabled property to true.
EndInt()	The EndInt() method is used to end the run time initialization of timer control that is used on a form or by another component.
Stop()	The Stop() method is used to stop the timer control's elapsed event by setting Enabled property to false.

Data:dataset and datagrid

The DataSet class is a member of the System.Data namespace. It represents the first of the two major components of the ADO.NET architecture you learned about on Day 1, "ADO.NET In Perspective," the other being the .NET Data Providers. Its major attributes include the following:

- It is XML-based.
- It is an in-memory cache of data that is not backed by a file or data store—it is disconnected.
- It is independent of a data store and cannot communicate with one by itself.
- It can store data in multiple tables from multiple data stores that can be related through foreign key relationships.
- It stores multiple versions of the data for each column and for each row in each table.
- It can be serialized with full fidelity to XML for transport between tiers of a distributed application even when those tiers reside on separate physical machines.

Table 3.1 Important DataSet Members

Member	*Description*
Properties	
CaseSensitive	Property that gets or sets whether string comparisons are case sensitive
DataSetName	Property that gets or sets the name of theDataSet
Collections	
Relations	A collection of relations house in aDataRelationCollection object that link tables through foreign keys
Tables	A collection of tables (DataTable objects exposed through a DataTableCollection object) that store the actual data
Methods	
AcceptChanges	Method that commits all changes to the DataSet
Clear	Method that removes all rows from all tables
Clone	Method that copies the structure but no data from the DataSet

Copy	Method that copies both the structure and data of a DataSet
GetChanges	Method that returns a copy of the DataSet with only changed rows or those that match a givenDataRowState filter
GetXml	Method that returns an XML representation of the data
GetXmlSchema	Method that returns an XML representation of the structure of the DataSet
HasChanges	Method that returns a value indicating that there are pending changes
Merge	Method that merges this DataSet with one provided
ReadXml	Method that loads an XML schema and data into a DataSet
ReadXmlSchema	Method that loads an XML schema into aDataSet
RejectChanges	Method that rolls back all changes made to aDataSet (opposite of AcceptChanges)
Reset	Method that reverts the DataSet to its original state
WriteXml	Method that writes XML data and optionally the schema to a file or stream
WriteXmlSchema	Method that writes the XML schema to a file or stream

DataGridView

DataGridView provides a visual interface to data. It is an excellent way to display and allow editing for your data. It is accessed with VB.NET code. Data edited in the DataGridView can then be persisted in the database.

The DataGridView control looks like the Excel spreadsheet, which can be added onto the form so that the end-user can use it for specific purposes.

The DataGridView control is basically made of rows and columns. Additionally, every column has its header, where the header text can be changed. The header text can be changed in the designer. It can also be changed programmatically. Vertical and horizontal scrollbars appear automatically whenever they are needed. The scrollbars are used to scroll through the pages to see more content. So a DataGridView control may contain information of multiple pages.

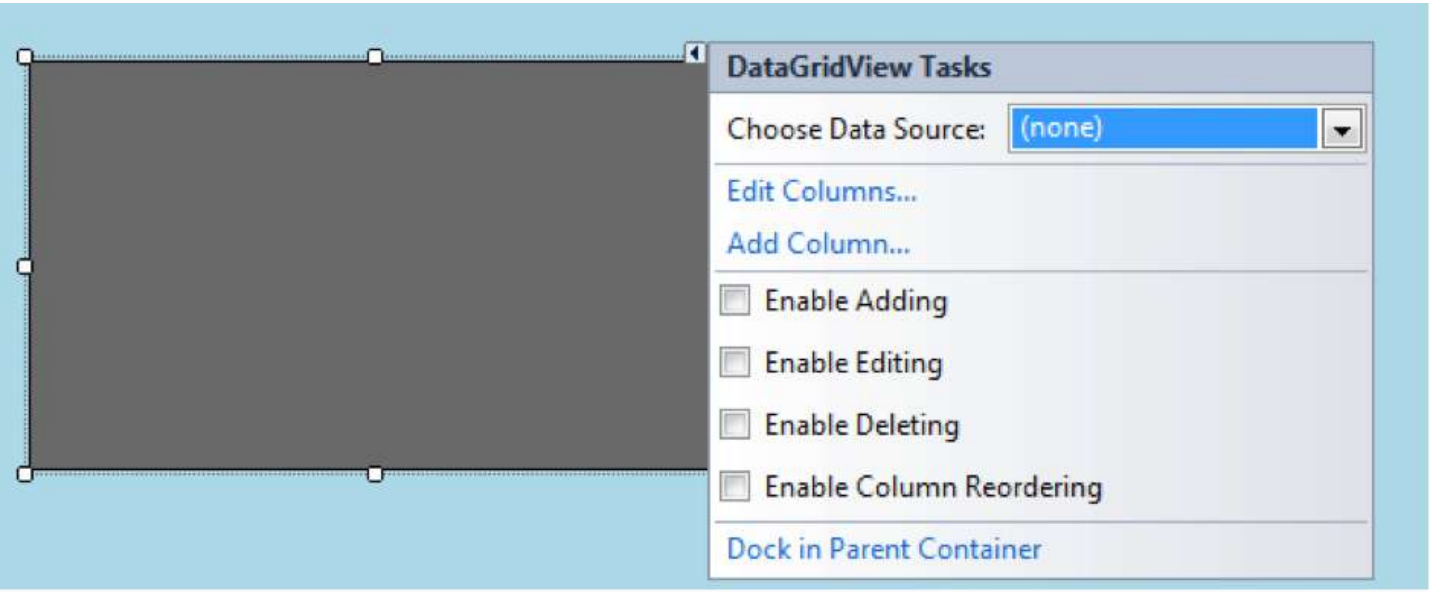
Show Data From Database into DataGridView

Although you can add data manually, the DataGridView control is generally used for showing data from the database. For example, if you have an Access database connected to the Visual Basic project, you can show data from it. The data from the database will be shown in rows and columns of the DataGridView control. Besides Access databases, you can also show data on the DataGridView control from other database management systems like Oracle, MS SQL Server, MySQL, dBASE, etc.

Choosing Data Source for the DataGridView control

After adding the DataGridView control on the form, the first thing that you need to do is to choose the data source, either from the designer or in the code, so that the data is automatically shown from the database.

Choosing data source from the designer: The following screenshot demonstrates how to choose data source from the designer.

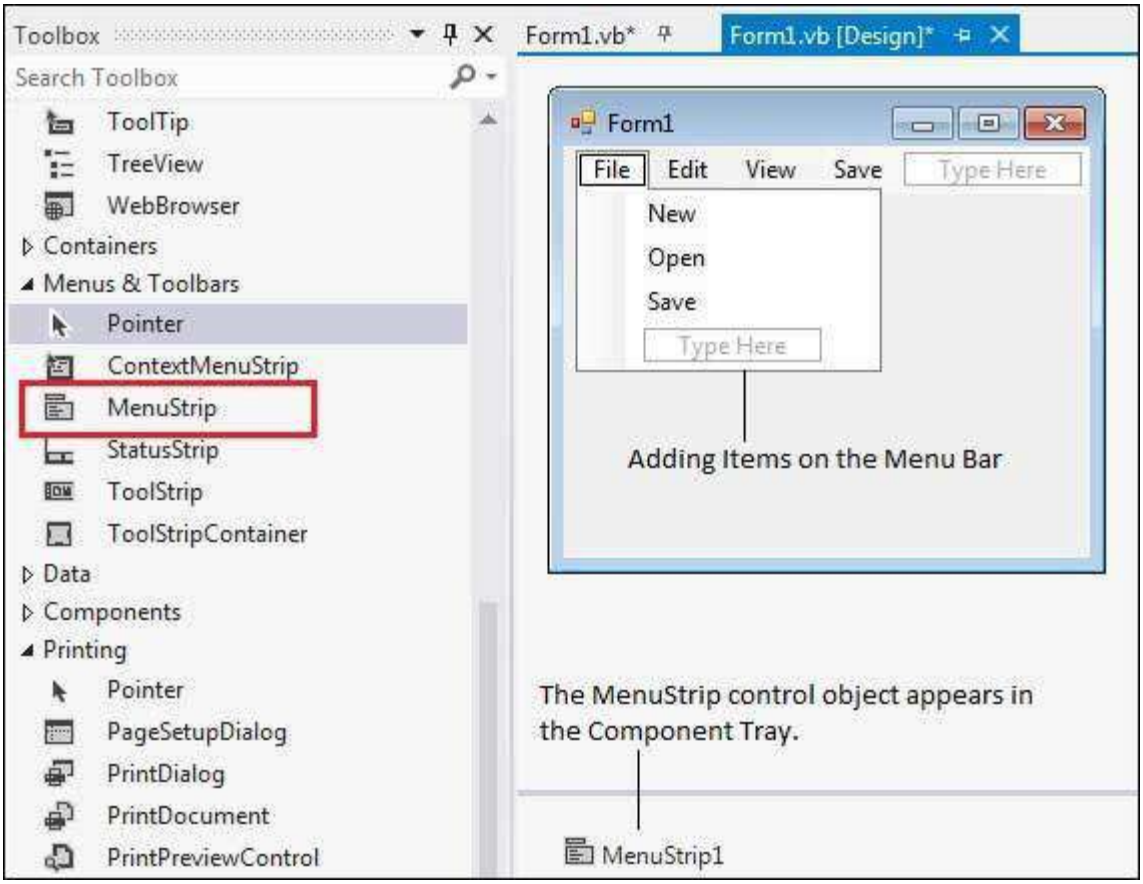


Menus

The **MenuStrip** control represents the container for the menu structure.

The MenuStrip control works as the top-level container for the menu structure. The ToolStripMenuItem class and the ToolStripDropDownMenu class provide the functionalities to create menu items, sub menus and drop-down menus.

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



Sr.No.	Property & Description
1	**CanOverflow** Gets or sets a value indicating whether the MenuStrip supports overflow functionality.
2	**GripStyle** Gets or sets the visibility of the grip used to reposition the control.
3	**MdiWindowListItem** Gets or sets the ToolStripMenuItem that is used to display a list of Multiple-document interface (MDI) child forms.
4	**ShowItemToolTips** Gets or sets a value indicating whether ToolTips are shown for the MenuStrip.

5	**Stretch** Gets or sets a value indicating whether the MenuStrip stretches from end to end in its container.

Sr.No.	Event & Description
1	**MenuActivate** Occurs when the user accesses the menu with the keyboard or mouse.
2	**MenuDeactivate** Occurs when the MenuStrip is deactivated.

Dialogue :

There are many built-in dialog boxes to be used in Windows forms for various tasks like opening and saving files, printing a page, providing choices for colors, fonts, page setup, etc., to the user of an application. These built-in dialog boxes reduce the developer's time and workload.

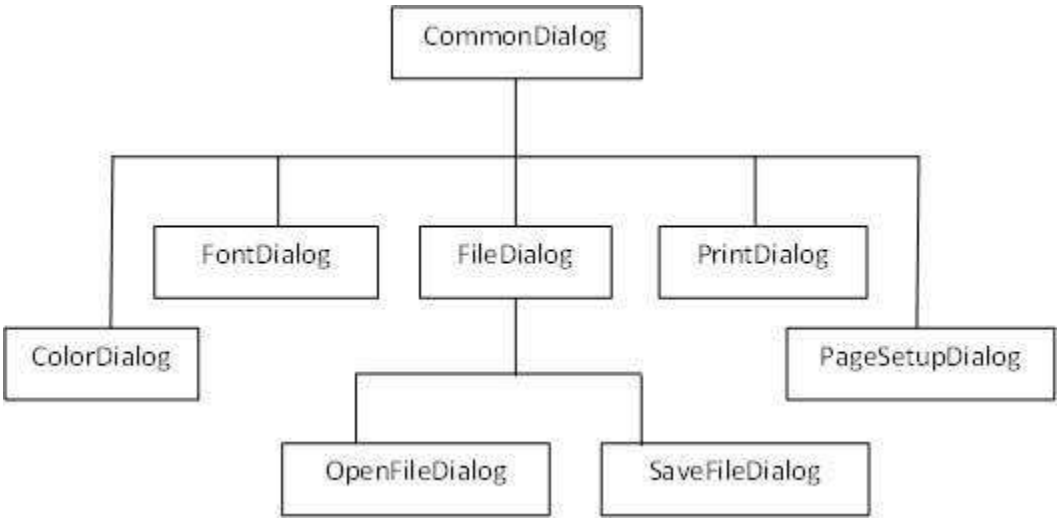
All of these dialog box control classes inherit from the **CommonDialog** class and override the *RunDialog()* function of the base class to create the specific dialog box.

The RunDialog() function is automatically invoked when a user of a dialog box calls its *ShowDialog()* function.

The **ShowDialog** method is used to display all the dialog box controls at run-time. It returns a value of the type of **DialogResult** enumeration. The values of DialogResult enumeration are −

- **Abort** − returns DialogResult.Abort value, when user clicks an Abort button.
- **Cancel** − returns DialogResult.Cancel, when user clicks a Cancel button.
- **Ignore** − returns DialogResult.Ignore, when user clicks an Ignore button.
- **No** − returns DialogResult.No, when user clicks a No button.
- **None** − returns nothing and the dialog box continues running.
- **OK** − returns DialogResult.OK, when user clicks an OK button
- **Retry** − returns DialogResult.Retry , when user clicks an Retry button
- **Yes** − returns DialogResult.Yes, when user clicks an Yes button

The following diagram shows the common dialog class inheritance −



All these above-mentioned classes have corresponding controls that could be added from the Toolbox during design time. You can include relevant functionality of these classes to your application, either by instantiating the class programmatically or by using relevant controls.

When you double click any of the dialog controls in the toolbox or drag the control onto the form, it appears in the Component tray at the bottom of the Windows Forms Designer, they do not directly show up on the form.

The following table lists the commonly used dialog box controls. Click the following links to check their detail

Sr.No.	Control & Description
1	ColorDialog It represents a common dialog box that displays available colors along with controls that enable the user to define custom colors.
2	FontDialog It prompts the user to choose a font from among those installed on the local computer and lets the user select the font, font size, and color.
3	OpenFileDialog It prompts the user to open a file and allows the user to select a file to open.
4	SaveFileDialog It prompts the user to select a location for saving a file and allows the user to

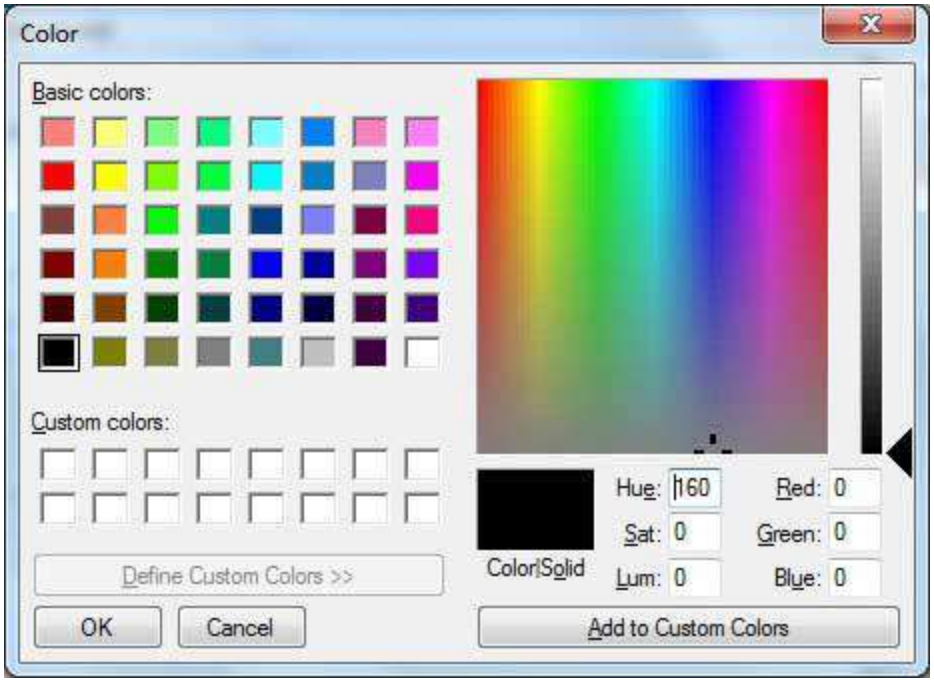
	specify the name of the file to save data.
5	PrintDialog It lets the user to print documents by selecting a printer and choosing which sections of the document to print from a Windows Forms application.

ColorDialog Control :

The ColorDialog control class represents a common dialog box that displays available colors along with controls that enable the user to define custom colors. It lets the user select a color.

The main property of the ColorDialog control is *Color*, which returns a **Color** object.

Following is the Color dialog box −



Properties of the ColorDialog Control

The following are some of the commonly used properties of the ColorDialog control −

Sr.No.	Property & Description
1	**AllowFullOpen** Gets or sets a value indicating whether the user can use the dialog box to define custom colors.
2	**AnyColor**

Gets or sets a value indicating whether the dialog box displays all available colors in the set of basic colors.

3 **CanRaiseEvents**
Gets a value indicating whether the component can raise an event.

4 **Color**
Gets or sets the color selected by the user.

5 **CustomColors**
Gets or sets the set of custom colors shown in the dialog box.

6 **FullOpen**
Gets or sets a value indicating whether the controls used to create custom colors are visible when the dialog box is opened.

7 **ShowHelp**
Gets or sets a value indicating whether a Help button appears in the color dialog box.

8 **SolidColorOnly**
Gets or sets a value indicating whether the dialog box will restrict users to selecting solid colors only.

Methods of the ColorDialog Control

The following are some of the commonly used methods of the ColorDialog control −

Sr.No. Method Name & Description

1 **Reset**
Resets all options to their default values, the last selected color to black, and the custom colors to their default values.

2 **RunDialog**
When overridden in a derived class, specifies a common dialog box.

3 **ShowDialog**
Runs a common dialog box with a default owner.

Events of the ColorDialog Control

The following are some of the commonly used events of the ColorDialog control −

Sr.No. Event & Description

1	**HelpRequest** Occurs when the user clicks the Help button on a common dialog box.	

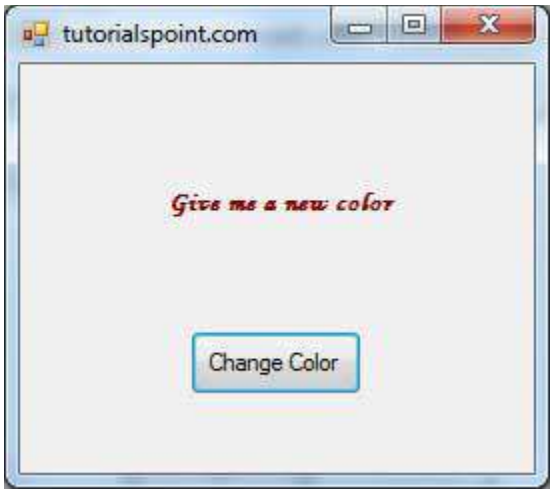
Example

In this example, let's change the forecolor of a label control using the color dialog box. Take the following steps –

- Drag and drop a label control, a button control and a ColorDialog control on the form.
- Set the Text property of the label and the button control to 'Give me a new Color' and 'Change Color', respectively.
- Change the font of the label as per your likings.
- Double-click the Change Color button and modify the code of the Click event.

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
  If ColorDialog1.ShowDialog <> Windows.Forms.DialogResult.Cancel Then
    Label1.ForeColor = ColorDialog1.Color
  End If
End Sub
```

When the application is compiled and run using **Start** button available at the Microsoft Visual Studio tool bar, it will show the following window −



Clicking on the Change Color button, the color dialog appears, select a color and click the OK button. The selected color will be applied as the forecolor of the text of the label.

FontDialog Control:

It prompts the user to choose a font from among those installed on the local computer and lets the user select the font, font size, and color. It returns the Font and Color objects.

Following is the Font dialog box −

By default, the Color ComboBox is not shown on the Font dialog box. You should set the **ShowColor** property of the FontDialog control to be **True**.

Properties of the FontDialog Control

The following are some of the commonly used properties of the FontDialog control −

Sr.No.	Property & Description
1	**AllowSimulations** Gets or sets a value indicating whether the dialog box allows graphics device interface (GDI) font simulations.
2	**AllowVectorFonts** Gets or sets a value indicating whether the dialog box allows vector font selections.
3	**AllowVerticalFonts** Gets or sets a value indicating whether the dialog box displays both vertical and horizontal fonts, or only horizontal fonts.
4	**Color** Gets or sets the selected font color.
5	**FixedPitchOnly** Gets or sets a value indicating whether the dialog box allows only the selection of fixed-pitch fonts.
6	**Font** Gets or sets the selected font.
7	**FontMustExist** Gets or sets a value indicating whether the dialog box specifies an error condition if the user attempts to select a font or style that does

not exist.

8	**MaxSize** Gets or sets the maximum point size a user can select.
9	**MinSize** Gets or sets the minimum point size a user can select.
10	**ScriptsOnly** Gets or sets a value indicating whether the dialog box allows selection of fonts for all non-OEM and Symbol character sets, as well as the ANSI character set.
11	**ShowApply** Gets or sets a value indicating whether the dialog box contains an **Apply** button.
12	**ShowColor** Gets or sets a value indicating whether the dialog box displays the color choice.
13	**ShowEffects** Gets or sets a value indicating whether the dialog box contains controls that allow the user to specify strikethrough, underline, and text color options.
14	**ShowHelp** Gets or sets a value indicating whether the dialog box displays a Help button.

Methods of the FontDialog Control

The following are some of the commonly used methods of the FontDialog control −

Sr.No.	Method Name & Description
1	**Reset** Resets all options to their default values.
2	**RunDialog** When overridden in a derived class, specifies a common dialog box.
3	**ShowDialog** Runs a common dialog box with a default owner.

Events of the FontDialog Control

The following are some of the commonly used events of the FontDialog control −

Sr.No.	Event & Description
1	**Apply** Occurs when the Apply button on the font dialog box is clicked.

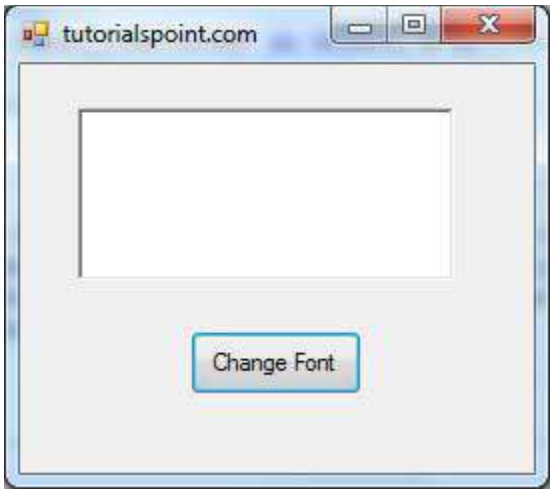
Example

In this example, let's change the font and color of the text from a rich text control using the Font dialog box. Take the following steps −

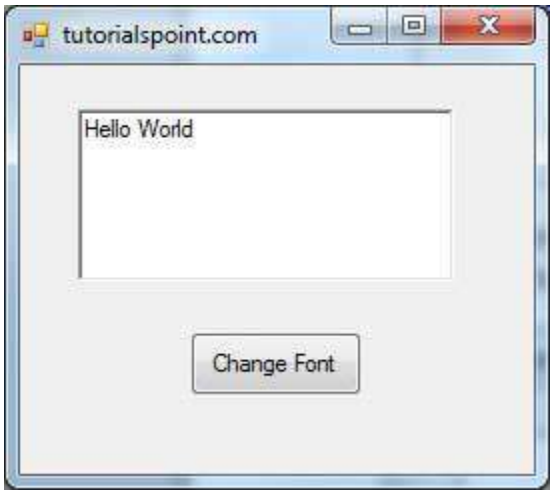
- Drag and drop a RichTextBox control, a Button control and a FontDialog control on the form.
- Set the Text property of the button control to 'Change Font'.
- Set the ShowColor property of the FontDialog control to True.
- Double-click the Change Color button and modify the code of the Click event −

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
  If FontDialog1.ShowDialog <> Windows.Forms.DialogResult.Cancel Then
    RichTextBox1.ForeColor = FontDialog1.Color
    RichTextBox1.Font = FontDialog1.Font
  End If
End Sub
```

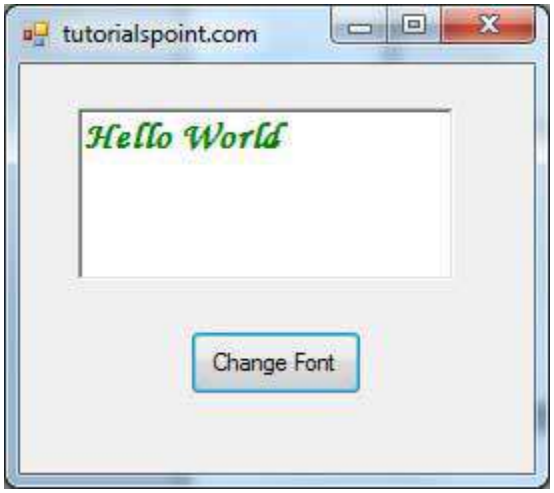
When the application is compiled and run using **Start** button available at the Microsoft Visual Studio tool bar, it will show the following window −



Enter some text and Click on the Change Font button.



The Font dialog appears, select a font and a color and click the OK button. The selected font and color will be applied as the font and fore color of the text of the rich text box.

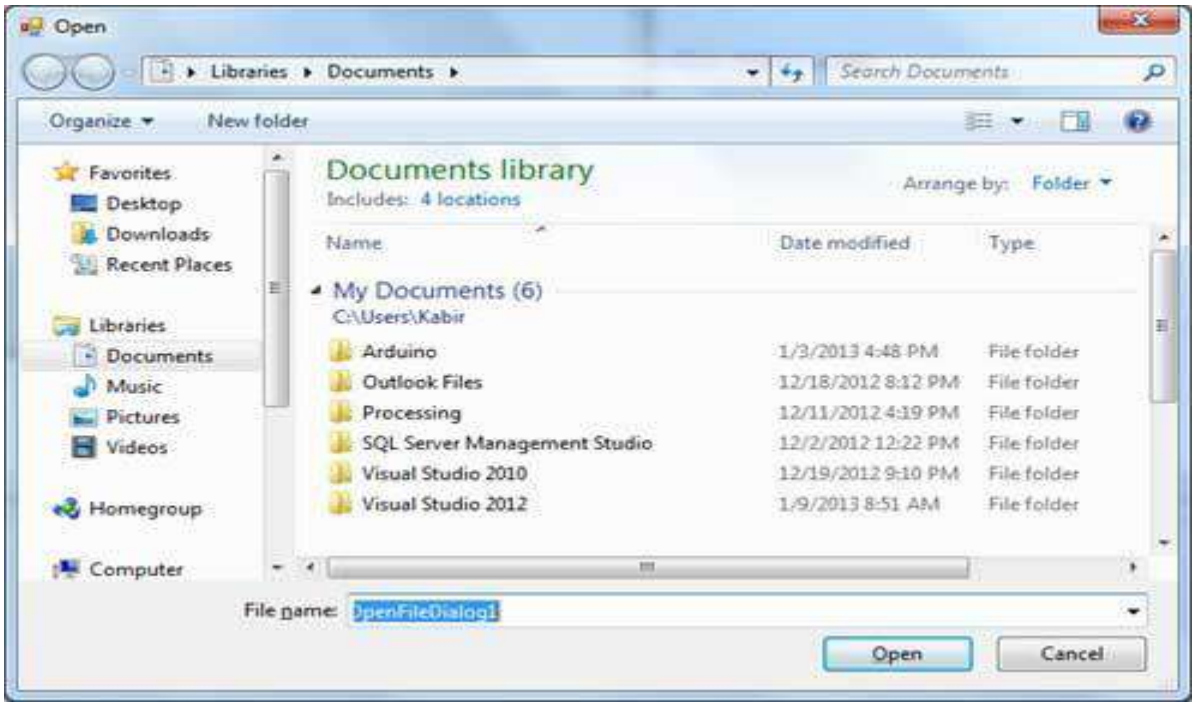


OpenFileDialog Control :

he **OpenFileDialog** control prompts the user to open a file and allows the user to select a file to open. The user can check if the file exists and then open it. The OpenFileDialog control class inherits from the abstract class **FileDialog**.

If the ShowReadOnly property is set to True, then a read-only check box appears in the dialog box. You can also set the ReadOnlyChecked property to True, so that the read-only check box appears checked.

Following is the Open File dialog box −



Properties of the OpenFileDialog Control

The following are some of the commonly used properties of the OpenFileDialog control −

Sr.No.	Property & Description
1	**AddExtension** Gets or sets a value indicating whether the dialog box automatically adds an extension to a file name if the user omits the extension.
2	**AutoUpgradeEnabled** Gets or sets a value indicating whether this FileDialog instance should automatically upgrade appearance and behavior when running on Windows Vista.
3	**CheckFileExists** Gets or sets a value indicating whether the dialog box displays a warning if the user specifies a file name that does not exist.
4	**CheckPathExists** Gets or sets a value indicating whether the dialog box displays a warning if the user specifies a path that does not exist.

5 **CustomPlaces**
Gets the custom places collection for this FileDialog instance.

6 **DefaultExt**
Gets or sets the default file name extension.

DereferenceLinks
7 Gets or sets a value indicating whether the dialog box returns the location of the file referenced by the shortcut or whether it returns the location of the shortcut (.lnk).

FileName
8 Gets or sets a string containing the file name selected in the file dialog box.

9 **FileNames**
Gets the file names of all selected files in the dialog box.

Filter
10 Gets or sets the current file name filter string, which determines the choices that appear in the "Save as file type" or "Files of type" box in the dialog box.

FilterIndex
11 Gets or sets the index of the filter currently selected in the file dialog box.

12 **InitialDirectory**
Gets or sets the initial directory displayed by the file dialog box.

Multiselect
13 Gets or sets a value indicating whether the dialog box allows multiple files to be selected.

ReadOnlyChecked
14 Gets or sets a value indicating whether the read-only check box is selected.

RestoreDirectory
15 Gets or sets a value indicating whether the dialog box restores the current directory before closing.

SafeFileName
16 Gets the file name and extension for the file selected in the dialog box. The file name does not include the path.

SafeFileNames
17 Gets an array of file names and extensions for all the selected files in the dialog box. The file names do not include the path.

18 **ShowHelp**
Gets or sets a value indicating whether the Help button is

displayed in the file dialog box.

19	**ShowReadOnly** Gets or sets a value indicating whether the dialog box contains a read-only check box.
20	**SupportMultiDottedExtensions** Gets or sets whether the dialog box supports displaying and saving files that have multiple file name extensions.
21	**Title** Gets or sets the file dialog box title.
22	**ValidateNames** Gets or sets a value indicating whether the dialog box accepts only valid Win32 file names.

Methods of the OpenFileDialog Control

The following are some of the commonly used methods of the OpenFileDialog control −

Sr.No.	Method Name & Description
1	**OpenFile** Opens the file selected by the user, with read-only permission. The file is specified by the FileName property.
2	**Reset** Resets all options to their default value.

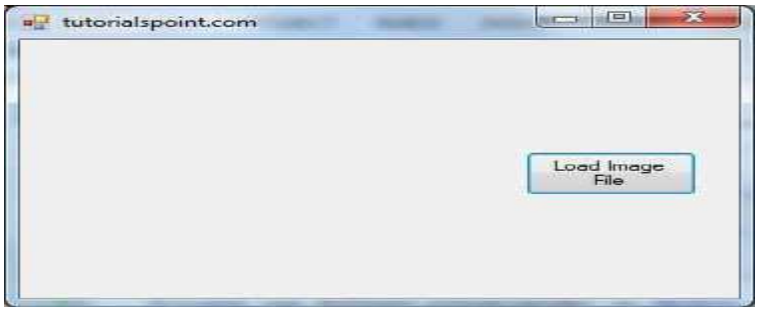
Example

In this example, let's load an image file in a picture box, using the open file dialog box. Take the following steps

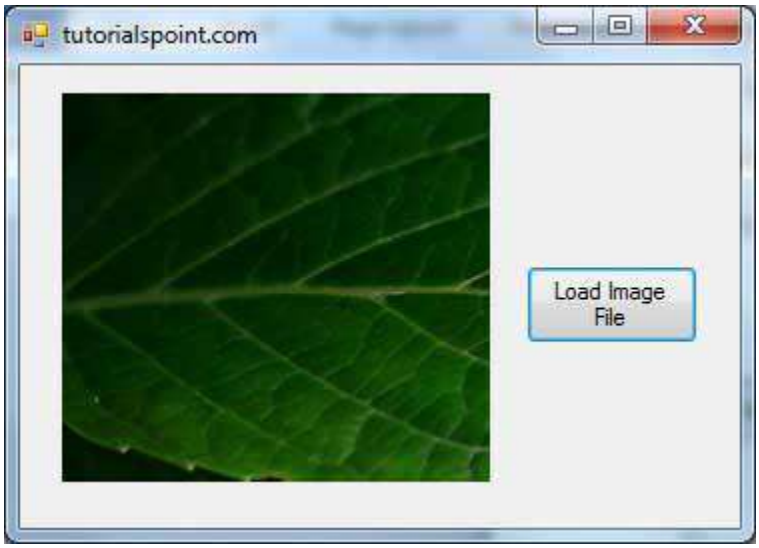
- Drag and drop a PictureBox control, a Button control and a OpenFileDialog control on the form.
- Set the Text property of the button control to 'Load Image File'.
- Double-click the Load Image File button and modify the code of the Click event:

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
  If OpenFileDialog1.ShowDialog <> Windows.Forms.DialogResult.Cancel Then
    PictureBox1.Image = Image.FromFile(OpenFileDialog1.FileName)
  End If
End Sub
```

When the application is compiled and run using **Start** button available at the Microsoft Visual Studio tool bar, it will show the following window −



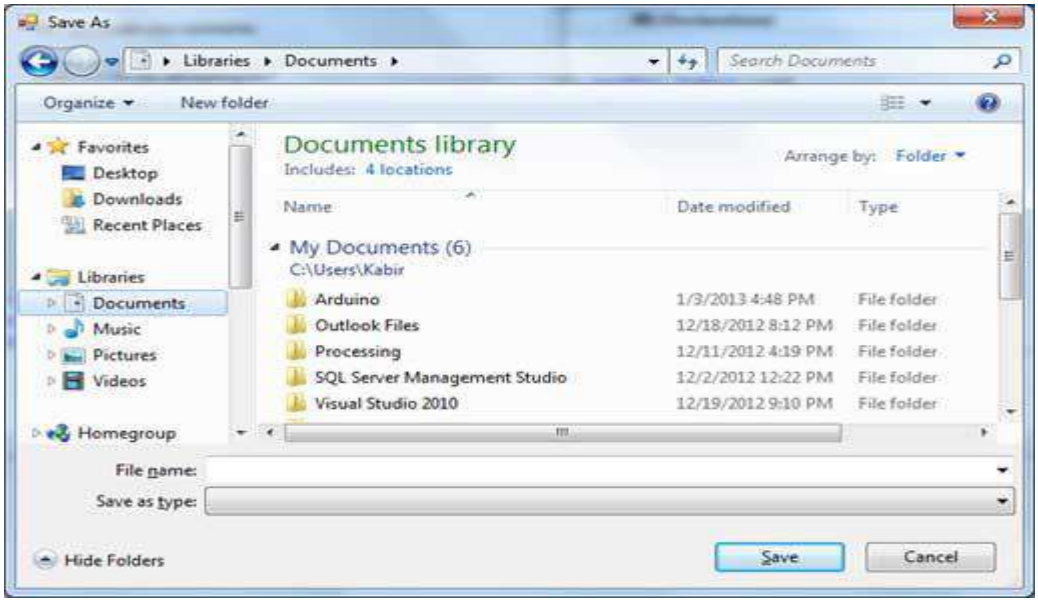
Click on the Load Image File button to load an image stored in your computer.



SaveFileDialog Control :

The **SaveFileDialog** control prompts the user to select a location for saving a file and allows the user to specify the name of the file to save data. The SaveFileDialog control class inherits from the abstract class FileDialog.

Following is the Save File dialog box −



Properties of the SaveFileDialog Control

The following are some of the commonly used properties of the SaveFileDialog control −

Sr.No.	Property & Description
1	**AddExtension** Gets or sets a value indicating whether the dialog box automatically adds an extension to a file name if the user omits the extension.
2	**CheckFileExists** Gets or sets a value indicating whether the dialog box displays a warning if the user specifies a file name that does not exist.
3	**CheckPathExists** Gets or sets a value indicating whether the dialog box displays a warning if the user specifies a path that does not exist.
4	**CreatePrompt** Gets or sets a value indicating whether the dialog box prompts the user for permission to create a file if the user specifies a file that does not exist.
5	**DefaultExt** Gets or sets the default file name extension.
6	**DereferenceLinks** Gets or sets a value indicating whether the dialog box returns the location of the file referenced by the shortcut or whether it returns the location of the shortcut (.lnk).
7	**FileName**

Gets or sets a string containing the file name selected in the file dialog box.

8 **FileNames**
Gets the file names of all selected files in the dialog box.

9 **Filter**
Gets or sets the current file name filter string, which determines the choices that appear in the "Save as file type" or "Files of type" box in the dialog box.

10 **FilterIndex**
Gets or sets the index of the filter currently selected in the file dialog box.

11 **InitialDirectory**
Gets or sets the initial directory displayed by the file dialog box.

12 **OverwritePrompt**
Gets or sets a value indicating whether the Save As dialog box displays a warning if the user specifies a file name that already exists.

13 **RestoreDirectory**
Gets or sets a value indicating whether the dialog box restores the current directory before closing.

14 **ShowHelp**
Gets or sets a value indicating whether the Help button is displayed in the file dialog box.

15 **SupportMultiDottedExtensions**
Gets or sets whether the dialog box supports displaying and saving files that have multiple file name extensions.

16 **Title**
Gets or sets the file dialog box title.

17 **ValidateNames**
Gets or sets a value indicating whether the dialog box accepts only valid Win32 file names.

Methods of the SaveFileDialog Control

The following are some of the commonly used methods of the SaveFileDialog control −

Sr.No. Method Name & Description

1 **OpenFile**
Opens the file with read/write permission.

2 **Reset**
Resets all dialog box options to their default values.

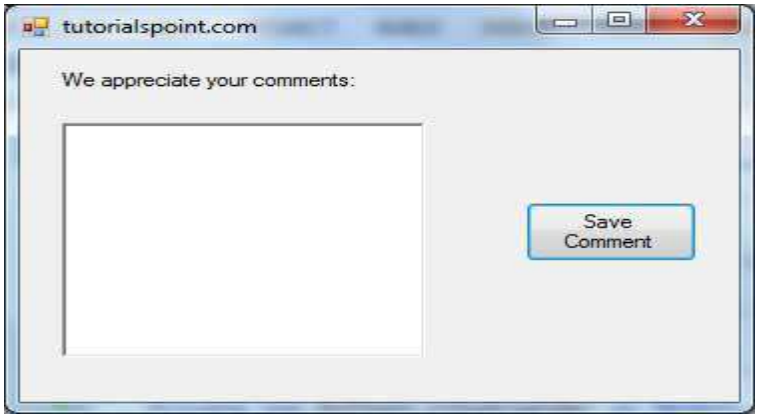
Example

In this example, let's save the text entered into a rich text box by the user using the save file dialog box. Take the following steps −

- Drag and drop a Label control, a RichTextBox control, a Button control and a SaveFileDialog control on the form.
- Set the Text property of the label and the button control to 'We appreciate your comments' and 'Save Comments', respectively.
- Double-click the Save Comments button and modify the code of the Click event as shown −

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
  SaveFileDialog1.Filter = "TXT Files (*.txt*)|*.txt"
    If SaveFileDialog1.ShowDialog = Windows.Forms.DialogResult.OK _
            Then
      My.Computer.FileSystem.WriteAllText _
      (SaveFileDialog1.FileName, RichTextBox1.Text, True)
    End If
End Sub
```

When the application is compiled and run using Start button available at the Microsoft Visual Studio tool bar, it will show the following window −



We have set the Filter property of the SaveFileDialog control to display text file types with .txt extensions only.

Write some text in the text box and click on the Save Comment button to save the text as a text file in your computer.

Modal And Modeless

Model dialog boxes forces the user to acknowledge the dialog before moving before moving onto the application. **Modeless** dialog boxes enable the user to interact with the dialog and the application interchangeably.

A modal dialog box doesn't allow the user to access the parent window while the dialog is open – it must be dealt with and closed before continuing. A modeless dialog can be open in the background.

Example for Model Dialog is Save, Save As Dialog in MS – Word. while it is opening you can't do anything in the application until you close that window. Example for Modeless Dialog is Find, Replace dialogs. You can use Find Dialog, same time you can also work in that word application.

A modeless dialog box has the following characteristics

- It has a thin border
- It can be neither minimized nor maximized. This means that it is not equipped with the Minimize or the Maximize buttons
- It is not represented on the taskbar with a button
- It must provide a way for the user to close it

What is an Exception? Explain types of exception

An exception is a problem that arises during the execution of a program. An exception is a response to an exceptional circumstance that arises while a program is running, such as an attempt to divide by zero.Exceptions provide a way to transfer control from one part of a program to another.

1. Structured Exception Handling

In structured exception handling, blocks of code are encapsulated, with each block having one or more associated handlers. Each handler specifies some form of filter condition on the type of exception it handles. When an exception is raised by code in a protected block, the set of corresponding handlers is searched in order, and the first one with a matching filter condition is executed. A single method can have multiple structured exception handling blocks, and the blocks can also be nested within each other.

The **Try...Catch...Finally** statement is used specifically for structured exception handling

- **Try** − A Try block identifies a block of code for which particular exceptions will be activated. It's followed by one or more Catch blocks.
- **Catch** − A program catches an exception with an exception handler at the place in a program where you want to handle the problem. The Catch keyword indicates the catching of an exception.
- **Finally** − The Finally block is used to execute a given set of statements, whether an exception is thrown or not thrown. For example, if you open a file, it must be closed whether an exception is raised or not.
- **Throw** − A program throws an exception when a problem shows up. This is done using a Throw keyword.

Try Dim i As Integer Dim resultValue As Integer	When you execute the program you will get the message box "Exception catch here ..." Because there is a divide by 0 exceptions occurs.

i = 100 resultValue = i / 0 MsgBox("The result is " & resultValue) Catch ex As Exception MsgBox("Exception catch here ..") Finally MsgBox("Finally block executed ") End Try	

2. **UnstructuredException Handling**

The **On Error** statement is used specifically for unstructured exception handling. In unstructured exception handling, **On Error** is placed at the beginning of a block of code. It then has "scope" over that block; it handles any errors occurring within the block. If the program encounters another **On Error** statement, that statement becomes valid and the first statement becomes invalid.

Button1_Click On Error GoTo nextstep Dim result As Integer Dim num As Integer num = 100 result = num / 0 nextstep: MsgBox("Control Here") End Sub	When you execute the program you will get the message box "Control Here" Because the On Error statement redirect the exception to the Label statement.

List of Standard Exceptions

Exception	Description
OutOfMemoryException	The exception that is thrown when there is not enough memory to continue the execution of a program.
OverflowException	The exception that is thrown when an arithmetic, casting, or conversion operation in a checked context results in an overflow.
StackOverflowException	The exception that is thrown when the execution stack overflows by having too many pending method calls. This class cannot be inherited.
NullReferenceException	The exception that is thrown when there is an attempt to dereference a null object reference.

IndexOutOfRangeException	The exception that is thrown when an attempt is made to access an element of an array with an index that is outside the bounds of the array. This class cannot be inherited.
FormatException	The exception that is thrown when the format of an argument does not meet the parameter specifications of the invoked method.
DivideByZeroException	The exception that is thrown when there is an attempt to divide an integral or decimal value by zero.
DllNotFoundException	The exception that is thrown when a DLL specified in a DLL import cannot be found.
ArgumentNullException	The exception that is thrown when a null reference (**Nothing** in Visual Basic) is passed to a method that does not accept it as a valid argument.
ArgumentOutOfRangeException	The exception that is thrown when the value of an argument is outside the allowable range of values as defined by the invoked method.
ArithmeticException	The exception that is thrown for errors in an arithmetic, casting, or conversion operation.
ArgumentException	The exception that is thrown when one of the arguments provided to a method is not valid.

Explain SDI & MDI Applications.

The SDI and MDI forms are the interface design for document handling within a single Windows application. The MDI stands for Multiple Document Interface whereas SDI stands for Single Document Interface.

MDI: A multiple document Interface is one that allows viewing multiple windows within a large window.

SDI: A single Document Interface is one where all Windows appear independently of one another without the unification of a single parent window.

The Visual Basic IDE can be viewed in two ways:

1. With the Multiple Document Interface (MDI)
2. Single Document Interface (SDI)

MDI view shows all the distinct windows of Visual Bassic IDE as child windows within on large IDE Window.

In the SDI view, distinct windows of the Visual Basic IDE exist independently of each other.

MDI Forms

- This is the main form or parent form which is not duplicated, but acts like a container for all the Windows which is also called the primary window.

- The windows in which the individual documents are displayed are called Child Windows.
- An MDI application must have atleast two form, the primary parent form and one or more child forms.
- The parent form may not contain any controls. While the parent Form is open in design mode, the icon on the tool box are not displayed, but you can't place any control on the form.
- The parent form usually have a menu.

Compare SDI & MDI

SDI (Single Document Interface) applications allow only one open document frame window at a time.
WordPad, Paint etc are examples of SDI applications.

MDI (Multi Document Interface) applications allow multiple document frame windows to be open in the same instance of an application. An MDI application has a window within which multiple MDI child windows, which are frame windows themselves, can be opened, each containing a separate document. In some applications, the child windows can be of different types, such as chart windows and spreadsheet windows. In that case, the menu bar can change as MDI child windows of different types are activated.
Word,Excel etc are examples of MDI applications.

Unit 4. Object Oriented Programming

What is Object Oriented programming?

Object oriented programming treats data as a critical element in the program development and does not allow it to flow freely around the system .it ties data more closely to the function that operate on it and protects it from accidental modification.

Classes and Objects

⇒ The idea is that classes are a type, and objects are examples or instances of that class.

⇒ It's easy to create classes and objects in Visual Basic.

⇒ To create a class, you only need to use the Class statement, which, like other compound statements in Visual Basic, needs to end with End Class.

General Syntax

```
Public Class DataClass
        --statementEnd
Class
```

⇒ This creates a new class named DataClass. You can create an object of this class,you must use theNew keyword to create a new instance of a class(object).

```
Dim data As New DataClass()
```

⇒ You also can do this like this:

```
Dim data As DataClass = New DataClass()
```

OOP Features of VB.Net

VB.Net is an event driven programming language. It has a GUI interface. It supports both Console and Window based application. It is pure OOP (Object Oriented Language). It supports following features.

1. **Inheritance** Visual Basic .NET supports inheritance by allowing you to define classes that serve as the basis for derived classes. Derived classes inherit and can extend the properties and methods of the base class. They can also override inherited methods with new implementations. All classes created with Visual Basic .NET are inheritable by default.

2. **Exception Handling** Visual Basic .NET supports structured exception handling, using an enhanced version of the Try...Catch...Finally syntax supported by other languages such as C++. Structured exception handling combines a modern control structure (similar to Select Case or While) with exceptions, protected blocks of code, and filters. Structured exception handling makes it easy to create and maintain programs with robust, comprehensive error handlers.

3. **Overloading** It is the ability to define properties, methods, or procedures that have the same name but use different data types. Overloaded procedures allow you to provide as many implementations as necessary to handle different kinds of data, while giving the appearance of a single, versatile procedure.

4. **Overriding** Properties and Methods The Overrides keyword allows derived objects to override characteristics inherited from parent objects. Overridden members have the same arguments as the members inherited from the base class, but different implementations. A member's new implementation can call the original implementation in the parent class by preceding the member name with MyBase.

5. **Constructors and Destructors** Constructors are procedures that control initialization of new instances of a class. Conversely, destructors are methods that free system resources when a class leaves scope or is set to Nothing. Visual Basic .NET supports constructors and destructors using the Sub New and Sub Finalize procedures.

6. **Interfaces**describe the properties and methods of classes, but unlike classes, do not provide implementations. The Interface statement allows you to declare interfaces, while the Implements statement lets you write code that puts the items described in the interface into practice

7. **Delegates**objects that can call the methods of objects on your behalf — are sometimes described as type-safe, object-oriented function pointers. You can use delegates to let procedures specify an event handler method that runs when an event occurs. You can also use delegates with multithreaded applications.

8. **Shared Members** are properties, procedures, and fields that are shared by all instances of a class. Shared data members are useful when multiple objects need to use information that is common to all. Shared class methods can be used without first creating an object from a class.

9. **Namespaces** prevent naming conflicts by organizing classes, interfaces, and methods into hierarchies.

10. **Assemblies** replace and extend the capabilities of type libraries by, describing all the required files for a particular component or application. An assembly can contain one or more namespaces.

11. **Multithreading** Visual Basic .NET allows you to write applications that can perform multiple tasks independently. A task that has the potential of holding up other tasks can execute on a separate thread, a process known as multithreading.

Define Classes and Objects

- The general meaning of **class** is category. Class is a pro-forma or blue-print that represents particular category. Classes are made of fields, properties, methods, and events. Classes are derived or user defined data type

- **Objects** are the basic runtime entities in object oriented system. Objects are also called Instance of the class. An objects represents a person, Bank-account, Vehicle,
  Book, products, Employee, etc…

1. Fields and properties represent information that an object contains. Fields are like variables in that they can be read or set directly. For example, if you have an object named "Car" you could store its color in a field named "Color."
2. Properties are retrieved and set like fields, but are implemented using property Get and property Set procedures, which provide more control on how values are set or returned. The layer of indirection between the value being stored and the procedures that use this value helps isolate your data and allows you to validate values before they are assigned or retrieved.
3. Methods represent actions that an object can perform. For example, a "Car" object could have "StartEngine," "Drive," and "Stop" methods. You define methods by adding procedures, either Sub routines or functions, to your class.
4. Events are notifications an object receives from, or transmits to, other objects or applications. Events allow objects to perform actions whenever a specific occurrence takes place. An example of an event for the "Car" class would be a "Check_Engine" event. Because Microsoft Windows is an event-driven operating system, events can come from other objects, applications, or user input such as mouse clicks or key presses.

Define Constructor and Destructer

- A constructor is a "Special" member function whose task is to initialize the object of its class.
- The constructor is invoked whenever an object of its associated class is created.
- It is called constructor because it constructs the values of data members of the classes.

Destructor:

- A Destructor , as the name implies , is used to destroy the objects that have been created by a constructor

PublicClassEmp	

```
'member declaration

DimempnoAsInteger
DimempnameAsString

' default constructor

PublicSubNew()
empno = 0
empname = ""
EndSub
'parametrized constructor

PublicSubNew(ByVal EMPNO AsInteger, ByVal EMPNAME
AsString)
Me.empname = EMPNAME
Me.empno = EMPNO
EndSub

' show method

PublicSub SHOW()
MsgBox("EMPNO = "&empno)
MsgBox("EMPNAME = "&empname)
EndSub

'destructor method 1

PublicSub dispose()
MsgBox("DESTROY")
EndSub

'destructor method 2ProtectedOverridesSub finalize()
MsgBox("object is being destroy") EndSub
EndClass
```

```
'default constructor call

Dim e2 AsNewEmp()
e2.SHOW()
e2.dispose()

'parametrized constructor call

Dim e1 AsNewEmp(101, "ASDSA")
 e1.SHOW()
```

Property Procedure:-

A property procedure is a series of Visual Basic statements that allow a programmer to create and manipulate custom properties.

A Property is similar to a Function. With a getter and a setter, it controls access to a value.

Property procedures should be used instead of **Public** variables in code that must be executed when the property value is set.

With Get, a property returns a value. With Set it stores a value. We must have both Get and Set unless we specify ReadOnly or WriteOnly on the property.

ReadOnly: Some properties are not meant to be assigned. The ReadOnly modifier changes the Property type to only have a Get method.

WriteOnly: Here we use the WriteOnly keyword on a property. WriteOnly means that a Property has only a Set() method and no Get method.

```
Classcircle
Dim r AsInteger

PublicProperty radius() AsInteger
Get
Return r
EndGet
Set(ByVal value AsInteger)
        r = value
EndSet
EndProperty

PublicReadOnlyProperty pi() AsSingle
Get
Return 3.14
EndGet
EndProperty

PublicFunction area() AsSingle
Return pi * r * r
EndFunction

EndClass
```

```
Dim c AsNewcircle
c.radius = 5
MsgBox(c.pi)
MsgBox(c.area)
```

Data Encapsulation And Data Abstraction :-

- The wrapping up of data and function into a single unit (called class) is known as **encapsulation**.
- Data is not access to the outside word and only those functions which are wrapped in the class can access it. Thus, it provides interface between the object's data and the program. It is also called "Data hiding" or "Information hiding".
- **Data abstraction** refers to, providing only essential information to the outside world and hiding their background details, i.e., to represent the needed information in program without presenting the details.
- "Data Abstraction", as it gives a clear separation between properties of data type and the associated implementation details. There are two types; they are "function abstraction" and "data abstraction"
- Functions that can be used without knowing how it's implemented are function abstraction. Data abstraction is using data without knowing how the data is stored
- Abstraction and encapsulation are related features in object oriented programming. Abstraction allows making relevant information visible and encapsulation enables a programmer to *implement the desired level of abstraction*.
- Encapsulation is implemented by using **access specifiers**. An **access specifier** defines the scope and visibility of a class member.

- **Access Specifiers**

They describes as the accessibility scope of a variable, method or a class. By using access specifiers we can control the scope of the member object of a class. Access specifiers were used for providing security of our applications. In Visual Basic .Net there are five access specifiers and they are as follows:

Public: It have no restriction on accessibility. We can use the public members from any were inside the class or outside the class.

Private: Their accessibility scope is limited to only inside the class in which they are declared. We can't access the Private members from outside the class and it is the least permissive access level.

Protected: The protected members have scope of accessibility within the class and classes derived(Inherited) from that class.

Friend: Friend members have the accessibility scope from the same assembly and program that contain their declarations.

Protected Friend: It behave like both protected and friend access specifiers. We can access the protected friend member from anywhere in same assembly and the classes inherited from the same class.

Shared Keyword

A shared method is not accessed via an object instance like a regular method, but rather is accessed directly from the class.

The shared keyword in VB.NET is the equivalent of the static keyword in C.

In VB.NET, the shared keyword can only be applied to methods within a class,

A shared variable is declared using the Shared keyword, much like a shared method

Classmyutil PublicSharedFunction square(ByVal n1 AsInteger) AsInteger Return n1 * n1 EndFunction PublicSharedFunction fact(ByVal n1 AsInteger) AsInteger	MsgBox(myutil.square(5)) MsgBox(myutil.fact(4))
DimansAsInteger ans = 1 Fori = 1 To n1 ans = ans * i Next Returnans EndFunction End Class	

Inheritance:-

- Inheritance is the process by which the objects of one class acquire the properties of another class.
- It supports the concept of hierarchical – classification.
- In Oop, the concept of inheritance provides the ideas of Reusability.
- This means that we can add additional features to an existing class without modifying it.
- **Overriding in VB**.net is method by which a inherited property or a method is overidden to perform a different functionality in a derived class. The base class function is declared using a keyword Overridable and the derived class function where the functionality is changed contains an keyword Overrides.
- **Inherits** Any derived class must inherit an existing class. The Inherits statement tells the compiler which class it derives from, and it must be the first executable statement in the derived class's code. A class that doesn't include the Inherits keyword is by definition a base class.
- **Overridable** Every member with this modifier may be overwritten by the derived class. Members declared with the Overridable keyword don't necessarily need to be overridden, so they must provide some functionality.
- **NotOverridable** Every member declared with this modifier can't be overridden in the inheriting class.

- **Overrides** Use this keyword to specify the member of the parent class you're overriding. If a member has the same name in the derived class as in the parent class, this member must be overridden.You can't use the Overrides keyword with members that were declared with the NotOverridable in the base class.

- **MyBase** - This keyword can be used from any subclass to make a call to any property or method in the base class. You can even use it to call the base class' method that you have overridden in the subclass

- **Me -**The Me keyword is the similar to the this keyword of c++. The Me keyword is used to get the reference of the current type.The keyword Me refers to the current instance of an object.

Single Inheritance

```
Dimem AsNewexam(1, "abc", 50, 78, 67)
em.show()
```

```
PublicClassstud

Publicrno AsInteger
Publicsname AsString
PublicSubNew()
rno = 0
sname = ""
EndSub

PublicSubNew(ByValrno AsInteger,
ByValsname AsString)
Me.rno = rno
Me.sname = sname
EndSub

PublicOverridableSub show()
MsgBox("Rollno "&rno&" Student Name "&sname)
EndSub

EndClass
```

```
PublicClassexam
InheritsStud
Dim m1 AsInteger
Dim m2 AsInteger
Dim m3 AsInteger
PublicSubNew()
    m1 = 0
    m2 = 0
    m3 = 0
EndSub
PublicSubNew(ByValrno AsInteger,
ByValsname AsString, ByVal m1 AsInteger, ByVal m2
AsInteger, ByVal m3 AsInteger)

MyBase.New(rno, sname)

Me.m1 = m1
Me.m2 = m2
Me.m3 = m3

EndSub
PublicOverridesSub show()
Dim tot AsInteger
Dim per AsSingle
    tot = m1 + m2 + m3
    per = tot / 3
MyBase.show()
MsgBox("Total "& tot)
MsgBox("Per"& per)
EndSub
EndClass
```

```
'create base class
PublicClassclass1
PubliciAsString = "Hello"
PublicSub print1()
MsgBox("I value is "&i)
MsgBox("class 1")
EndSub
EndClass
'create derived class
PublicClassclass2
Inheritsclass1
PublicSub print2()
MyBase.print1()
MsgBox("class 2")
EndSub
EndClass
'new derived class
PublicClassclass3
Inheritsclass2
PublicSub print3()
MyBase.print2()
MsgBox("class 3")
EndSub
EndClass
```

Multilevel inheritance:

```
Dim c1 AsNewclass3()
      c1.print3()
```

```
'hierarchical
'create base class
PublicClassclass1
PublicSub  print1()
MsgBox("class 1")
EndSub
EndClass
'cderived class
PublicClassclass2
Inheritsclass1

PublicSub print2()
MyBase.print1()
MsgBox("class 2")
EndSub
EndClass
' derived class
PublicClassclass3
Inheritsclass1
PublicSub print3()
MyBase.print1()
MsgBox("class 3")
EndSub
EndClass
```

Hierarchical inheritance

```
Dim c2 AsNewclass2()
Dim c3 AsNewclass3()

      c2.print2()
      c3.print3()
```

Polymorphism:-

- Polymorphism simply means “One name And Multiple behavior”.
- Polymorphism plays an important role in allowing objects having different internal structure to share the same external interface.
- The overloaded member functions are selected for invoking by matching the arguments. In these both, data type and no. of arguments are matched. This information is known to the compiler at the time of compilation and compiler is able to select appropriate function for particular called at compile time this concept is called **Compile time polymorphism. (function overloading)**
- When appropriate member function is selected while program is running this is called **Runtime polymorphism.**
- **Overloads :**Specifies that a property or procedure redeclares one or more existing properties or procedures with the same name.

Compile time Polymorphism	
Classoverload Dim r AsDouble PublicOverloadsSub area(ByVal r) MsgBox("Area of the Circle :") MsgBox(1 / 3 * 3.14 * r * r * r) EndSub	Dim r AsNewoverload() r.area(3.1) r.area(4, 5)
Dim length AsInteger Dim width AsInteger PublicOverloadsSub area(ByVal length, ByVal width) MsgBox(" Area of the Rectangle :") MsgBox(length * width) EndSub EndClass	

Run time Polymorphism

```
PublicMustInheritClassShape

MustOverrideFunction area() AsInteger
MustOverrideFunctionvol() AsInteger

PublicSub  show()
MsgBox("Shape")
EndSub

EndClass
```

```
Dim s AsShape

Dim r AsNewrect(4, 5, 6)

s = r
MsgBox(s.area)
MsgBox(s.vol)

s.show()

Dim c AsNewmycircle()
c.radius = 5

      s = c
MsgBox(s.area)
MsgBox(s.vol)
```

```
PublicClassrect
InheritsShape

Dim l AsInteger
Dim b AsInteger
Dim h AsInteger

PublicSubNew(ByVal l AsInteger, ByVal b AsInteger,
ByVal h AsInteger)
Me.l = l
Me.b = b
Me.h = h
EndSub

PublicOverridesFunction area() AsInteger
Return l * b
EndFunction

PublicOverridesFunctionvol() AsInteger
Return l * b * h
EndFunction
```

```
Classmycircle
InheritsShape

Dim r AsInteger

PublicProperty radius() AsInteger
Get
Return r
EndGet
Set(ByVal value AsInteger)
        r = value
EndSet
EndProperty

PublicReadOnlyProperty pi() AsSingle
Get
Return 3.14
EndGet
EndProperty
```

```
EndClass
```

```
PublicOverridesFunction area() AsInteger
Return pi * r * r

EndFunction

PublicOverridesFunctionvol() AsInteger
Return 4 / 3 * pi * r * r * r
EndFunction
EndClass
```

- **MustInherit** This class must be inherited. You can't create an object of this class in your code and, therefore, you can't access its methods. (Like Abstract Class C++)

- **MustOverride** Every member declared with this modifier must be overridden. This means that the derived class must be inherited by some other class, which then receives the obligation to override the original member declared as MustOverride.

- **NotInheritable** Prevents the class from being inherited. No other classes can be derived from this class. The base data types, for example, are not inheritable. In other words one can't create a new class based on the Integer data type

```
PublicNotInheritableClasstest

PublicSub show()
MsgBox("Test")
EndSub

EndClass
```

Interface

- Interfaces, like classes, define a set of properties, methods, and events. But unlike classes, interfaces do not provide implementation.

- They are implemented by classes, and defined as separate entities from classes.

- An interface represents a contract, in that a class that implements an interface must implement every aspect of that interface exactly as it is defined.

- With interfaces, you can define features as small groups of closely related members. You can develop enhanced implementations for your interfaces

- Although interface implementations can evolve, interfaces themselves cannot be changed once published.

- To define interfaces we use the Interface statement, and to implement interfaces the Implements keyword can be used.

[accessmodifier] InterfaceInerfacename
[[modifiers] Property membername]
[[modifiers] Function membername]
[[modifiers] Sub membername]
	`'interface`
	`PublicInterfacei1`
	`Sub show()`
	`EndInterface`
	
	`'class`

End Interface	`PublicClasshello`
	`Implementsi1`
	`PrivateSub show() Implementsi1.show`
	`MsgBox("hello all")`
	`EndSub`
	`EndClass`

Multiple Inheritance in Vb.Net

Method 1

```
Dim d AsNewdisplay()
d.show()
      d.show1()
```

```
'interface 1 base
PublicInterfaceinterface1
Sub show()
EndInterface

'interface 2 base
PublicInterfaceinterface2
Sub show()
EndInterface

' class derived
Classdisplay
Implementsinterface1, interface2


PublicSub show() Implementsinterface1.show
MsgBox("interface 1")
EndSub

PublicSub show1() Implementsinterface2.show
MsgBox("interface 2")
EndSub

EndClass
```

Method 2

```
Dim f AsNewfinal()
      f.show1()
```

```
'interface 1 base
PublicInterfaceinterface1
Sub show()
EndInterface

'base class
Classclasshello
Sub show()
MsgBox("class hello...")
EndSub
EndClass
'class derived implement interface and inherit class
Classfinal
Inheritsclasshello
Implementsinterface1

PublicSub show1() Implementsinterface1.show
MyBase.show()
MsgBox("interface 1")
EndSub
EndClass
```

- **MyClass**– This keyword allows you to call an Overridable method implemented in class and make sure that implementation of the method in this class is called rather than an overridden method in a derived class. MyClass is a keyword, not a real object. MyClass cannot be assigned to a variable, passed to procedures, or used in an Is comparison. MyClass refers to the containing class and its inherited members. MyClass can be used as a qualifier for Shared members. MyClass cannot be used in standard modules.

```
'myclass keyword

ClassBaseClass
PublicOverridableSubMyMethod()
MsgBox("Base class string")
EndSub

PublicSubUseMe()
Me.MyMethod()  ' Use calling class's version, even if an
override.
EndSub

PublicSubUseMyClass()
MyClass.MyMethod()  ' Use this version and not any
override.
EndSub
EndClass

ClassDerivedClass
InheritsBaseClass

PublicOverridesSubMyMethod()
MsgBox("Derived class string")
EndSub
EndClass
```

```
Dim d AsNewDerivedClass()

d.UseMe()  ' Displays "Derived class string".

d.UseMyClass()  ' Displays "Base class string".
```

Early and Late Binding

The compiler performs a process called binding when an object is assigned to an object variable.

⇒ The early binding (static binding) refers to compile time binding and late binding (dynamic binding)refers to runtime binding.

Early Binding (Static binding)

⇒ When perform Early Binding, an object is assigned to a variable declared to be of a specific objecttype. Early binding objects are basically a strong type objects or static type objects.

⇒ While Early Binding, methods, functions and properties which are detected and checked duringcompile time and perform other optimizations before an application executes.

⇒ The biggest advantage of using early binding is for performance and ease of development.

Late binding (Dynamic binding)

⇒ By contrast, in late binding functions, methods, variables and properties are detected and checked only at the run-time. It implies that the compiler does not know what kind of object or actual type ofan object or which methods or properties an object contains until run time.

⇒ The biggest advantages of late binding are that the Objects of this type can hold references to anyobject.

MustInherit Keyword (Creating Abstract Classes) OR How to create abstract class in vb.net.

⇒ Using the MustInherit Keyword we can create abstract class.The MustInherit keyword is used to declare a class that cannot be instantiated and can be used only as a base class, also called an abstractbase class.

Example

```
Public MustInherit Class Animal
   Protected MainForm As Form1
   Public Sub New(ByVal form1 As Form1)
      MainForm = form1
   End Sub

   Public Overridable Sub Breathing()
      MainForm.TextBox1.Text = "Breathing..."
   End SubEnd
Class

Public Class Dog Inherits
   Animal
                  //
End Class

Public Class Fish
   Inherits Animal
      // End Class
```

⇒ In above code, I derive two classes, Fish and Dog, from the Animal class. To make Animal an abstractclass, all I have to do is to use the MustInherit keyword

Me, MyBase and MyClass keywords

Me keyword

⇒ The Me keyword provides a way to refer to the specific instance of a class or structure in which the code is currently executing. Me behave like either an object variable or a structure variable referring to the current instance.

⇒ Using Me is particularly useful for passing information about the currently executing instance of aclass or structure to a procedure in another class, structure, or module.

MyBase Keyword

⇒ You can use the MyBase keyword to access methods in a base class when overriding methods in aderived class.

⇒ For example, suppose you are designing a derived class that overrides a method inherited from thebase class; in this case, the overridden method can call the original method in the base class using MyBase.

MyClass keyword

⇒ The MyClass keyword behaves like an object variable referring to the current instance of a class as originally implemented.

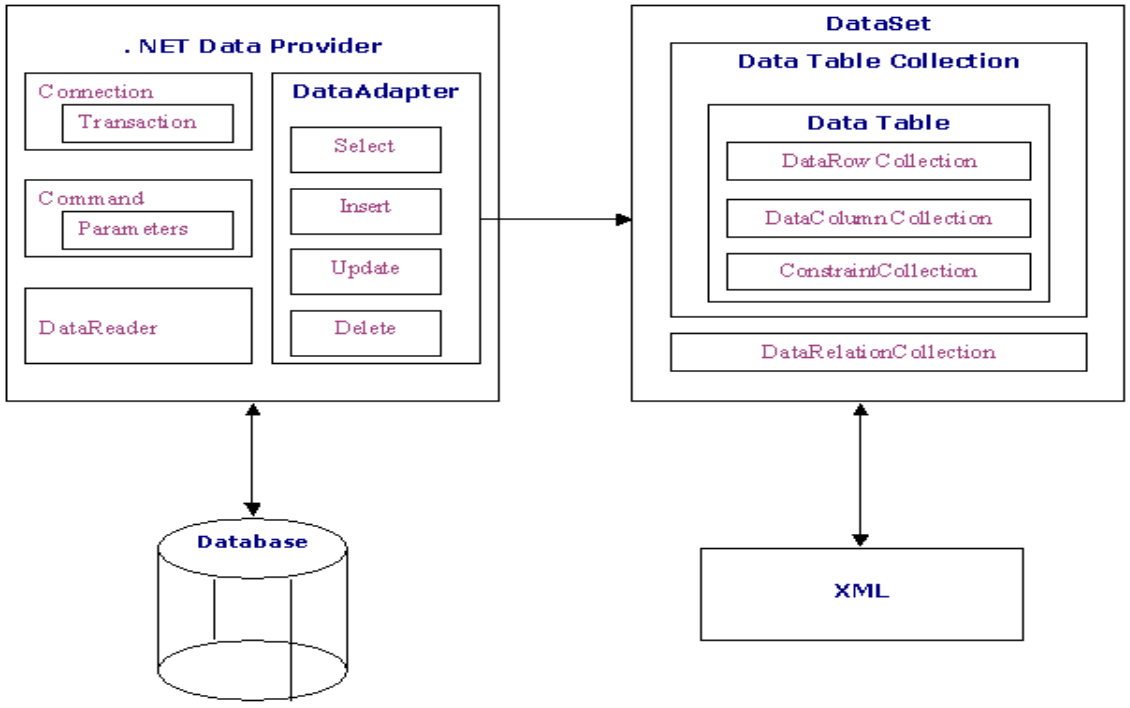
⇒ MyClass is similar to Me, but all method calls on it are treated as if the method were NotOverridable.

The ADO.NET Data Architecture

The ADO.NET is designed to work with multiple kinds of data sources in same fashion. You can categorize ADO.NET components in two categories: disconnected and the .NET data providers. The disconnected components build the basic ADO.NET architecture. You can use these components (or classes) with or without data providers.

The data provider components are specifically designed to work with different kinds of data sources. For example, ODBC data providers work with ODBC data sources and OleDb data providers work with OLE-DB(object linking and embedding database) data sources and Sql data providers work with MSSQL data sources and OracleDB data providers work with Oracle-DB data sources.

Data Access in ADO.NET relies on two components: DataSet and Data Provider.



ADO .NET Data Architecture

DataSet

The dataset is a disconnected, in-memory representation of data. It can be considered as a local copy of the relevant portions of the database. The DataSet is persisted in memory and the data in it can be manipulated and updated independent of the database. When the use of this DataSet is finished, changes can be made back to the central database for updating.

Data Provider

The Data Provider is responsible for providing and maintaining the connection to the database. A DataProvider is a set of related components that work together to provide data in an efficient and performance driven manner. The .NET Framework currently comes with Four DataProviders: the SQL Data Provider, ODBC Data Provider, OLEDB Data Provider, Oracle Data Provider. Each DataProvider consists of the following component classes:

The Connection object which provides a connection to the database
The Command object which is used to execute a command
The DataReader object which provides a forward-only, read only, connected recordset
The DataAdapter object which populates a disconnected DataSet with data and performs updated using the command object or the DataAdapter.

Component classes that make up the Data Providers

The Connection Object

The Connection object creates the connection to the database. Microsoft Visual Studio .NET provides two types of Connection classes: the SqlConnection object, which is designed specifically to connect to Microsoft SQL Server 7.0 or later, and the OleDbConnection object, which can provide connections to a wide range of database types like Microsoft Access and Oracle. The Connection object contains all of the information required to open a connection to the database.

The Command Object

The Command object is represented by two corresponding classes: SqlCommand and OleDbCommand. Command objects are used to execute commands to a database across a data connection. The Command objects can be used to execute stored procedures on the database, SQL commands, or return complete tables directly. Command objects provide three methods that are used to execute commands on the database:

ExecuteNonQuery: Executes commands that have no return values such as INSERT, UPDATE or DELETE
ExecuteScalar: Returns a single value from a database query
ExecuteReader: Returns a result set by way of a DataReader object

The DataReader Object

The DataReader object provides a forward-only, read-only, connected stream recordset from a database. Unlike other components of the Data Provider, DataReader objects cannot be directly instantiated. Rather, the DataReader is returned as the result of the Command object's ExecuteReader method. The SqlCommand.ExecuteReader method returns a SqlDataReader object, and the OleDbCommand.ExecuteReader method returns an OleDbDataReader object. The DataReader can provide rows of data directly to application logic when you do not

need to keep the data cached in memory. Because only one row is in memory at a time, the DataReader provides the lowest overhead in terms of system performance but requires the exclusive use of an open Connection object for the lifetime of the DataReader.

The DataAdapter Object

The DataAdapter is the class at the core of ADO .NET's disconnected data access. It is essentially the middleman facilitating all communication between the database and a DataSet. The DataAdapter is used either to fill a DataTable or DataSet with data from the database with it's Fill method. After the memory-resident data has been manipulated, the DataAdapter can commit the changes to the database by calling the Update method. The DataAdapter provides four properties that represent database commands:

SelectCommand
InsertCommand
DeleteCommand
UpdateCommand

When the Update method is called, changes in the DataSet are copied back to the database and the appropriate InsertCommand, DeleteCommand, or UpdateCommand is executed.

Use of ADO.NET objects

There are two ways to access & manipulate data of database. First method is visually (with graphical tools) & second method is via coding (using ADO.NET objects directly in coding).

ADO .NET objects are **Connection Object, Command object, DataAdapter object, DataSet objects, DataReader Object, DataTable Object, DataRow Object, DataColumn Object, etc.**

(1) Connection Objects:

The Connection object provides connectivity (physical connection) to a data source (database). Using its method, you can open & close the connection, change the database & manage transactions.

The Connection object is the first component of ADO.NET. A connection sets a link between a data source and ADO.NET. A Connection object sits between a data source and a DataAdapter (via Command).

Connection class exist for **ODBC** (OdbcConnection), **OLE DB** (OledbConnection), **SQL Server** (SqlConnection) & **Oracle** (OracleConnection)

Property :

(1) ConnectionString: It is the string that is used to connect (open) a database when the open method is executed.

ConnectionString property of OledbConnection object has arguments like Provider, Data Source, Database, User ID, Password.

(2) ConnectionTimeout : The maximum time the Connection object attempts to make the connection before throwing an exception. (before terminating the attempt and generating an error). By default is 15 seconds.

(3) DataSource : It is used to specify the server name of the computer on which the database is running. When connecting to an access database, this specifies the path & database name.
(4) State : Gets (returns) the current state of the connection. For example we get Closed, if the connection is closed. & we get Open, if the connection is open.
MsgBox(Con.State.ToString)
(5) ServerVersion : Gets the version of the server.
(6) Provider : This property represents the name of the provider.

Provider parameter specifies the driver that uses to communicate with the database. The most common drivers are **Microsoft.Jet.OLEDB.4.0** for Access, **SQLOLEDB** for SQL server & **MSDAORA** for Oracle.

Method :

(1) Open : Opens a database connection with the property settings specified by the ConnectionString.
(2) Close : Closes the connection to the data source. After the connection is close no transaction can be perform on the database data.

```
Con.Close( )
Con.Dispose( )     'Releases the resources used by the connection object.
Con = Nothing      'Release your reference to the connection object
```

(3) BeginTransaction : Starts (begins) a database transaction.

(2) Command Objects:

The Command object is used to execute SQL statements (Select, Insert, Update & Delete) as well as stored procedure. In addition to the DML statements, you can also execute DDL statements that change the structure of the database.

You can also use the **Parameters** collection in the Command class to pass parameters to stored procedures or SQL statements.

Command object exist for ODBC (OdbcCommand), OLE DB (OledbCommand), SQL Server (SqlCommand) & Oracle (OracleCommand).

Property :

1. **CommandText:** It is the string, contains either SQL statements or name of the stored procedure to be executed.
2. **CommandType:** It represents the type of the **Command** object. Depending upon the command type Command object executes the command. The different command types are as follows.

   StoredProcedure : The name of a stored procedure.
   TableDirect : The name of a table.
   Text : SQL statements. (Default)

For example: Cmd.CommandType = CommandType.**Text**

(3) **Connection:** The name of the active Connection object, through which the command is to be executed.
(4) **Parameters:** The parameters property contains a collection of parameters for the SQL statements or stored procedure.

Methods:

(1) **ExecuteNonQuery** : Executes commands that do not return data rows. But it returns number of rows affected by the commands. (Such as **SQL INSERT**, **DELETE**, **UPDATE**, and **SET** statements).
(2) **ExecuteScalar :** Calculates and returns a single value, such as a sum, min, max from a database. Used for aggregate function.
(3) **ExecuteReader :** Executes SQL commands that return rows. ExecuteReader method is used to create data reader.
(4) **Cancel :** Cancels the execution of the command.

(3) Data Adapter Objects :

The DataAdapter object provides the bridge between the DataSet object and the data source (database) for retrieving and saving data.

The DataAdapter's sole purpose is to retrieves data from the database, then populates (fill) the Datasets & also used to send (propagate) the Datasets changes to the database. (The **DataAdapter** object has **Fill** method to load data from the data source into the dataset, and the **Update** method to send changes you've made in the dataset back to the data source).

The DataAdapter contains four command objects: SelectCommand, InsertCommand, UpdateCommand, and DeleteCommand. The DataAdapter uses the SelectCommand to fill a DataSet & the remaining three commands to transmit changes back to the data source.

Data adapter object exist for ODBC (ODBCDataAdapter), OLE DB (OleDbDataAdapter), SQL Server (SqlDataAdapter) & Oracle (OracleDataAdapter).

Property :

(1) **SelectCommand:** The name of the Command object used to retrieve rows from the data source.
(2) **InsertCommand:** The name of the Command object used to insert rows in the data source.
(3) **UpdateCommand:** The name of the Command object used to update rows in the data source.
(4) **Delete Command:** The name of the Command object used to delete rows in the data source.

Methods :

(1) Fill:- The Fill method which loads data from the data source (database) into the Dataset. If the Connection is closed before Fill is called, it is opened to retrieve data and then closed.

Syntax 1:
OleDbDataAdapter.Fill(DataSet)
Syntax 2:
OleDbDataAdapter.Fill(DataTable)
Syntax 3:

OleDbDataAdapter.Fill(DataSet, TableName)
(2) Update: Update method is used to send changes you've made in the dataset back to the data source (database).

(4) DataSet Objects :

It is the major component of ADO.NET. DataSet is a memory-resident representation of data. **A dataset is a disconnected cache of data, and, that is stored in memory.** DataSet is always disconnected from the data source. It can contain data from multiple sources.

As we have seen in ADO.NET object model, The DataSet composed of two primary objects: the **DataTableCollection**, accessed through Tables property, and **DataRelationCollection** accessed through the Relations property. The DataTableCollection contains zero or more DataTable objects, which are in turn made up of three collections: DataColumnColection, DataRowCollection, and ConstraintCollection. The DataRelationCollection contains zero or more DataRelation objects.

The dataset is a disconnected, in-memory representation of data. An advantage of this is that we do not need to have a continuous connection to the database.

Property :
(1) Relations : It is the collection of DataRelation objects, which defines the relationship of the DataTables within the dataset.
(2) Tables : It is the collection of DataTables contained in the dataset.

Method :
(1) AcceptChanges : Accepts (Commits) all the pending changes made to the dataset.
(2)RejectChanges: Roll back all changes pending in the DateSet. Rolls back the changes made to the dataset since it was created or since the **AcceptChanges** method was called.
(3) Copy : Copies the structure & contents of the DataSet.
(4) Clear : Empties all the tables in the DataSet.
(5) Reset: Returns the DataSet back to its original state.

(5) DataReader Objects:

The DataReader in addition to datasets, there are also datareaders, which are extremely fast, read-only, forward only low-overhead way of retrieving information from the database. You can only move through records with in ascending order means you cannot go backward. ExecuteReader method of a command object is used to create data reader.

DataReader is appropriate when you are processing rows individually and then discarding them. For example, Report generators, you get a performance benefit from **DataReader. DataReader** is best suited for retrieving huge amounts of data, as the data is not cached in the memory.

Property :
(1) FieldCount: Gets the number of columns in the current row.
(2) HasRows: Returns a Boolean value indicating whether the DataReader contains rows of data.
(3) IsClosed: Indicates whether the DataReader is closed.

(4) Item : Gets the value of a column(field). For example If employee table has three fields, EmpNo, EmpName & City, then EmpNo is Item(0), EmpName is Item(1) & City is Item(2).

Method :

(1) Close : Closes the data reader.
(2) GetName : Gets (returns)the name of the specified column.
(3) GetValue : Gets a field's value (column's value) in its native format.
(4) GetValues : Gets all columns in the current row.
(5) IsDBNull : Indicates if a column contains nonexistent (or missing) values.
(6) Read : Read method returns true if there are more rows. It advances the DataReader to the next record.

(6) DataTable Objects :

Datasets are made up of **DataTable** objects. Data in the DataSet is stored in memory in the form of DataTable Objects.

The DataTableCollection contains zero or more DataTable objects, which are in turn made up of three collections: DataColumnColection, DataRowCollection, and ConstraintCollection (used to ensure integrity of data , ForeignKeyConstraint & UniqueConstraint)

(7) DataRow Objects :

DataRow objects represent rows in a **DataTable** object. You use **DataRow** objects to get access to, insert, delete, and update the records in a table.

(8) DataColumn Objects :

DataColumn objects represent the columns, that is, the fields, in a data table.

(9) DataRelation Objects :

The **DataRelation** class supports data relations between data tables. The DataRelationCollection contains zero or more DataRelation objects. It is accessed through the Relations property.

Data Binding :

It is the process of attaching the Data source to a data aware control is called as Data Binding.

.NET supports two types of Data Binding.

(1) Simple Data Binding &
(2) Complex Data Binding.

(1) Simple Data Binding : **In Simple Data Binding, only a single value (one data element) from a DataSet can be bound to control at a time.** In this type of Data Binding, any value from the DataSet can be bound to any property of a control.

Simple Binding at design time :

For example To bind EmpName field of DataSet11, do as follows.

First select the TextBox control, then exapand its **DataBinding** property in the Properties window. Then click on Text property, then click on drop down arrow. The property window display a list of data sources that you can bind to the TextBox.(As shown in figure). For example **DataSet11 - Emp.EmpNo**

Simple Binding in code (at run time)

At run time, we can use a control's **DataBindings** property. It has **Add** method.

```
TextBox1.DataBindings.Add("Text", DataSet11, "Emp.
EmpName")
```

(2) Complex Data Binding : In Complex Data Binding, a control is bound to a complete DataSet (more than one data element) instead of just one column from it.

Most controls support only simple data binding but some controls, such as DataGridView, ComboBox, ListBox & CheckedListBox support complex data binding.

For complex data binding, following properties can be use.

(a) DataSource : It is the name of the data source, for example **DataSet11**.

(b) DataMember : It is the name of the table in a dataset such as the **Employee** table, the DataGridView should display.

(c) DisplayMember : It is the name of the field, that will be display on the control.

(d) ValueMember : It is the name of the field (for example dept no), its corresponding value will be return to the **SelectedValue** property.

Example : To display department names in combo box, we have to set following properties at run time.

```
ComboBox1.DataSource = DataSet11
ComboBox1.DisplayMember = "Dept.DName"
```

Now to get the respective department No, When Department name is selected, you have to set **ValueMember** property to DeptNo.

```
ComboBox1.ValueMember = "Dept.DeptNo"
```

You can access the Department No using the **SelectedValue** property.

```
MsgBox(ComboBox1.SelectedValue)
```