PROGRAMMING WITH VISUAL BASIC

INDEX

MODULE I

MODULE II

6. LOOPING STATEMENT

7. ARRAYS

    7.1 DYNAMIC

8. FUNCTIONS AND PROCEDURES

    8.1 BUILT-IN FUNCTIONS

    8.2 MATHEMATICAL FUNCTIONS

    8.3 STRING FUNCTIONS

9. OBJECT ORIENTED PROGRAMMING

    9.1 CREATING CLASSES

    9.2 OBJECTS

    9.3 PROPERTIES

    9.4 EVENTS

    9.5 CONSTRUCTORS

    9.6 DESTRUCTORS

    9.7 EXCEPTION HANDLING

MODULE III

1. PROPERTIES

2. EVENTS AND METHODS OF FORM

3. LABEL

4. TEXTBOX

5. LISTBOX

6. COMBO BOX,

7. RADIOBUTTON

8. BUTTON

9. CHECK BOX

10. PROGRESS BAR

11. DATE TIME PICKER

12. CALENDAR

13. PICTURE BOX

14. HSCROLLBAR

15. VSCROLLBAR

16. GROUP BOX

17. TOOLTIP

18. TIMER.

MODULE IV

1. MENUS AND TOOLBARS

    1.1 MENU STRIP,

    1.2 TOOL STRIP,

    1.3 STATUS STRIP,

2. BUILT-IN DIALOG BOXES

    2.1 OPEN FILE DIALOGS,

    2.2 SAVE FILE DIALOGS,

    2.3 FONT DIALOGS,

    2.4 COLOR DIALOGS,

    2.5 PRINT DIALOGS,

3. INPUTBOX,

4. MSG BOX,

5. INTERFACING WITH END USER

    5.1 CREATING MDI PARENT AND CHILD.

MODULE V

1. INTRODUCTION TO .NET FRAMEWORK - Visual Programming

.NET FRAMEWORK

The .Net framework is a revolutionary platform that helps you to write the following types of applications

- Windows applications
- Web applications
- Web services

The .Net framework applications are multi-platform applications. The framework has been designed in such a way that it can be used from any of the following languages: Visual Basic, C#, C++, Jscript, and COBOL, etc.

The .Net framework consists of an enormous library of codes used by the client languages like VB.Net. These languages use object-oriented methodology.

Following are some of the components of the .Net framework –

- Common Language Runtime (CLR)
- The .Net Framework Class Library
- Common Language Specification
- Common Type System
- Metadata and Assemblies
- Windows Forms
- ASP.Net and ASP.Net AJAX
- ADO.Net
- Windows Workflow Foundation (WF)
- Windows Presentation Foundation
- Windows Communication Foundation (WCF)
- LINQ

1.1 VISUAL BASIC.NET

VB.Net is a simple, modern, object-oriented computer programming language developed by Microsoft to combine the power of .NET Framework and the common language runtime with the productivity benefits that are the hallmark of Visual Basic. This tutorial will teach you basic VB.Net programming and will also take you through various advanced concepts related to VB.Net programming language.

Visual Basic .NET (VB.NET) is an object-oriented computer programming language implemented on the .NET Framework. Although it is an evolution of classic Visual Basic language, it is not backwards compatible with VB6, and any code written in the old version does not compile under VB.NET.

Like all other .NET languages, VB.NET has complete support for object-oriented concepts. Everything in VB.NET is an object, including all the primitive types (Short, Integer, Long, String, Boolean, etc.) and user-defined types, events, and even assemblies. All objects inherit from the base class Object.

VB.NET is implemented by Microsoft's .NET framework. Therefore, it has full access to all the libraries in the .Net Framework. It's also possible to run VB.NET programs on Mono, the open-source alternative to .NET, not only under Windows, but even Linux or Mac OSX.

The following reasons make VB.Net a widely used professional language –

- Modern, general purpose.
- Object oriented.
- Component oriented.
- Easy to learn.
- Structured language.
- It produces efficient programs.
- It can be compiled on a variety of computer platforms.
- Part of .Net Framework.

1.1.1 FEATURES

- It is an object-oriented programming language that follows various oops concepts such as abstraction, encapsulation, inheritance, and many more.
- It is used to design user interfaces for window, mobile, and web-based applications.
- It supports a rapid application development tool kit in which you can get various codes automatically from its libraries.
- It is not a case-sensitive language like other languages such as C++, java, etc.
- It supports Boolean conditions for decision-making in programming.
- It also supports the multithreading concept, in which you can do multiple tasks at the same time.
- It uses an external object as a reference.

- Automatic initialized a garbage collection.
- It follows a structured and extensible programming language for error detection and recovery.
- It is useful to develop web, window, and mobile applications.

1.1.2 IDE

An integrated development environment (IDE), also known as integrated design environment and integrated debugging environment, is a type of computer software that assists computer programmers to develop software. In the case of Visual Basic .NET, that IDE is Visual Studio.

The Visual Studio IDE consists of several sections, or tools, that the developer uses while programming. As you view the IDE for a new project you generally have three sections in view:

- The Toolbox on the left
- The Solution Explorer on the right
- The Code / Design view in the middle

1.1.2.1 MENU SYSTEM

1.1.2.1.1 FILE MENU:

The File menu contains commands for opening and saving projects or project items, as well as commands for adding new or existing items to the current project.

For the time being, use the New > Project command to create a new project, Open Project/Solution to open an existing project or solution, Save All to save all components of the current project, and the Recent Projects submenu to open one of the recent projects.

1.1.2.1.2 EDIT MENU:

The Edit menu contains the usual editing commands.

The Edit menu is a typical example, It’s quite short when you’re designing the form and quite lengthy when you edit code.

1.1.2.1.3 DATA MENU:

This menu contains commands you will use with projects that access data.

1.1.2.1.4 VIEW MENU:

This menu contains commands to display any toolbar or window of the IDE.

1.1.2.1.5 PROJECT MENU:

This menu contains commands for adding items to the current project (an item can be a form, a file, a component, or even another project).

The last option in this menu is the Project Properties command, which opens the project's Properties Pages. The Add Reference and Add Web Reference commands allow you to add references to .NET components and web components, respectively.

1.1.2.1.6 BUILD MENU:

The Build menu contains commands for building (compiling) your project.

1.1.2.1.7 DEBUG MENU:

This menu contains commands to start or end an application, as well as the basic debugging tools.

1.1.2.1.8 FORMAT MENU:

The Format menu, which is visible only while you design a Windows or web form, contains commands for aligning the controls on the form.

1.1.2.1.9 TOOLS MENU:

This menu contains a list of useful tools, such as the Macros command, which leads to a submenu with commands for creating macros.

1.1.2.1.10 WINDOW MENU:

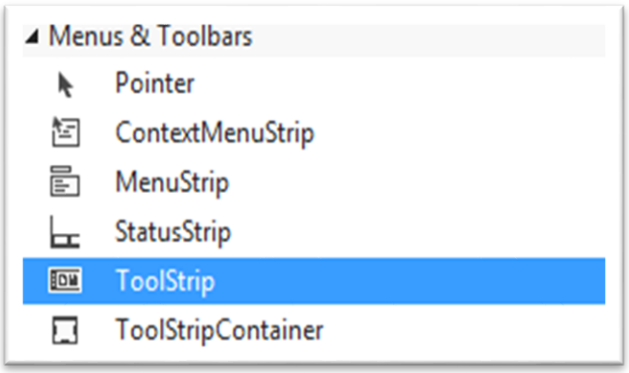
This is the typical Window menu of any Windows application. In addition to the list of open windows, it also contains the Hide command, which hides all toolboxes, leaving the entire window of the IDE devoted to the code editor or the Form Designer.

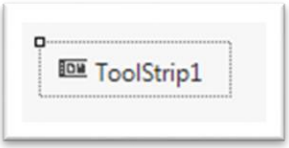
1.1.2.1.11 HELP MENU:

This menu contains the various help options. The Dynamic Help command opens the Dynamic Help window, which is populated with topics that apply to the current operation. The Index command opens the Index window, in which you can enter a topic and get help on the specific topic.

1.1.2.2 TOOLBARS

The Toolbar is a palette of developer objects, or controls, that are placed on forms or web pages, and then code is added to allow the user to interact with them. An example would be TextBox, Button and ListBox controls. With these three controls added to a Windows Form object the developer could write code that would take text, input by the application user, and added to the ListBox after the button was clicked.





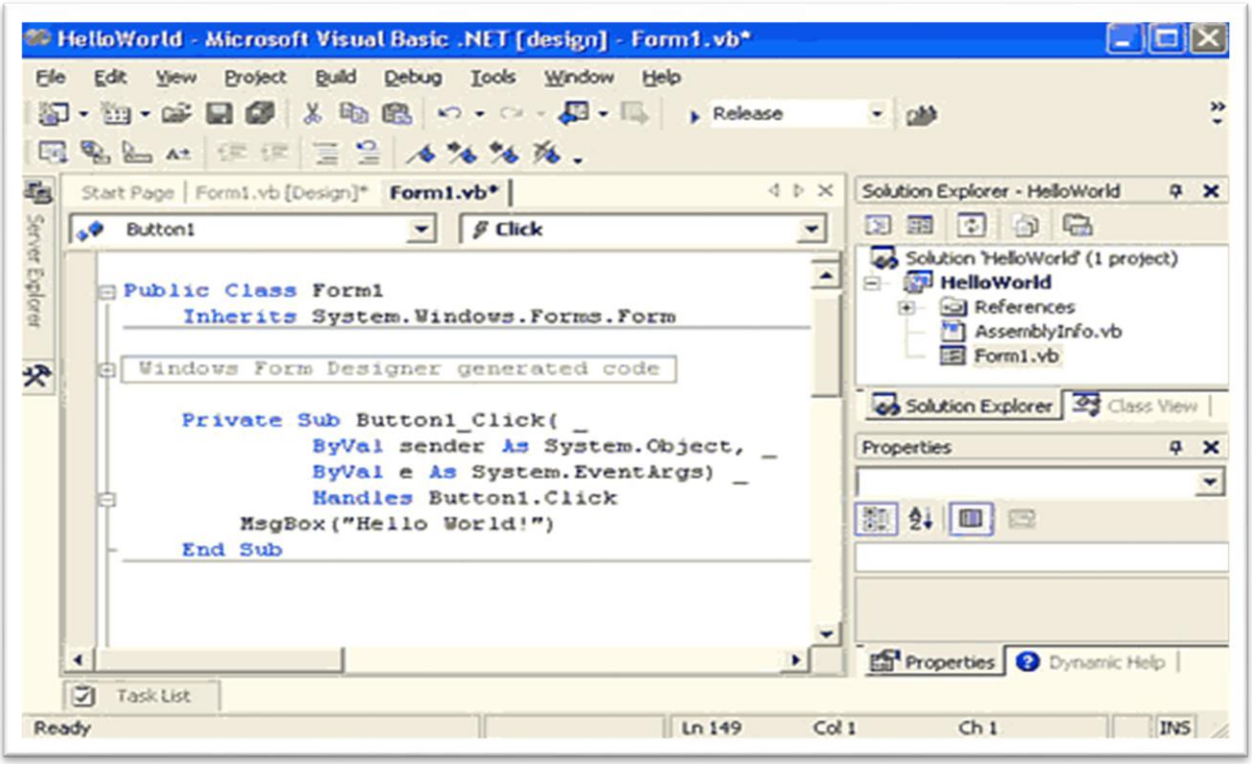

```
Private Sub ToolStripButton1_Click(sender As Object, e As EventArgs) _
                                        Handles ToolStripButton1.Click

    MessageBox.Show("Save button clicked")

End Sub
```

1.1.2.3 CODE DESIGNER

This is where the magic takes place. Forms are designed graphically. In other words, the developer has a form on the screen that can be sized and modified to look the way it will be displayed to the

application users. Controls are added to the form from the Toolbox, the color and caption can be changed along with many other items.

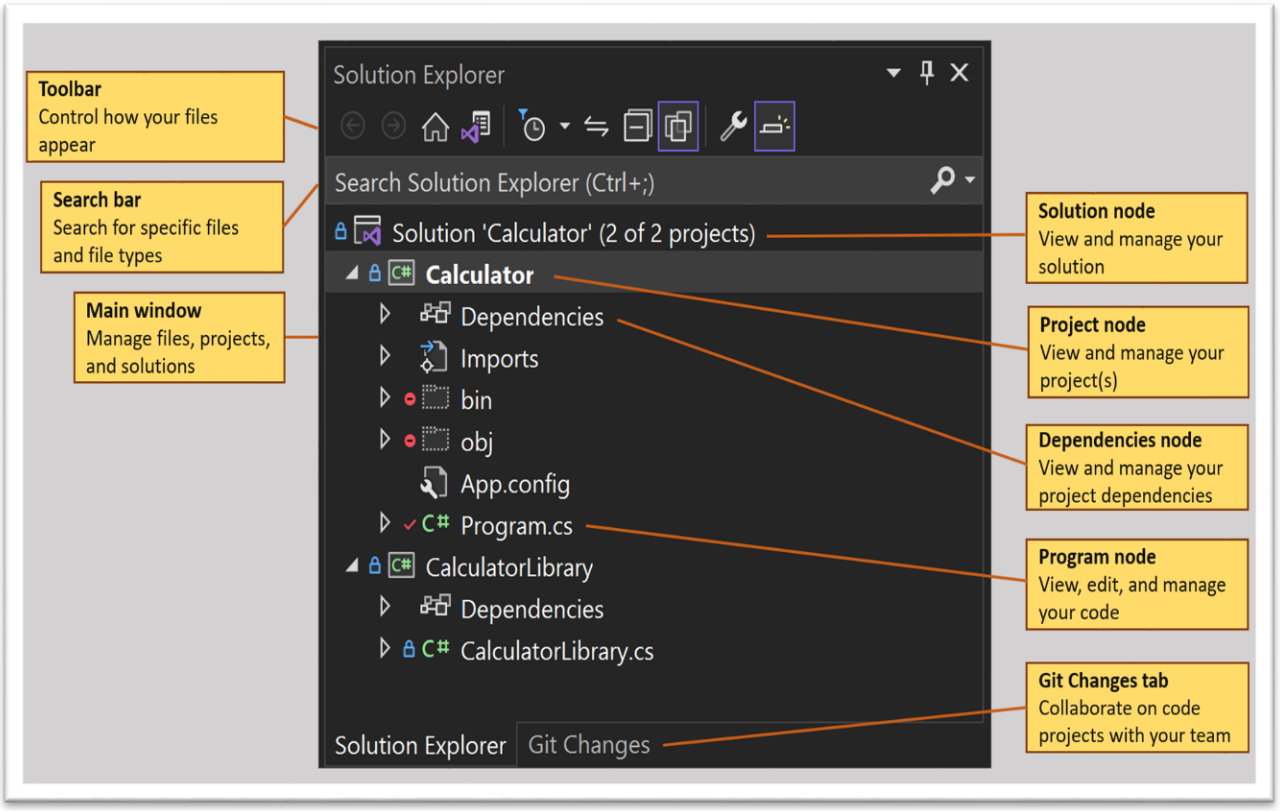
This center window of the IDE is also where developers write the code that makes everything in the application work. The code is written in modules, or files, that are either connected to an object (Forms) or called specifically when needed.



1.1.2.4 SOLUTION EXPLORER

Solution Explorer in Visual Basic.net 2008 lists of all the files associated with a project. It is docked on the right under the Toolbar in the VB IDE. In VB 6 this was known as 'Project Explorer' Solution Explorer has options to view the code, form design, refresh listed files.

Projects files are displayed in a drop down tree like structure, widely used in Windows based GUI applications. This is a section that is used to view and modify the contents of the project.



1.1.2.5 OBJECT BROWSER

The Object Browser allows you to browse through all available objects in your project and see their properties, methods and events. In addition, you can see the procedures and constants that are available from object libraries in your project. Use the Object Browser to find and use objects that you create, as well as objects from other applications.

TO NAVIGATE THE OBJECT BROWSER

Activate a module.

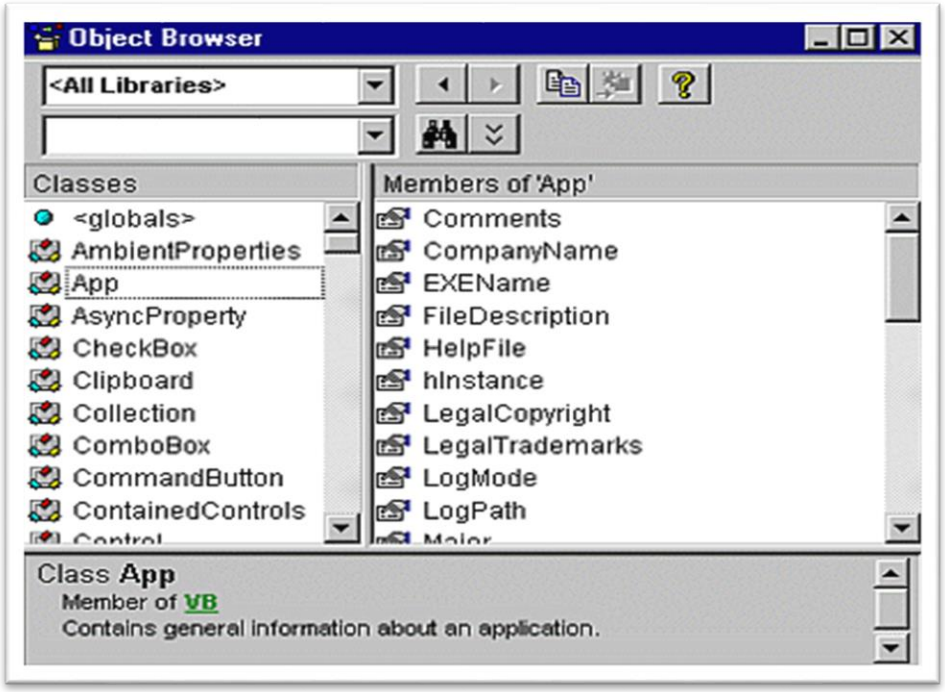
From the View menu, choose Object Browser (F2), or use the toolbar shortcut: Toolbar button.

Select the name of the project or library that you want to view in the Project/Library list.

Use the Class list to select the class; use the Member list to select specific members of your class or project.

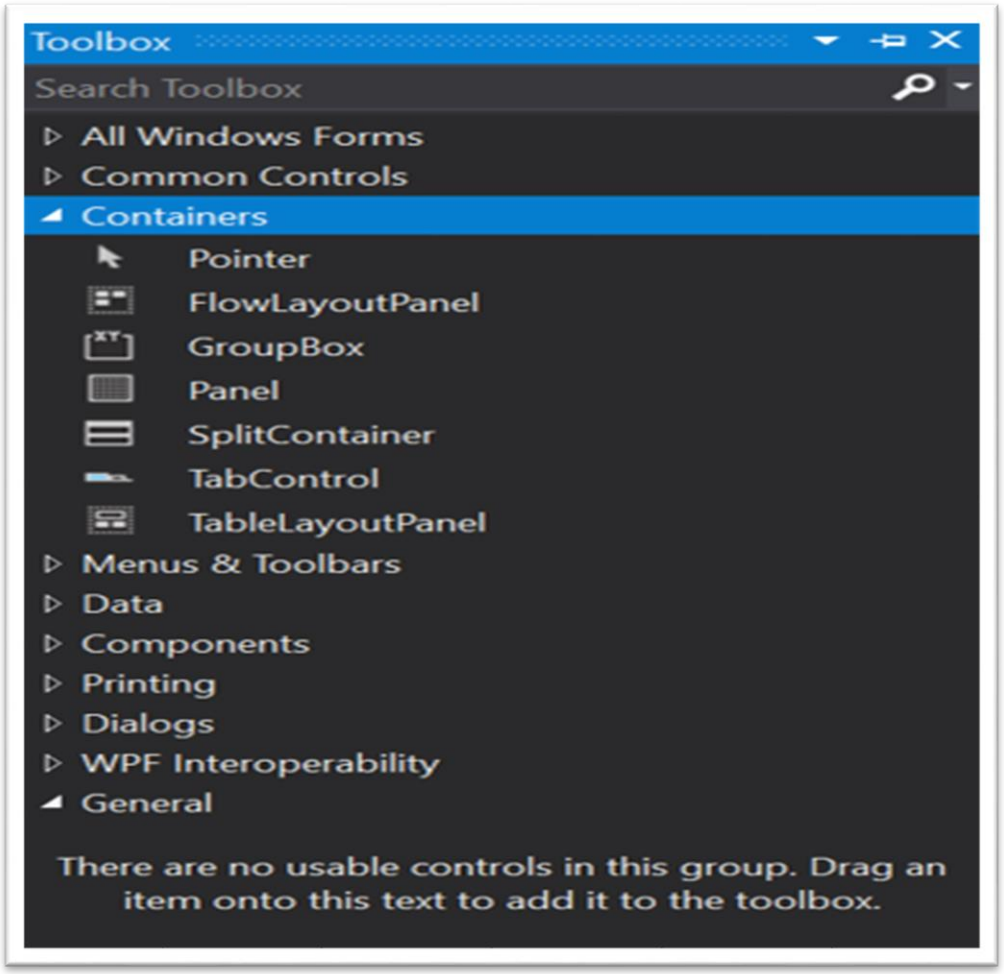
View information about the class or member you selected in the Details section at the bottom of the window.

Use the Help button to display the Help topic for the class or member you selected.



1.1.2.6 TOOLBOX

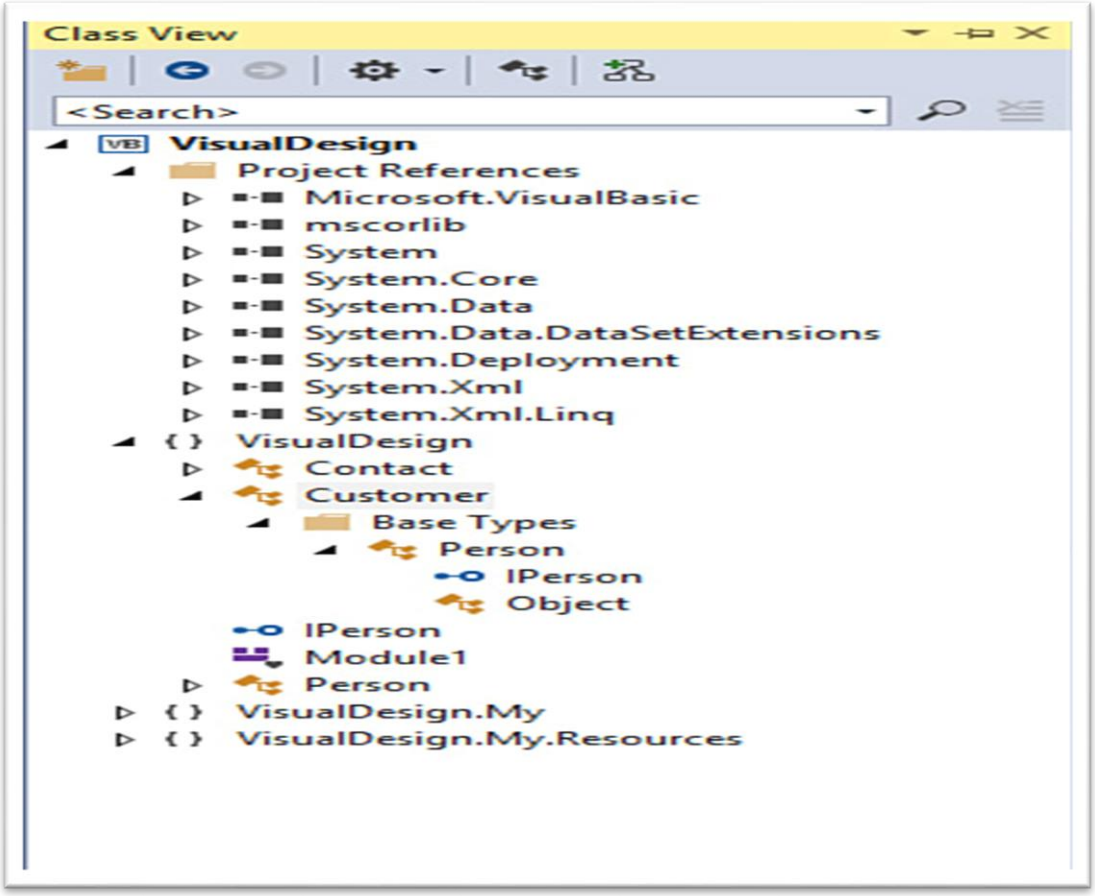
The Toolbox is a palette of developer objects, or controls, that are placed on forms or web pages, and then code is added to allow the user to interact with them. An example would be TextBox, Button and ListBox controls. With these three controls added to a Windows Form object the developer could write code that would take text, input by the application user, and added to the ListBox after the button was clicked.



1.1.2.7 CLASSVIEW WINDOW

We can press Ctrl+Shift+C if it's not already available as a floating window.

It's a browsing tool that shows a graphical representation of the objects' hierarchy in your solution and enables you to search for specific members or get information about types being part of your project, including base types

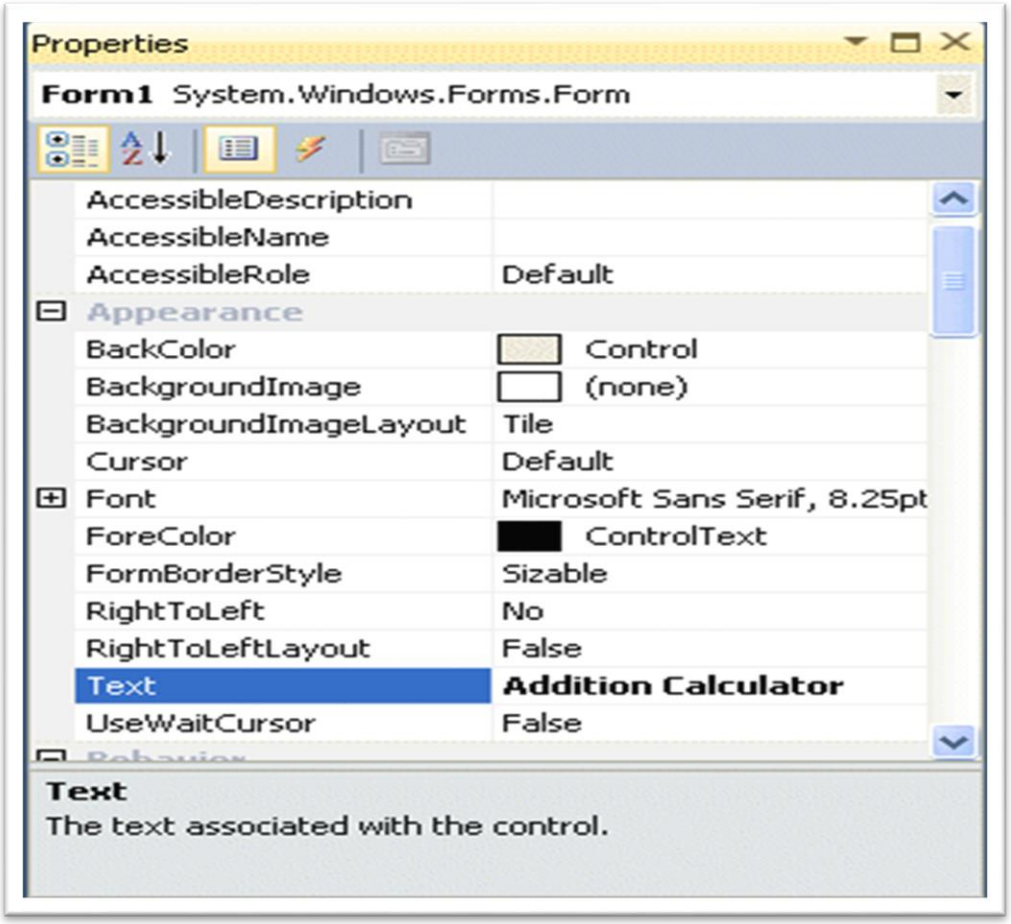


1.1.2.8 PROPERTIES WINDOW

Windows form properties in Visual Basic.net 2008 lists the properties of an selected object. Every object in VB has its own properties that can be used to change the look and even the functionality of the object. Properties Window lists the properties of the forms and controls in an alphabetical order by default.

The properties windows shows all the control (like textbox) properties to be change at design time. Most of this properties can be change at run time with some code, but basically most of this properties change the way the control is display on your application.

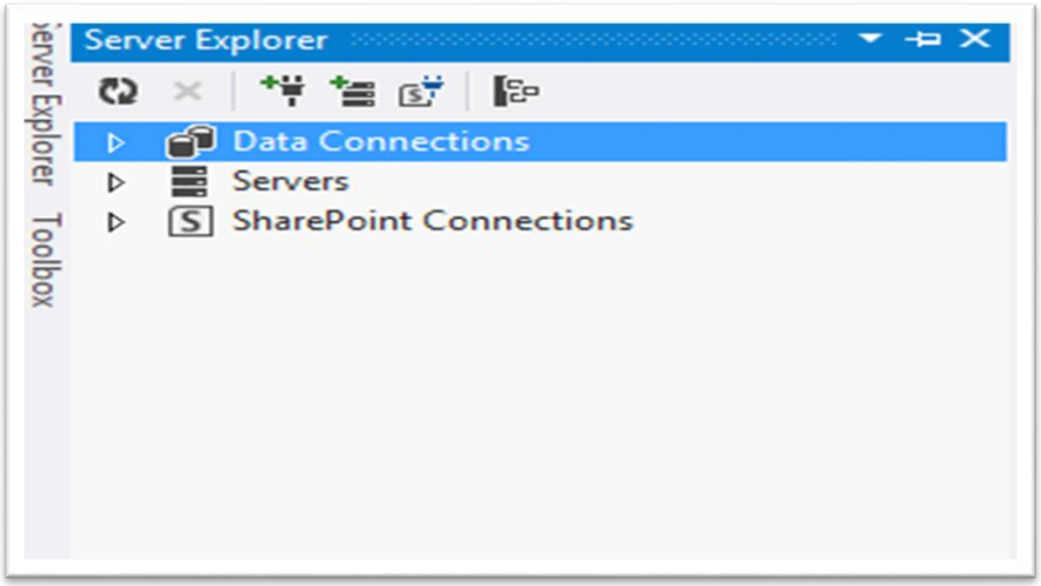
Following is the screen shot of Properties Window.



1.1.2.9 SERVER EXPLORER

The Server Explorer is a shortcut to accessing servers, either installed on the system, or connected to the system. These servers are usually database servers such as SQL Server. By accessing the server, you access all the databases on the specific server, and then you can build the connections needed inside your program, for your program.

The Data Connections item allows you to add, edit, and remove data connections within your program. The Servers item gives you access to all the servers connected to your computer.

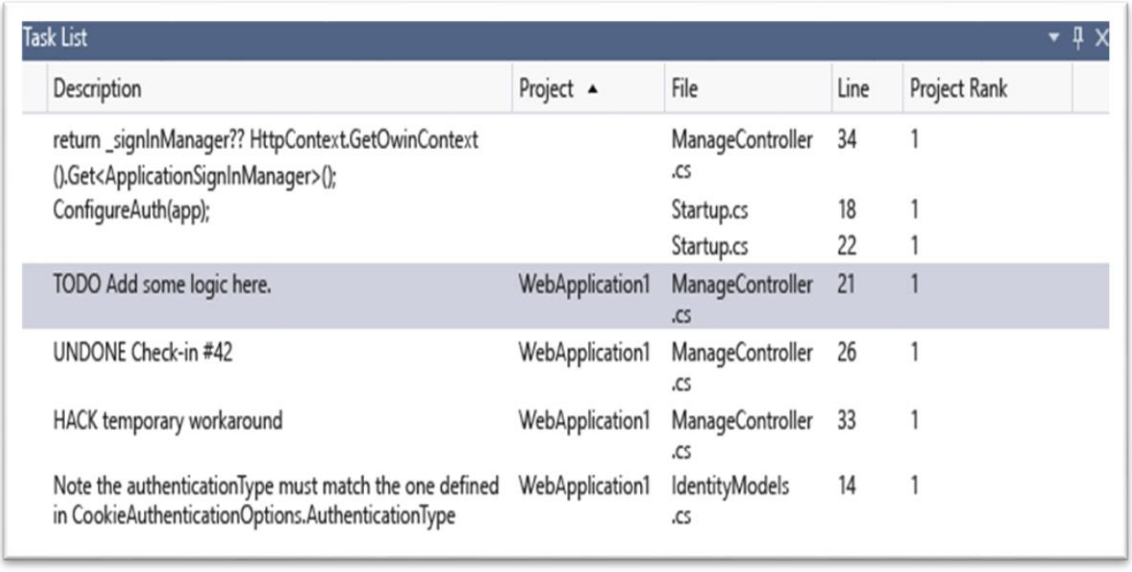


1.1.2.10 TASK LIST

Use Task List to track code comments that use tokens such as TODO and HACK, or custom tokens, and to manage shortcuts that take you directly to a predefined location in code. Select an item in the list to go to its location in the source code.

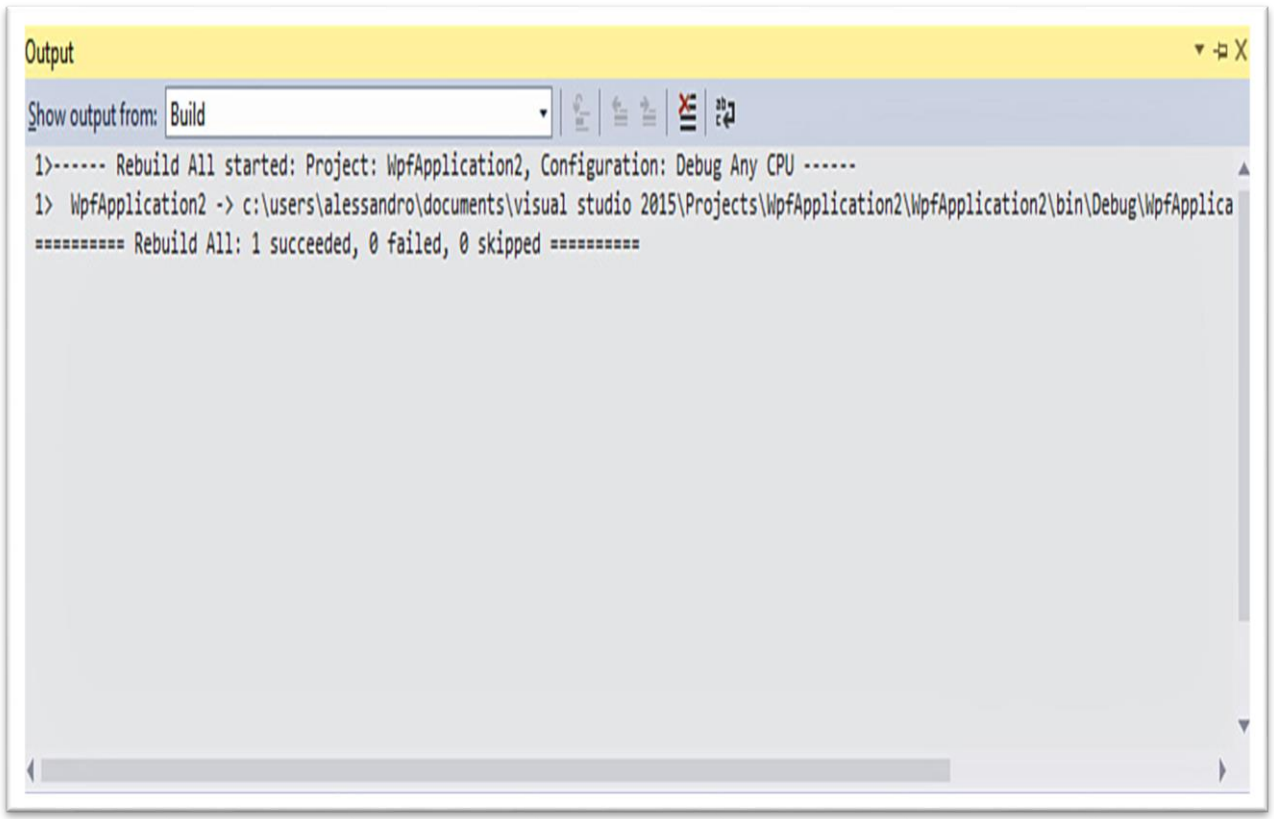
To open Task List, select View > Task List, or from the keyboard press Ctrl+\,T.

1.1.2.11 OUTPUT WINDOW



The Output window displays status messages for various features in the integrated development environment (IDE). To open the Output window, on the menu bar, choose View > Output, or press Ctrl+Alt+O.

To display the Output window whenever you build a project, in the Options dialog box, on the Projects and Solutions > General page, select Show Output window when build starts. Then, with a code file open for editing, choose Go to Next Message and Go To Previous Message on the Output window toolbar to select entries in the Output pane. As you do this, the insertion point in the code editor jumps to the line of code where the selected problem occurs.



1.1.2.12 COMMAND WINDOW

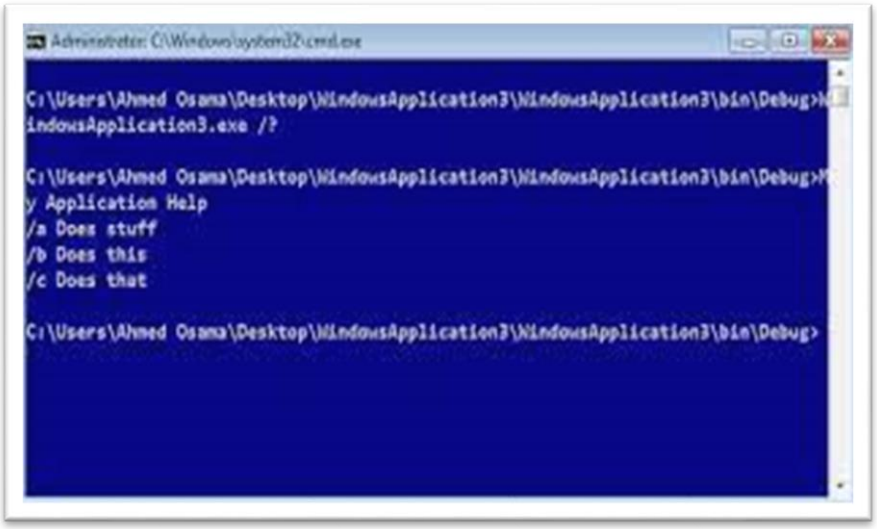
The Command window is used to execute commands or aliases directly in the Visual Studio integrated development environment (IDE). You can execute both menu commands and commands that do not appear on any menu.
To display the Command window, choose Other Windows from the View menu, and select Command Window.

To check the value of variable var a, use the print command:

```
>Debug.Print varA
```

The question mark (?) is an alias for Debug. Print, so this command can also be written:

```
>? varA
```



MODULE II

1. DATA TYPES

In VB.NET, data type is used to define the type of a variable or function in a program. Furthermore, the conversion of one data type to another type using the data conversion function.

A Data Type refers to which type of data or value is assigning to a variable or function so that a variable can hold a defined data type value. For example, when we declare a variable, we have to tell the compiler what type of data or value is allocated to different kinds of variables to hold different amounts of space in computer memory.

Syntax:

```
Dim Variable_Name as DataType
```

Different Data Types and their allocating spaces in VB.NET

DATA TYPES	REQUIRED SPACE	VALUE RANGE
Boolean	A Boolean type depends on the implementing platform	True or False
Byte	1 byte	Byte Range start from 0 to 255 (unsigned)
Char	2 bytes	Char Range start from 0 to 65535 (unsigned)
Date	8 bytes	Date range can be 0:00:0 (midnight) January 1, 0001 to 11:5959 PM of December 31, 9999.
Decimal	16 bytes	Range from 0 to +/- 79,228,162,514,264,337,593,543,950,335 (+/-7.9…E+28) without any decimal point; And 0 to +/- 7.9228162514226433759354395033 5 with 28 position to the right of the decimal
Double	8 bytes	-1.79769313486231570E+308 to -4.94-65645841246544E-324 for negative values; 4.94065645841246544E-324 to

		1.79769313486231570E+308, for positive values
Integer	4 bytes	-2,147,483,648 to 2,147,483,647 (signed)
Long	8 bytes	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 (9.2…E + 18) (signed)
Object	Object size based on the platform such as 4 bytes in 32-bit and 8 bytes in 64-bit platform	It can store any type of data defined in a variable of type Object
Sbyte	1 byte	-128 to 127 (signed)
Short	2 bytes	-32,768 to 32,767 (signed)
Single	4 bytes	-3.4028235E + 38 to -1.401298E-45 for negative values; And for positive value: 1.401298E-45 to 3.4028235E + 38.
String	String Datatype depend on the implementing platform	It accepts Unicode character from 0 to approximately 2 billion characters.
Uinteger	4 bytes	The range start from 0 to 4,294,967,295 (unsigned)
Ulong	8 bytes	The range of ULong start from 0 to 18,446,744,073,709,551,615 (1.8…E + 19) (unsigned)
User-Defined (structure)	A user-defined data type depends on the implementing platform	Each member of the structure has its own data type and limits independent of the other members' ranges.
Ushort	2 bytes	Range from 0 to 65,535 (unsigned)

Example Code:

```
Module Data_Type

    Sub Main()

        Dim B As Byte = 1

        Dim Num As Integer = 5

        Dim Si As Single

        Dim Db As Double

        Dim Get_Date As Date
```

```
        Dim C As Char
        Dim Str As String

        B = 1
        Num = 20
        Si = 0.12
        Db = 2131.787
        Get_Date = Today
        C = "A"
        Str = "Hello Friends..."

        Console.Writeline("Welcome To The Javatpoint")
        Console.Writeline("Byte Is: {0}", B)
        Console.Writeline("Integer Number Is: {0}", Num)
        Console.Writeline("Single Data Type Is: {0}", Si)
        Console.Writeline("Double Data Type Is: {0}", Db)
        Console.Writeline("Today Is: {0}", Get_Date)
        Console.Writeline("Character Is: {0}", B)
        Console.Writeline("String Message Is: {0}", Str)
        Console.Readkey()
    End Sub
End Module
```

Type (Datatype) Conversion Functions in VB.NET

The following functions are available for conversion.

1. CBool(expression): It is used to convert an expression into a Boolean data type.
2. CByte(expression): It is used to convert an expression to a Byte data type.
3. CChar(expression): It is used to convert an expression to a Char data type.
4. CDate(expression): It is used to convert an expression to a Date data type.
5. CDbl(expression): It is used to convert an expression into a Double data type.
6. CDec(expression): It is used to convert an expression into a Decimal data type.
7. CInt(expression): It is used to convert an expression to an Integer data type.
8. CLng(expression): It is used to convert an expression to a Long data type.
9. CObj(expression): It is used to convert an expression to an Object data type.
10. CSByte(expression): It is used to convert an expression to an SByte data type.
11. CShort(expression): It is used to convert an expression to a Short data type.
12. CSng(expression): It is used to convert an expression into a Single data type.
13. CStr(expression): It is used to convert an expression into a String data type.
14. CUInt(expression): It is used to convert an expression to a UInt data type.
15. CULng(expression): It is used to convert an expression to a ULng data type.
16. CUShort(expression): It is used to convert an expression into a UShort data type.

In the following, program we have performed different conversion.

2. KEYWORDS

Keywords are reserved words with special meaning in a programming language, they cannot be used as tokens for such purposes as naming variables and subroutines. Visual Basic is built using keywords. VB.NET keywords are shown in the Table. You can use them to build your programs.

AddHandler	AddressOf	Alias	And	AndAlso	As	Boolean
ByRef	Byte	ByVal	Call	Case	Catch	CBool
CByte	CChar	CDate	CDec	CDbl	Char	CInt

3. DECLARING VARIABLES AND CONSTANTS

A **VARIABLE** is a name given to reserve a memory place that our programs can manipulate. Each 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; and the set of operations that can be applied to the variable.

Variable Declaration in VB.Net

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

Syntax

Dim var-name as data type

Ex:

Dim pi as double

Pi = 3.14

EXAMPLE :

```
Public Class Form1
    Dim a, b, c As Integer

    Private Sub Form1_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Load
    End Sub
    Private Sub Button1_Click(ByVal sender As Object, ByVal e
As System.EventArgs) Handles Button1.Click
        a = Val(TextBox1.Text)
        b = Val(TextBox2.Text)
```

```
        c = Val(TextBox1.Text) + Val(TextBox2.Text)
    End Sub

    Private Sub TextBox3_Click(ByVal sender As Object, ByVal e
As System.EventArgs) Handles TextBox3.Click
        TextBox3.Text = Val(TextBox1.Text) +
Val(TextBox2.Text)
    End Sub
    Private Sub Button2_Click(ByVal sender As Object, ByVal e
As System.EventArgs) Handles Button2.Click
        TextBox3.Text = c
    End Sub
End Class
```

The **CONSTANTS** refer to fixed values that the program may not alter during its execution. Constants can be of any of the basic data types like an integer constant, a floating constant, a character constant, or a string literal. In VB.Net, constants are declared using the Const statement.

Syntax

Const Identifier [As TypeName] = Constant Expression

Ex .

Const num As Integer = 100

4. OPERATORS

Operators are symbols that specify operations to be performed on one or two operands. It tells the compiler to perform specific mathematical or logical manipulations. VB.Net is rich in built-in operators and provides following types of commonly used operators.

- Arithmetic Operators
- Relational (Comparison) Operators
- Logical
- Assignment Operators
- Bitwise Operators
- Special Operators

ARITHMETIC OPERATORS

The arithmetic operators perform the standard arithmetic operations on numeric values. Following table shows all the arithmetic operators supported by VB.Net.

Ex: Assume variable A holds 2 and variable B holds 7, then:

Operator	Description	Example
^	Exponentiaton: Raises one operand to the power of another	B^A will give 49
+	Adds two operands	A + B will give 9
-	Subtracts second operand from the first	A - B will give -5
*	Multiplies both operands	A * B will give 14
/	Divides one operand by another and returns a floating point result	B / A will give 3.5
\	Divides one operand by another and returns an integer result	B \ A will give 3
MOD	Modulus Operator and remainder of after an integer division	B MOD A will give 1

RELATIONAL OPERATORS

The relational operators all perform some comparison between two operands and return a Boolean

value indicating whether the operands satisfy the comparison. Following table shows all the comparison operators supported by VB.Net.

Ex: Assume variable A holds 10 and variable B holds 20, then:

Operator	Description	Example
=	Checks if the values of two operands are equal or not; if yes, then condition becomes true.	(A = B) is not true.
<>	Checks if the values of two operands are equal or not; if values are not equal, then condition becomes true.	(A <> B) is true.

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

LOGICAL OPERATORS

Logical operators take Boolean (logical) operands and return the Boolean result of their operations. Logical operators are:

Operator	Description	Example
And	It is the logical as well as bitwise AND operator. If both the operands are true, then condition becomes true. This operator does not perform short-circuiting, i.e., it evaluates both the expressions.	(A And B) is False.
Or	It is the logical as well as bitwise OR operator. If any of the two operands is true, then condition becomes true. This operator does not perform short-circuiting, i.e., it evaluates both the expressions.	(A Or B) is True.
Not	It is the logical as well as bitwise NOT operator. Use to reverses the logical state of its operand. If a condition is true, then Logical NOT operator will make false.	Not(A And B) is True.
Xor	It is the logical as well as bitwise Logical Exclusive OR operator. It returns True if both expressions are True or both expressions are False; otherwise it returns False. This operator does not perform short-circuiting, it always evaluates both expressions and there is no short-circuiting counterpart of this operator.	A Xor B is True.
AndAlso	It is the logical AND operator. It works only on Boolean data. It performs short-circuiting.	(A AndAlso B) is False.
OrElse	It is the logical OR operator. It works only on Boolean data. It performs short-circuiting.	(A OrElse B) is True.
IsFalse	It determines whether an expression is False.	

IsTrue	It determines whether an expression is True.	

ASSIGNMENT OPERATORS

```
These operators assign values to variables, which are the left operands of
an assignment expression. The

assignment operators come in two forms, simple and compound. The simple
assignment uses the equal sign to assign the operand on the right side to
the operand on the left side.

Ex:   Dim x As Integer = 5;

X = 5       //Simple

x += 1            //Compound
```

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B will assign value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assigns the result to left operand	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assigns the result to left operand	C -= A is equivalent to C = C – A
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assigns the result to left operand	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and assigns the result to left operand (floating point division)	C /= A is equivalent to C = C / A
\=	Divide AND assignment operator, It divides left operand with the right operand and assigns the result to left operand (Integer division)	C \= A is equivalent to C = C \A
^=	Exponentiation and assignment operator. It raises the left operand to the power of the right operand and assigns the result to left operand.	C^=A is equivalent to C = C ^ A
<<=	Left shift AND assignment operator	C <<= 2 is same as C = C << 2

>>=	Right shift AND assignment operator	C >>= 2 is same as C = C >> 2
&=	Concatenates a String expression to a String variable or property and assigns the result to the variable or property.	Str1 &= Str2 is same as Str1 = Str1 & Str2

BITWISE OPERATORS

When your operands are numbers instead of Boolean values, And, Or, and Xor perform bitwise operations on the operands instead of logical ones. Instead of returning True or False, they return either a 1 or a 0 depending on the outcome.

Syntax: expression1 And expression2, the operator returns 1 if both operands are 1; but it returns 0 if one of the operands is 0

The statement expression1 Or expression2 returns 1 even if only one of the operands is 1.

Bitwise Operators and the Values They Return

- And (bit And) Returns 1 if both operands are 1otherwise 0. Valid types areByte, Short, Integer, Long, or enumerated types.
- Or (bit Or) Returns 1 if either is 1otherwise 0. Valid types are Byte, Short, Integer, Long, or enumerated types.
- Xor (bit Xor) Returns 1 if either operand is 1, 0 if both are 1, and 0 if both are 0. Valid types are Byte, Short, Integer, Long, or enumerated types.
- Not (bit Not) If the bit of an operand = 0, then bit in result = 1. If the bit of an operand = 1, then bit in result = 0. ~ (complement)

Specialized Operators: The .NET Framework defines a number of specialized operators.

Unary Operators : There are three unary operators supported by Visual Basic:

Unary Plus is also the additive operator.

Ex:

```
Dim S As String = "what is "

Debug.WriteLine(S + "this")
```

Unary Minus converts a positive number into a negative one. For instance, this simple math

Ex:

```
x = 3 + –1

Debug.WriteLine(x)
```

Unary Logical Not Performs logical negation on its operand. This operator also performs bitwise operations on Byte, Short, Integer, and Long [and all enumerated types].

· If the Expression is False, then the Result is True. · If the Expression is True, then the Result is False

CONCATENATION OPERATOR

The concatenation operator, represented by the ampersand (& or +), combines two string operands and returns a single string expression. The usage is Value = operand & operand

Ex:

```
Dim X As String = "1"

Dim Y As String = "23"

Debug.WriteLine (X & Y)
```

Is - True if two object references refer to the same object

Like - Performs string pattern matching

Address Of - Gets the address of a procedure.

GetType - Gets information about a type.

Operator Precedence: Operator precedence defines the order in which operators are evaluated

Operator	Precedence
Exponentiation (^)	Highest
Unary identity and negation (+, -)	

Multiplication and floating-point division (*, /)	
Integer division (\)	
Modulus arithmetic (Mod)	
Addition and subtraction (+, -)	
Arithmetic bit shift (<<, >>)	
All comparison operators (=, <>, <, <=, >, >=, Is, IsNot, Like, TypeOf...Is)	
Negation (Not)	
Conjunction (And, AndAlso)	
Inclusive disjunction (Or, OrElse)	
Exclusive disjunction (Xor)	Lowest

OPERATOR OVERLOADING

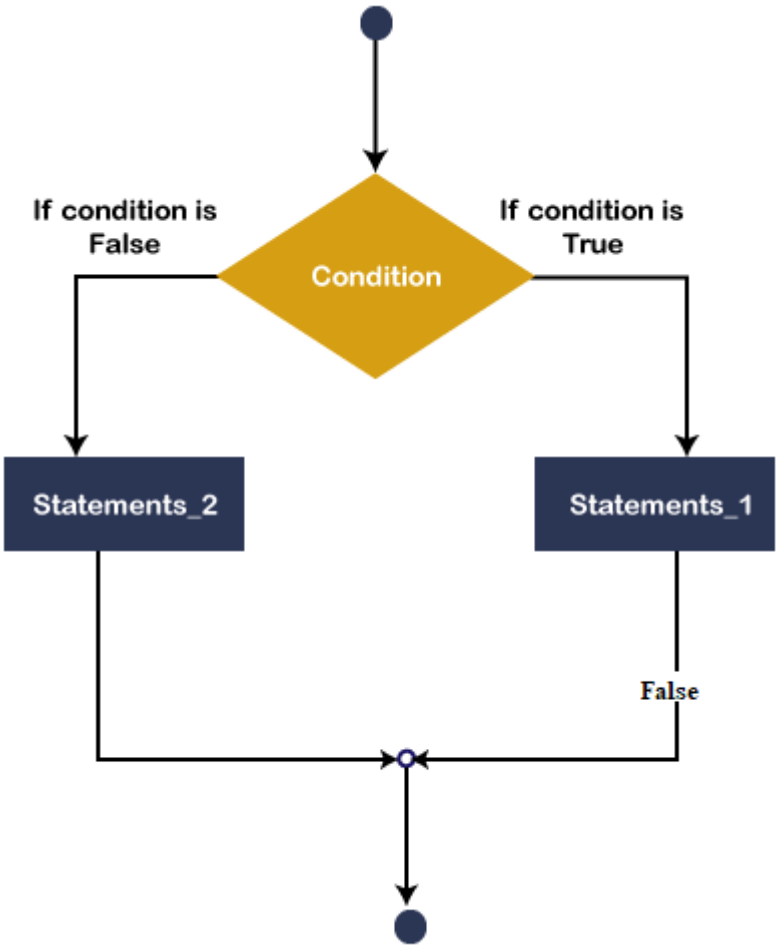
The .NET Framework permits its member languages to implement custom or "user-defined" operators. This is known as operator overloading. Overloading an operator is simply the ability to redefine its operation for a specific type. The built in + operator is used as a unary operator, as the addition (infix) operator, and even as a concatenation operator.

Ex:

```
Dim Str1 As String = "God"

Dim Str2 As String = " bless"

Dim Str3 As String

Str3 = Str1 + Str2

Str3 = Str1 & Str2
```

5.CONDITIONAL STATEMENTS

In **VB.NET**, the **control statements** are the statements that controls the execution of the program on the basis of the specified condition. It is useful for determining whether a condition is true or not. If the condition is true, a single or block of statement is executed. In the control statement, we will use **if- Then, if Then Else, if Then ElseIf** and the **Select case** statement.



- If-Then Statement
- If-Then Else Statement
- If-Then ElseIf Statement
- Select Case Statement
- Nested Select Case Statements

❖ If-Then Statement

The **If-Then** Statement is a control statement that defines one or more conditions, and if the particular condition is satisfied, it executes a piece of information or statements.

Syntax:

```
If condition Then
[Statement or block of Statement]
End If
```

In **If-Then** Statement, the **condition** can be a Boolean, logical, or relational condition, and the statement can be single or group of statements that will be executed when the condition is true.

Example 1: Write a simple program to print a statement in VB.NET.

```
Module Module1
' Declaration of variable str
 Dim str As String = "JavaTpoint"
 Sub Main()
     ' if str equal to "JavaTpoint", below Statement will be executed.
     If str = "JavaTpoint" Then
         Console.WriteLine("Welcome to the JavaTpoint")
     End If
 Console.WritLine("press any key to exit?")
 Console.ReadKey()
 End Sub
End Module
```

❖ If-Then Else Statement

The **If-Then** Statement can execute single or multiple statements when the condition is true, but when the expression evaluates to **false**, it does nothing. So, here comes the **If-Then-Else** Statement. The IF-Then-Else Statement is telling what **If** condition to do when if the statement is false, it executes the Else statement. Following is the If-Then-Else statement syntax in VB.NET as follows:

Syntax:

```
If (Boolean_expression) Then
   Statement
Else
   Statement
End If
```

Example 1: Write a program to check whether the number is even or odd.

```
Module If_Else_statement

    Sub Main()

        Dim num As Integer

        Console.WriteLine("Enter the Number")

        num = Console.ReadLine() 'read data from console

        If (num Mod 2 = 0) Then ' if condition is true, print the if statement

            Console.WriteLine("It is an even number")

        Else 'otherwise, Else statement is executed.

            Console.WriteLine("It is an odd number")

        End If

    Console.WriteLine("press any key to exit...")

        Console.ReadKey()

    End Sub

End Module
```

❖ If-Then ElseIf Statement

The **If-Then-ElseIf** Statement provides a choice to execute only one condition or statement from multiple statements. Execution starts from the top to bottom, and it checked for each If condition. And if the condition is met, the block of If the statement is executed. And **if none** of the conditions are true, the last **block** is executed.

Syntax

```
If(condition 1)Then
 ' Executes when condition 1 is true
  ElseIf( condition 2)Then
    ' Executes when condition 2 is true
  ElseIf( boolean_expression 3)Then
    ' Executes when the condition 3 is true
 Else
End If
```

Example 1: Write a program to show the uses of If... ElseIf statements.

```
Module if_elseIf
    Sub Main()
        Dim var1 As Integer
          Console.WriteLine(" Input the value of var1: ")
        var1 = Console.ReadLine()
        If var1 = 20 Then
            'if condition is true then print the following statement'
            Console.WriteLine(" Entered value is equal to 20")
        ElseIf var1 < 50 Then
            Console.WriteLine(" Entered value is less than 50")

         ElseIf var1 >= 100 Then
            Console.WriteLine(" Entered value is greater than 100")
        Else
            'if none of the above condition is satisfied, print the fol
lowing statement
       Console.WriteLine(" Value is not matched with above condition")
```

```
        End If
        Console.WriteLine(" You have entered : {0}", var1)
        Console.WriteLine(" press any key to exit...")
        Console.ReadKey()
    End Sub
End Module
```

❖ Select Case Statement

In VB.NET, the Select Case statement is a collection of multiple case statements, which allows executing a single case statement from the list of statements. A selected case statement uses a variable to test for equality against multiple cases or statements in a program. If the variable is matched with any test cases, that statement will be executed. And if the condition is not matched with any cases, it executes the default statement.

Using the select case statement in VB.NET programming, you can replace the uses of multiple If-Then-Else If statement from the program for better readability and easy to use.

Syntax

```
Select Case [variable or expression]
        Case value1 'defines the item or value that you want to match.
                // Define a statement to execute
        Case value2 'defines the item or value that you want to match.
                // Define a statement to execute


        Case Else
                // Define the default statement
        End Select
```

1. Select Case Variable / expression
2. Case value1
3. Statement1
4. Case value2, value3
5. Statement2
6. Case Else
7. // define the default statement if none of the condition is true

8. End Select

Example 1: Write a program to display the Days name using the select case statement in VB.NET

```
Imports System
Module Select_case
   Sub Main()
      'define a local variable.
      Dim Days As String
      Days = "Thurs"
      Select Case Days
         Case "Mon"
            Console.WriteLine(" Today is Monday")
         Case "Tue"
            Console.WriteLine(" Today is Tuesday")
         Case "Wed"
            Console.WriteLine("Today is Wednesday")
         Case "Thurs"
            Console.WriteLine("Today is Thursday")
         Case "Fri"
            Console.WriteLine("Today is Friday")
         Case "Sat"
            Console.WriteLine("Today is Saturday")
         Case "Sun"
            Console.WriteLine("Today is Sunday")
         Case Else
            Console.WriteLine(" You have typed Something wrong")

      End Select
      Console.WriteLine("You have selected : {0}", Days)
      Console.WriteLine("Press any key to exit...")
      Console.ReadLine()
   End Sub
End Module
```

6. LOOPING STATEMENT

A **Loop** is used to repeat the same process multiple times until it meets the specified condition in a program. By using a loop in a program, a programmer can repeat any number of statements up to the desired number of repetitions. A loop also provides the suitability to a programmer to repeat the statement in a program according to the requirement. A loop is also used to reduce the program **complexity, easy to understand**, and easy to **debug**.

Advantages of VB.NET loop

- ❖ It provides code iteration functionality in a program.
- ❖ It executes the statement until the specified condition is true.
- ❖ It helps in reducing the size of the code.
- ❖ It reduces compile time.

Types of Loops

- ❖ Do While Loop
- ❖ For Next Loop
- ❖ For Each Loop
- ❖ While End Loop
- ❖ With End Loop

❖ Do While Loop

In VB.NET, Do While loop is used to execute blocks of statements in the program, as long as the condition remains true. It is similar to the While End Loop, but there is slight difference between them. The **while** loop **initially checks** the defined condition, if the condition becomes true, the while loop's statement is executed. Whereas in the **Do** loop, is opposite of the while loop, it means that it executes the Do statements, and then it checks the condition.

Syntax:

```
Do
        [ Statements to be executed]
        Loop While Boolean_expression
// or
Do
        [Statement to be executed]
```

Loop Until Boolean_expression

In the above syntax, the **Do** keyword followed a block of statements, and **While** keyword checks **Boolean_expression** after the execution of the first Do statement.

Example 1. Write a simple program to print a number from 1 to 10 using the Do While loop in VB.NET.

```
Imports System
Module Do_loop
   Sub Main()
      ' Initializatio and Declaration of variable i
      Dim i As Integer = 1
      Do
         ' Executes the following Statement
         Console.WriteLine(" Value of variable I is : {0}", i)
         i = i + 1 'Increment the variable i by 1
      Loop While i <= 10 ' Define the While Condition

      Console.WriteLine(" Press any key to exit...")
      Console.ReadKey()
   End Sub
End Module
```

❖ For Next Loop

A **For Next loop** is used to repeatedly execute a sequence of code or a block of code until a given condition is satisfied. A For loop is useful in such a case when we know how many times a block of code has to be executed. In VB.NET, the For loop is also known as For Next Loop.

Syntax

```
For variable_name As [ DataType ] = start To end [ Step step ]
        [ Statements to be executed ]
Next
```

- **For**: It is the keyword that is present at the beginning of the definition.
- **variable_name**: It is a variable name, which is required in the For loop Statement. The value of the variable determines when to exit from the **For-Next loop,** and the value should only be a numeric.
- **[Data Type]**: It represents the Data Type of the **variable_name.**
- **start To end**: The **start** and **end** are the two important parameters representing the initial and final values of the **variable_name.** These parameters are helpful while the execution begins, the initial value of the variable is set by the start. Before the completion of each repetition, the variable's current value is compared with the end value. And if the value of the variable is less than the end value, the execution continues until the variable's current value is greater than the end value. And if the value is exceeded, the loop is terminated.
- **Step**: A step parameter is used to determine by which the **counter** value of a variable is increased or decreased after each iteration in a program. If the counter value is not specified; It uses 1 as the default value.
- **Statements**: A statement can be a single statement or group of statements that execute during the completion of each iteration in a loop.
- **Next**: In VB.NET a **Next** is a keyword that represents the end of the **For loop's**

Example 1. **Write a simple program to print the number from 1 to 10 using the For Next loop.**

```
Imports System
Module Number
   Sub Main()
     ' It is a simple print statement, and 'vbCrLf' is used to jump in the next line.
     Console.Write(" The number starts from 1 to 10 " & vbCrLf)
     ' declare and initialize variable i
     For i As Integer = 1 To 10 Step 1
        ' if the condition is true, the following statement will be executed
```

```
        Console.WriteLine(" Number is {0} ", i)
        ' after completion of each iteration, next will update the variable counter
    Next
    Console.WriteLine(" Press any key to exit... ")
    Console.ReadKey()
  End Sub
End Module
```

❖ FOR EACH LOOP

In the VB.NET, **For Each loop** is used to iterate block of statements in an array or collection objects. Using For Each loop, we can easily work with collection objects such as lists, arrays, etc., to execute each element of an array or in a collection. And when iteration through each element in the array or collection is complete, the control transferred to the next statement to end the loop.

Syntax:

```
For Each var_name As [ DataType ] In Collection_Object
[ Statements to be executed]
Next
```

For Each loop is used to read each element from the collection object or an array. The **Data Type** represents the type of the variable, and **var_name** is the name of the variable to access elements from the **array** or **collection object** so that it can be used in the body of For Each loop.

Eg: Write a simple program to understand the uses of For Each Next loop in VB.NET.

For_Each_loop.vb

```
Imports System

Module For_Each_loop

    Sub Main()

        'declare and initialize an array as integer

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

```
        Dim i As Integer 'Declare i as Integer
        For Each i In An_array
            Console.WriteLine(" Value of i is {0}", i)
        Next
        Console.WriteLine("Press any key to exit...")
        Console.ReadLine()
    End Sub
End Module
```

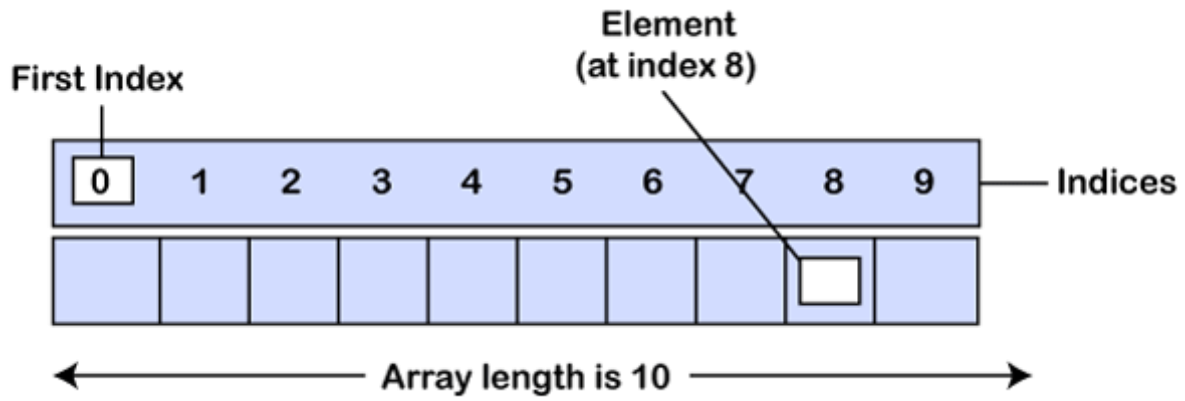
- While End Loop
- With End Loop

7. ARRAYS

An **array** is a linear data structure that is a collection of data elements of the same type stored on a **contiguous memory** location.

Each data item is called an element of the array.

It is a fixed size of sequentially arranged elements in computer memory with the first element being at **index 0** and the last element at index **n 1,** where **n** represents the total number of elements in the array.



Declaration of VB.NET Array

We can declare an array by specifying the data of the elements followed by parentheses () in the VB.NET.

SYNTAX

```
Dim array_name As [Data_Type] ()
```

Array_name is the name of an array
Data_Type represents the type of element (Integer, char, String, Decimal)

Initialization of VB.NET Array

In VB.NET, we can initialize an array with New keyword at the time of declaration.

FOR EXAMPLE:

'Declaration and Initialization of an array elements with size 6

```
Dim num As Integer() = New Integer(5) { }
Dim num As Integer() = New Integer(5) {1, 2, 3, 4, 5, 6}
```

Initialize an array with 5 elements that indicates the size of an array :

```
Dim arr_name As Integer()=New Integer(){5, 10, 5, 20, 15}
```

Declare an array

```
Dim array1 As Char()
array1 = New Char() {'A', 'B', 'C', 'D', 'E'}
```

MULTIDIMENSIONAL ARRAY

Declaration of Multidimensional Array
Declaration of two-dimensional array

```
    Dim twoDimenArray As Integer( , ) = New Integer(3, 2)
 {}
```

Representation of Three Dimensional array

```
    Dim arrThree(2, 4, 3) As Integer
```

7.1 DYNAMIC:

Dynamic arrays are arrays that can be dimensioned and re-dimensioned as per the need of the program. You can declare a dynamic array using the ReDim statement.

Syntax:

```
    ReDim [Preserve] arrayname(subscripts)
```

- The **Preserve** keyword helps to preserve the data in an existing array, when you resize it.
- **arrayname** is the name of the array to re-dimension.
- **subscripts** specifies the new dimension.

Ex:

```
        Dim marks() As Integer

        ReDim marks(2)

        marks(0) = 85

        marks(1) = 75

        marks(2) = 90

'Change the size of an existing array and retain the current values.

        ReDim  Preserve marks(5)

        marks(3) = 80

        marks(4) = 76

        marks(5) = 92
```

8. FUNCTIONS AND PROCEDURES

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.

Syntax:

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

    Statements

End Function
```

- **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 – s**pecifies the data type of the variable the function returns. In VB.Net, a function can return a value to the calling code in two ways:

  By using the return statement

  By assigning the value to the function name

Eg:

```
Function hypotenuse(ByVal side1 As Single, ByVal side2 As Single) As
Single

    Return Math.Sqrt((side1 ^ 2) + (side2 ^ 2))

End Function
```

SUB PROCEDURES

A Sub procedure is a series of Visual Basic statements enclosed by the Sub and End Sub statements. 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.

Defining Sub Procedures

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

Syntax:

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

      Statements

End Sub
```

- **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

Ex:

```
Sub CalculatePay(ByRef hours As Double, ByRef wage As Decimal)

      Dim pay As Doub

      pay = hours * wage

      Console.WriteLine("Total Pay: {0:C}", pay)

End Sub
```

8.1 BUILT-IN FUNCTIONS

Date & Time

The Date data type contains date values, time values, or date and time values. The default value of Date is 00:00:00 . The equivalent .NET data type is System.datetime.

The datetime structure represents an instant in time, typically expressed as a date and time of day.

The dateandtime module contains the procedures and properties used in date and time operations

The following table lists some of the commonly used properties of the DateTime Structure –

S.No	Property	Description
1	**Date**	Gets the date component of this instance.

2	**Dav**	Gets the dav of the month represented bv this instance.
5	**Hour**	Gets the hour component of the date represented bv this instance.
8	**Minute**	Gets the minute component of the date represented bv this instance.
9	**Month**	Gets the month component of the date represented bv this instance.
10	**Now**	Gets a **DateTime** object that is set to the current date and time on this computer, expressed as the local time.
11	**Second**	Gets the seconds component of the date represented bv this
13	**TimeOfDav**	Gets the time of dav for this instance.
14	**Todav**	Gets the current date.

8.2 MATHEMATICAL FUNCTIONS

Math functions are provided by the members of the Math class. All members of the Math class are shared, so it is not necessary to instantiate the class before accessing its members. Members are simply accessed through the class name.

Some of the Math functions:

```
Dim result As Double = Math.Cos(45)
```

The Math class exposes two constants: E

The ratio of the circumference of a circle to its diameter: PI

Computes the absolute value of a number: **Math. Abs (number)**

Returns the larger of two numbers: Max

Returns the smaller of two numbers: Min

Calculates the square root of a number: Sqrt

Raises a given number to a given power: Pow

Rounds a number to either a whole number or a specified decimal place: Round

Ex:

```
Dim num1, num2 As Single

num1 = TextBox1.Text
```

```
num2 = Math.Round(num1, 2)

Label1.Text = num2
```

8.3 STRING FUNCTIONS

In VB.Net, strings as array of characters, however, more common practice is to use the String keyword to declare a string variable. The string keyword is an alias for the System.String class.

Creating a String Object

- You can create string object using one of the following methods:
- By assigning a string literal to a String variable
- By using a String class constructor
- By using the string concatenation operator (+)
- By retrieving a property or calling a method that returns a string
- By calling a formatting method to convert a value or object to its string representation

Ex:

```
Dim str1 as string = {"Hello", "World", "Welcome")
```

Some Properties of the String Class

S.No	Property Name & Description
1	**Chars:** Gets the *Char* object at a specified position in the current *String* object.
2	**Len:** Gets the number of characters in the current String object.

String-handling functions and methods.

Concatenate twostrings : use &, +, String.Concat, String.Join

Compare two strings: use strcomp, String.Compare, String.Equals, String.compareto

Convert strings: use strconv, cstr, String. Tostring

Right ("Phrase", n): The Right function extracts the right portion of a phrase.

Ex:

```
            Dim str1, str2 As String

            str1 = "This is test"

            str2 = "This is text"

            If (String.Compare(str1, str2) = 0) Then

                    Console.WriteLine(str1 + " and " + str2 + " are
equal.")

            Else

                    Console.WriteLine(str1 + " and " + str2 + " are not
equal.")

                 MsgBox(String.Concat(str1, str2))

                 MsgBox(str.Length())

            End If
```

9. OBJECT ORIENTED PROGRAMMING

An object-oriented programming language has three core technologies namely

- **Encapsulation**
- **Inheritance**
- **Polymorphism**

Encapsulation:

Encapsulation refers to the creation of self-contained modules that bind processing functions to the data. These user-defined data types are called classes. Each class contains data as well as a set of methods which manipulate the data. The data components of a class are called instance variables and one instance of a class is an object. For example, in a library system, a class could be a member, and John and Sharon could be two instances (two objects) of the library class.

Inheritance:

The ability to reuse existing objects is a major advantage of object-oriented programming.

Polymorphism:

The ability to take multiple form of existing methods.

An object is a structure that contains data and methods that manipulate the data. Almost everything that you do in Visual Basic is associated with objects. Object-Oriented Programs (OOP) are written based on the Class Objects and not on the functionality of the program. VB.NET is completely object oriented.

9.1 CREATING CLASSES

VB.Net allows users to write programs that break down into modules. These modules represent the real-world objects known as classes or types. A class consists of data members as well as methods. A class can also comprise subclass. For example, the apple tree is a subclass of the plant class and the apple in your backyard is an instance of the apple tree class.

9.2 OBJECTS

A class is a template that defines the form of an object. It specifies both the data and the code that will operate on that data. VB.Net uses a class specification to construct objects. Objects are instances of a class.

Syntax:

creating a class and its object

```
Public Class UserName

            declare variable members

            define methods ……

End Class

Dim object name As New Classname()

Dim obj1 as New Plant()             // object created

Dim obj2 As Plant = New Plant()     // object created
```

9.3 PROPERTIES

Properties are retrieved and set like fields, but are implemented by property Get and property Set procedures, which provide more control on how values are set or returned.

9.4 EVENTS

An event is a message sent by an object announcing that something has happened. In VB.Net most programs are event driven– meaning the flow of execution is determined by external occurrences called events. For example, when a user clicks a control on a form, the form can raise a Click event and call a procedure that handles the event.

Ex:

declare events within classes

```
Class Class1

    Public Event AnEvent(ByVal EventNumber As Integer)

End Class
```

Handle events using WithEvents and the Handles clause

```
Public Sub ClassInst_AnEvent(ByVal EventNumber As Integer) _

      Handles ClassInst.AnEvent

    MsgBox("Received event number: " & CStr(EventNumber))

End Sub
```

Handle events using AddHandler

```
Public Sub EHandler(ByVal EventNumber As Integer)

    MsgBox("Received event number " & CStr(EventNumber))

End Sub
```

9.5 CONSTRUCTORS

In VB.NET, the constructor is a special method that is implemented when an object of a particular class is created. Constructor is also useful for creating and setting default values for a new object of a data member.

When we create a class without defining a constructor, the compiler automatically creates a default constructor. In this way, there is a constructor that is always available in every class. Furthermore, we can create more than one constructor in a class with the use of New keyword but with the different parameters, and it does not return anything.

Default Constructor: In VB.NET, when we create a constructor without defining an argument, it is called a default constructor.

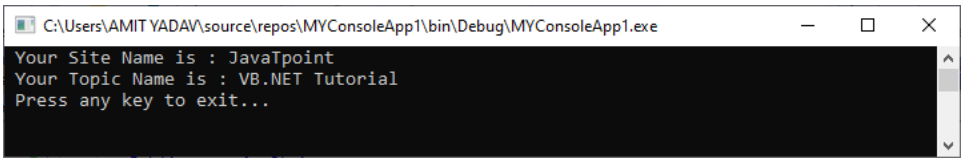
SYNTAX

```
Public Class MyClass

      ' Creates a Constructor using the New

      Public Sub New()

            'Statement to be executed

      End Sub

Public Class
```

Eg: Default Constructor

```
Imports System

Module Default_Const

    Class Tutor

        Public name As String

        Public topic As String

        ' Create a default constructor

        Public Sub New()

            name = "JavaTpoint"

            topic = "VB.NET Tutorial"

        End Sub

    End Class

    Sub Main()

        ' The constructor will automatically call when the instance of the class is created

        Dim tutor As Tutor = New Tutor() ' Create an object as a tutor

        Console.WriteLine(" Your Site Name is : {0}", tutor.name)
```

```
        Console.WriteLine(" Your Topic Name is : {0}", tutor.topic)

        Console.WriteLine(" Press any key to exit...")

        Console.ReadKey()

    End Sub

End Module
```



(B) PARAMETERIZED CONSTRUCTOR

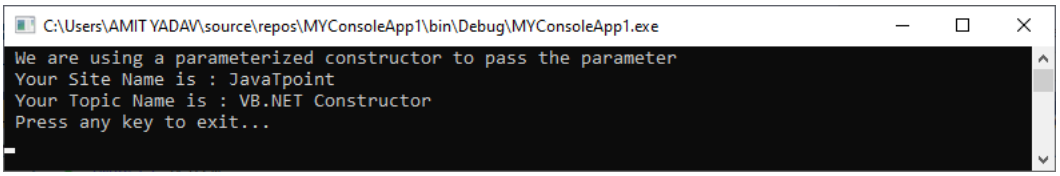
In VB.NET, when we pass one or more arguments to a constructor, the constructor is known as a parameterized constructor. And the object of the class should be initialized with arguments when it is created.

Eg: Parameterized Constructor

```
Imports System

Module Para_Const

    Class Tutor

        Public name As String

        Public topic As String

        ' Create a parameterized constructor to pass parameter

        Public Sub New(ByVal a As String, ByVal b As String)

            name = a

            topic = b

            Console.WriteLine(" We are using a parameterized constructor
to pass the parameter")

        End Sub

    End Class

    Sub Main()
```

```
        ' The constructor will automatically call when the instance of the
class is created

        Dim tutor As Tutor = New Tutor("JavaTpoint", "VB.NET Constructor")

        Console.WriteLine(" Your Site Name is : {0}", tutor.name)

        Console.WriteLine(" Your Topic Name is : {0}", tutor.topic)

        Console.WriteLine(" Press any key to exit...")

        Console.ReadKey()

    End Sub

End Module
```

Output:



9.6 DESTRUCTORS

In VB.NET, Destructor is a special function that is used to destroy a class object when the object of the class goes out of scope. It can be represented as the Finalize() method and does not accept any parameter nor return any value. Furthermore, it can be called automatically when a class object is not needed.

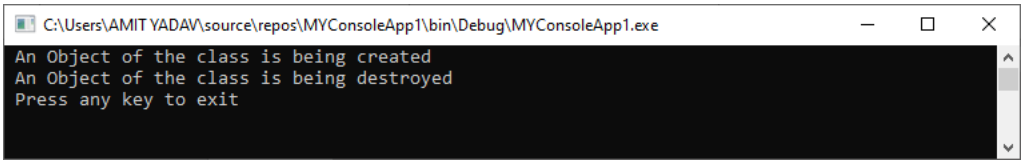
SYNTAX

```
      Class My_Destructor

                ' define the destructor

            Protected Overrides Sub Finalize()

                  ' Statement or code to be executed

            End Sub

      End Class
```

Eg Destructor program

```
Imports System

Module Destruct

    Class Get_Destroy

        Public Sub New()

            Console.WriteLine(" An Object of the class is being created")

        End Sub

        ' Use Finalize() method of Destructor

        Protected Overrides Sub Finalize()

            Console.WriteLine(" An Object of the class is being
destroyed")

            Console.WriteLine(" Press any key to exit")

        End Sub

    End Class

    Sub Main()

        Get_Details() ' After invoking the Get_Details() method, garbage
collector is called automatically

        GC.Collect()

        Console.ReadKey()

        End Sub

    Public Sub Get_Details()

        Dim dest As Get_Destroy = New Get_Destroy()

    End Sub

End Module
```

Output:



CONSTRUCTOR OVERLOADING

In VB.NET, the overloading of constructors is a concept in which we can overload constructors by defining more than one constructor with the same name but with different parameter lists to perform different tasks.

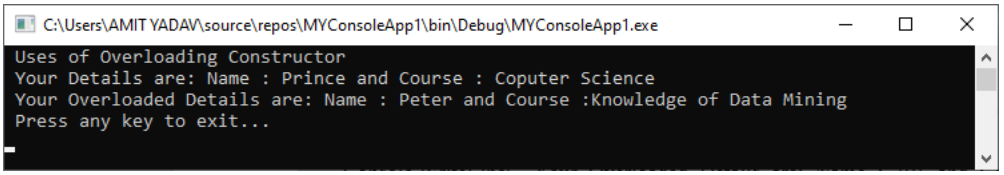
Let's create a program to use the Constructor Overloading in a class.

Const_Over.vb

```
Imports System

Module Const_Over

    Class Details

        Public name As String

        Public course As String

        ' Define Default Constructor

        Public Sub New()

            name = "Prince"

            course = "Coputer Science"

            Console.WriteLine(" Uses of Overloading Constructor")

        End Sub

        ' Create a parametrized constructor

        Public Sub New(ByVal a As String, ByVal b As String)

            name = a
```

```
            course = b

        End Sub

    End Class

    Sub Main()

        ' Called default constructor

        Dim det As Details = New Details()

        ' Called the parametrized constructor

        Dim det1 As Details = New Details("Peter", "Knowledge of Data
Mining")

        Console.WriteLine(" Your Details are: Name : {0} and Course :
{1}", det.name, det.course)

        Console.WriteLine(" Your Overloaded Details are: Name : {0} and
Course :{1}", det1.name, det1.course)

        Console.WriteLine(" Press any key to exit...")

        Console.ReadKey()

    End Sub

End Module
```

Output:



9.7 EXCEPTION HANDLING

An exception is an unwanted error that occurs during the execution of a program and can be a system exception or application exception. Exceptions are nothing but some abnormal and typically an event or condition that arises during the execution, which may interrupt the normal flow of the program.

An exception can occur due to different reasons, including the following:

- A user has entered incorrect data or performs a division operator, such as an attempt to divide by zero.
- A connection has been lost in the middle of communication, or system memory has run out.

When an error occurred during the execution of a program, the exception provides a way to transfer control from one part of the program to another using **exception handling** to handle the error.

VB.NET exception has four built-in keywords such as **Try, Catch, Finally, and Throw** to handle and move controls from one part of the program to another.

Keyword	Description
Try	A try block is used to monitor a particular exception that may throw an exception within the application. And to handle these exceptions, it always follows one or more catch blocks.
Catch	It is a block of code that catches an exception with an exception handler at the place in a program where the problem arises.
Finally	It is used to execute a set of statements in a program, whether an exception has occurred.
Throw	As the name suggests, a throw handler is used to throw an exception after the occurrence of a problem.

Syntax

```
Try
   [ tryStatements ]
   [ Exit Try ]
[ Catch [ exception [ As type ] ] [ When expression ]
   [ catchStatements ]
   [ Exit Try ] ]
[ Catch ... ]
[ Finally
   [ finallyStatements ] ]
End Try
```

Exception Classes in .Net Framework

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

Eg program exception handeling

```
Module exceptionProg
   Sub division(ByVal num1 As Integer, ByVal num2 As Integer)
```

```
      Dim result As Integer
      Try
         result = num1 \ num2
      Catch e As DivideByZeroException
         Console.WriteLine("Exception caught: {0}", e)
      Finally
         Console.WriteLine("Result: {0}", result)
      End Try
   End Sub
   Sub Main()
      division(25, 0)
      Console.ReadKey()
  End Sub
End Module
```

Throwing Objects

You can throw an object if it is either directly or indirectly derived from the System.Exception class.

You can use a throw statement in the catch block to throw the present object as –

```
     Throw [ expression ]
```

The following program demonstrates this –

```
Module exceptionProg
   Sub Main()
      Try
         Throw New ApplicationException("A custom exception _ is
being thrown here...")
      Catch e As Exception
         Console.WriteLine(e.Message)
      Finally
         Console.WriteLine("Now inside the Finally Block")
      End Try
      Console.ReadKey()
   End Sub
End Module
```

MODULE - III

1. PROPERTIES

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.

PROPERTIES OF FORM:

S.N	Properties	Description
1	**AcceptButton**	The button that's automatically activated when you press Enter, no matter which control has the focus at the time. Usually the OK button on a form is set as AcceptButton for a form.
2	**CancelButton**	The button that's automatically activated when you hit the Esc key. Usually, the Cancel button on a form is set as CancelButton for a form.
3	**AutoScale**	This Boolean property determines whether the controls you place on the form are automatically scaled to the height of the current font. The default value of this property is True. This is a property of the form, but it affects the controls on the form.
4	**AutoScroll**	This Boolean property indicates whether scroll bars will be automatically attached to the form if it is resized to a point that not all its controls are visible.
5	**AutoScrollMinSize**	This property lets you specify the minimum size of the form, before the scroll bars are attached.

6	**AutoScrollPosition**	The AutoScrollPosition is the number of pixels by which the two scroll bars were displaced from their initial locations.
7	**BackColor**	Sets the form background color.
8	**BorderStyle**	The BorderStyle property determines the style of the form's border and the appearance of the form − • **None** − Borderless window that can't be resized. • **Sizable** − This is default value and will be used for resizable window that's used for displaying regular forms. • **Fixed3D** − Window with a visible border, "raised" relative to the main area. In this case, windows can't be resized. • **FixedDialog** − A fixed window, used to create dialog boxes. • **FixedSingle** − A fixed window with a single line border. • **FixedToolWindow** − A fixed window with a Close button only. It looks like the toolbar displayed by the drawing and imaging applications. • **SizableToolWindow** − Same as the FixedToolWindow but resizable. In addition, its caption font is smaller than the usual.
9	**ControlBox**	By default, this property is True and you can set it to False to hide the icon and disable the Control menu.
10	**Enabled**	If True, allows the form to respond to mouse and keyboard events; if False, disables form.
11	**Font**	This property specify font type, style, size
12	**HelpButton**	Determines whether a Help button should be displayed in the caption box of the form.

13	**Height**	This is the height of the Form in pixels.
14	**MinimizeBox**	By default, this property is True and you can set it to False to hide the Minimize button on the title bar.
15	**MaximizeBox**	By default, this property is True and you can set it to False to hide the Maximize button on the title bar.
16	**MinimumSize**	This specifies the minimum height and width of the window you can minimize.
17	**MaximumSize**	This specifies the maximum height and width of the window you maximize.
18	**Name**	This is the actual name of the form.
19	**StartPosition**	This property determines the initial position of the form when it's first displayed. It will have any of the following values − • **CenterParent** − The form is centered in the area of its parent form. • **CenterScreen** − The form is centered on the monitor. • **Manual** − The location and size of the form will determine its starting position. • **WindowsDefaultBounds** − The form is positioned at the default location and size determined by Windows. • **WindowsDefaultLocation** − The form is positioned at the Windows default location and has the dimensions you've set at design time.
20	**Text**	The text, which will appear at the title bar of the form.
21	**Top, Left**	These two properties set or return the coordinates of the form's top-left corner in pixels.

22	**TopMost**	This property is a True/False value that lets you specify whether the form will remain on top of all other forms in your application. Its default property is False.
23	**Width**	This is the width of the form in pixel.

2. EVENTS AND METHODS OF FORM

EVENTS:

In VB.NET, events are used to provide a way for objects to communicate with each other. They allow one object to notify other objects when a specific action or condition occurs.

Sr.No.	Event	Description
1	**Activated**	Occurs when the form is activated in code or by the user.
2	**Click**	Occurs when the form is clicked.
3	**Closed**	Occurs before the form is closed.
4	**Closing**	Occurs when the form is closing.
5	**DoubleClick**	Occurs when the form control is double-clicked.
6	**DragDrop**	Occurs when a drag-and-drop operation is completed.
7	**Enter**	Occurs when the form is entered.
8	**GotFocus**	Occurs when the form control receives focus.

9	**HelpButtonClicked**	Occurs when the **Help** button is clicked.
10	**KeyDown**	Occurs when a key is pressed while the form has focus.
11	**KeyPress**	Occurs when a key is pressed while the form has focus.
12	**KeyUp**	Occurs when a key is released while the form has focus.
13	**Load**	Occurs before a form is displayed for the first time.
14	**LostFocus**	Occurs when the form loses focus.
15	**MouseDown**	Occurs when the mouse pointer is over the form and a mouse button is pressed.
16	**MouseEnter**	Occurs when the mouse pointer enters the form.
17	**MouseHover**	Occurs when the mouse pointer rests on the form.
18	**MouseLeave**	Occurs when the mouse pointer leaves the form.
19	**MouseMove**	Occurs when the mouse pointer is moved over the form.
20	**MouseUp**	Occurs when the mouse pointer is over the form and a mouse button is released.
21	**MouseWheel**	Occurs when the mouse wheel moves while the control has focus.
22	**Move**	Occurs when the form is moved.

23	**Resize**	Occurs when the control is resized.
24	**Scroll**	Occurs when the user or code scrolls through the client area.
25	**Shown**	Occurs whenever the form is first displayed.
26	**VisibleChanged**	Occurs when the Visible property value changes.

Methods:

Methods, also known as functions or procedures, are blocks of code that perform a specific task. They can accept parameters and return values.

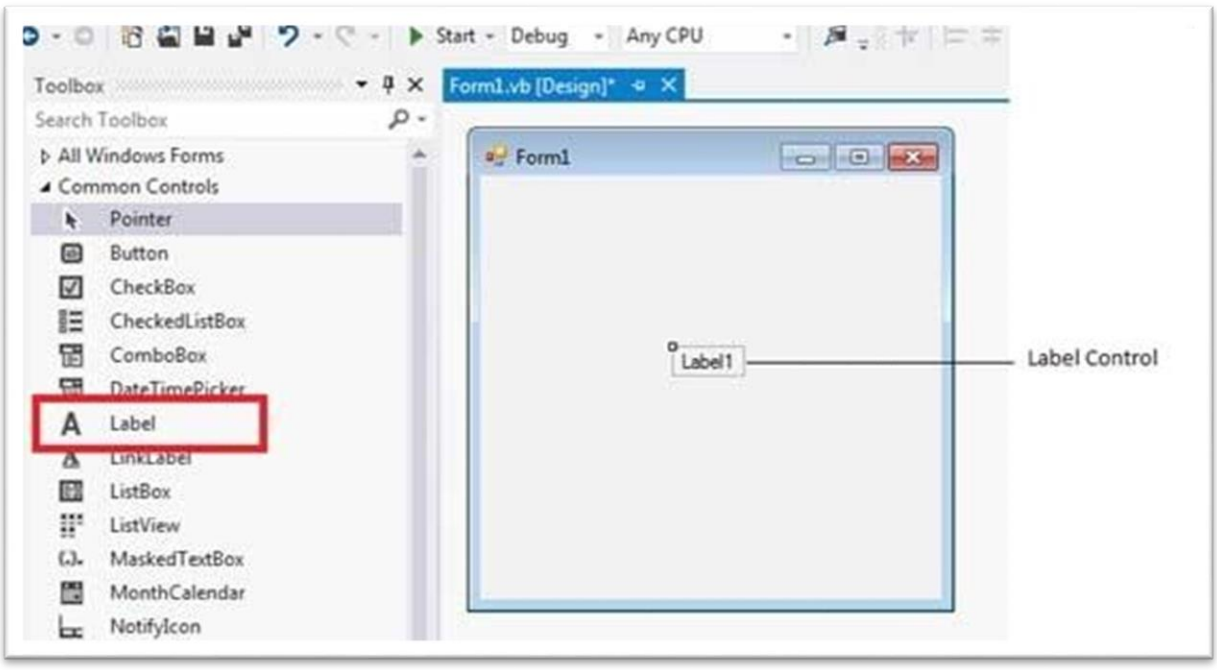
Sr.No.	Method Name & Description
1	**Activate** Activates the form and gives it focus.
2	**ActivateMdiChild** Activates the MDI child of a form.
3	**AddOwnedForm** Adds an owned form to this form.
4	**BringToFront** Brings the control to the front of the z-order.
5	**CenterToParent** Centers the position of the form within the bounds of the parent form.
6	**CenterToScreen**

	Centers the form on the current screen.
7	**Close** Closes the form.
8	**Contains** Retrieves a value indicating whether the specified control is a child of the control.
9	**Focus** Sets input focus to the control.
10	**Hide** Conceals the control from the user.
11	**Refresh** Forces the control to invalidate its client area and immediately redraw itself and any child controls.
12	**Scale(SizeF)** Scales the control and all child controls by the specified scaling factor.
13	**ScaleControl** Scales the location, size, padding, and margin of a control.
14	**ScaleCore** Performs scaling of the form.
15	**Select** Activates the control.
16	**SendToBack**

	Sends the control to the back of the z-order.
17	**SetAutoScrollMargin** Sets the size of the auto-scroll margins.
18	**SetDesktopBounds** Sets the bounds of the form in desktop coordinates.
19	**SetDesktopLocation** Sets the location of the form in desktop coordinates.
20	**SetDisplayRectLocation** Positions the display window to the specified value.
21	**Show** Displays the control to the user.
22	**ShowDialog** Shows the form as a modal dialog box.

3. LABEL

The Label control represents a standard Windows label. It is generally used to display some informative text on the GUI which is not changed during runtime.


Start
Debug
Any CPU
Toolbox
Search Toolbox
All Windows Forms
Common Controls
Pointer
Button
CheckBox
CheckedListBox
ComboBox
DateTimePicker
Label
LinkLabel
ListBox
ListView
MaskedTextBox
MonthCalendar
NotifyIcon
Form1.vb [Design]*
Form1
Label1
Label Control

Properties of the label Control

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

Sr.No.	Property & Description
1	**Autosize** Gets or sets a value specifying if the control should be automatically resized to display all its contents.
2	**BorderStyle** Gets or sets the border style for the control.
3	**FlatStyle** Gets or sets the flat style appearance of the Label control
4	**Font** Gets or sets the font of the text displayed by the control.
5	**FontHeight** Gets or sets the height of the font of the control.
6	**ForeColor** Gets or sets the foreground color of the control.
7	**PreferredHeight** Gets the preferred height of the control.
8	**PreferredWidth** Gets the preferred width of the control.
9	**TabStop** Gets or sets a value indicating whether the user can tab to the Label. This property is not used by this class.

10	**Text** Gets or sets the text associated with this control.
11	**TextAlign** Gets or sets the alignment of text in the label.

Methods of the Label Control

The following are some of the commonly used methods of the Label control –

Sr.No.	Method Name & Description
1	**GetPreferredSize** Retrieves the size of a rectangular area into which a control can be fitted.
2	**Refresh** Forces the control to invalidate its client area and immediately redraw itself and any child controls.
3	**Select** Activates the control.
4	**Show** Displays the control to the user.
5	**ToString** Returns a String that contains the name of the control.

Events of the Label Control

The following are some of the commonly used events of the Label control –

Sr.No.	Event & Description
1	**AutoSizeChanged** Occurs when the value of the AutoSize property changes.
2	**Click** Occurs when the control is clicked.
3	**DoubleClick** Occurs when the control is double-clicked.
4	**GotFocus** Occurs when the control receives focus.
5	**Leave** Occurs when the input focus leaves the control.
6	**LostFocus** Occurs when the control loses focus.
7	**TabIndexChanged** Occurs when the TabIndex property value changes.
8	**TabStopChanged** Occurs when the TabStop property changes.
9	**TextChanged** Occurs when the Text property value changes.

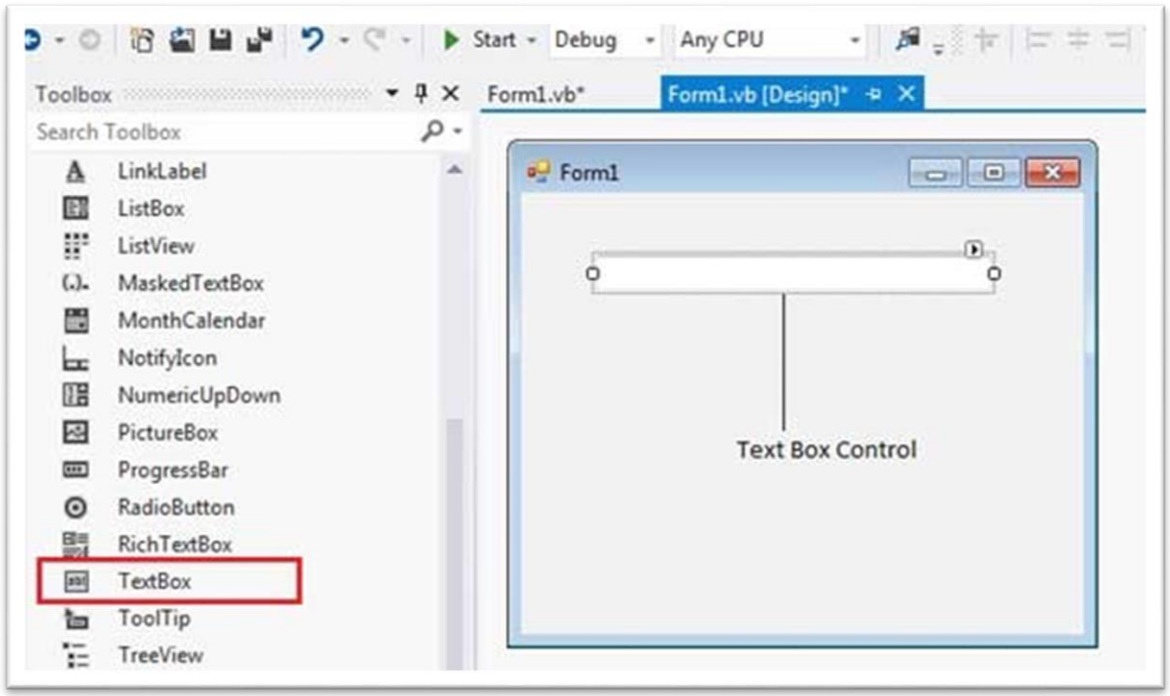
4. TEXTBOX

Text box controls allow entering text on a form at runtime. By default, it takes a single line of text, however, you can make it accept multiple texts and even add scroll bars to it.

Let's create a text box by dragging a Text Box control from the Toolbox and dropping it on the form.

THE PROPERTIES OF THE TEXTBOX CONTROL

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



Sr.No.	Property & Description
1	**AcceptsReturn** Gets or sets a value indicating whether pressing ENTER in a multiline TextBox control creates a new line of text in the control or activates the default button for the form.

2	**AutoCompleteCustomSource** Gets or sets a custom System.Collections.Specialized.StringCollection to use when the AutoCompleteSourceproperty is set to CustomSource.
3	**AutoCompleteMode** Gets or sets an option that controls how automatic completion works for the TextBox.
4	**AutoCompleteSource** Gets or sets a value specifying the source of complete strings used for automatic completion.
5	**CharacterCasing** Gets or sets whether the TextBox control modifies the case of characters as they are typed.
6	**Font** Gets or sets the font of the text displayed by the control.
7	**FontHeight** Gets or sets the height of the font of the control.
8	**ForeColor** Gets or sets the foreground color of the control.
9	**Lines** Gets or sets the lines of text in a text box control.
10	**Multiline** Gets or sets a value indicating whether this is a multiline TextBox control.
11	**PasswordChar**

	Gets or sets the character used to mask characters of a password in a single-line TextBox control.
12	**ReadOnly** Gets or sets a value indicating whether text in the text box is read-only.
13	**ScrollBars** Gets or sets which scroll bars should appear in a multiline TextBox control. This property has values − • None • Horizontal • Vertical • Both
14	**TabIndex** Gets or sets the tab order of the control within its container.
15	**Text** Gets or sets the current text in the TextBox.
16	**TextAlign** Gets or sets how text is aligned in a TextBox control. This property has values − • Left • Right • Center
17	**TextLength** Gets the length of text in the control.
18	**WordWrap** Indicates whether a multiline text box control automatically wraps words to the beginning of the next line when necessary.

THE METHODS OF THE TEXTBOX CONTROL

The following are some of the commonly used methods of the TextBox control –

Sr.No.	Method Name & Description
1	**AppendText** Appends text to the current text of a text box.
2	**Clear** Clears all text from the text box control.
3	**Copy** Copies the current selection in the text box to the **Clipboard**.
4	**Cut** Moves the current selection in the text box to the **Clipboard**.
5	**Paste** Replaces the current selection in the text box with the contents of the **Clipboard**.
6	**Paste(String)** Sets the selected text to the specified text without clearing the undo buffer.
7	**ResetText** Resets the Text property to its default value.
8	**ToString** Returns a string that represents the TextBoxBase control.
9	**Undo**

	Undoes the last edit operation in the text box.

Events of the TextBox Control

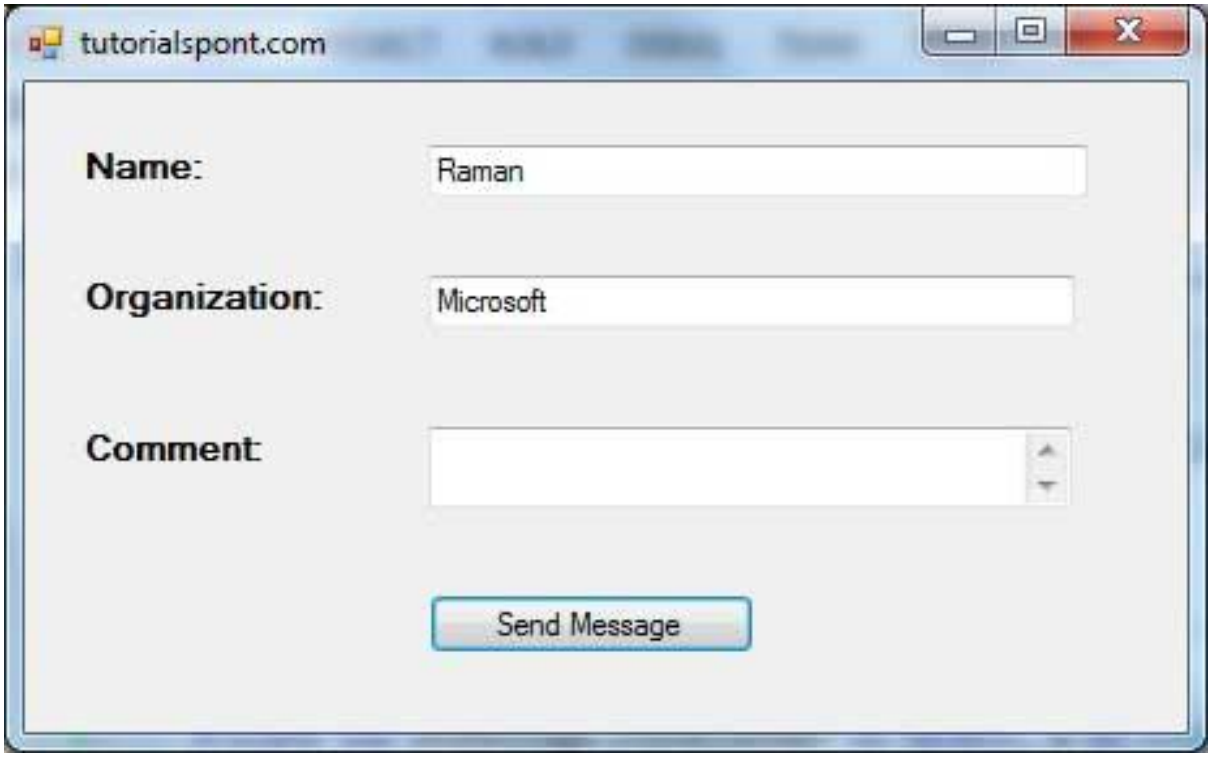
The following are some of the commonly used events of the Text control –

Sr.No.	Event & Description
1	**Click** Occurs when the control is clicked.
2	**DoubleClick** Occurs when the control is double-clicked.
3	**TextAlignChanged** Occurs when the TextAlign property value changes.

Eg Program:

```
Public Class Form1
  Private Sub Form1_Load(sender As Object, e As EventArgs) _
     Handles MyBase.Load
    ' Set the caption bar text of the form.
    Me.Text = "tutorialspont.com"
  End Sub

  Private Sub btnMessage_Click(sender As Object, e As EventArgs) _
  Handles btnMessage.Click
    MessageBox.Show("Thank you " + txtName.Text + " from " + txtOrg.Text)
  End Sub
End Class
```

Output:



5. LISTBOX

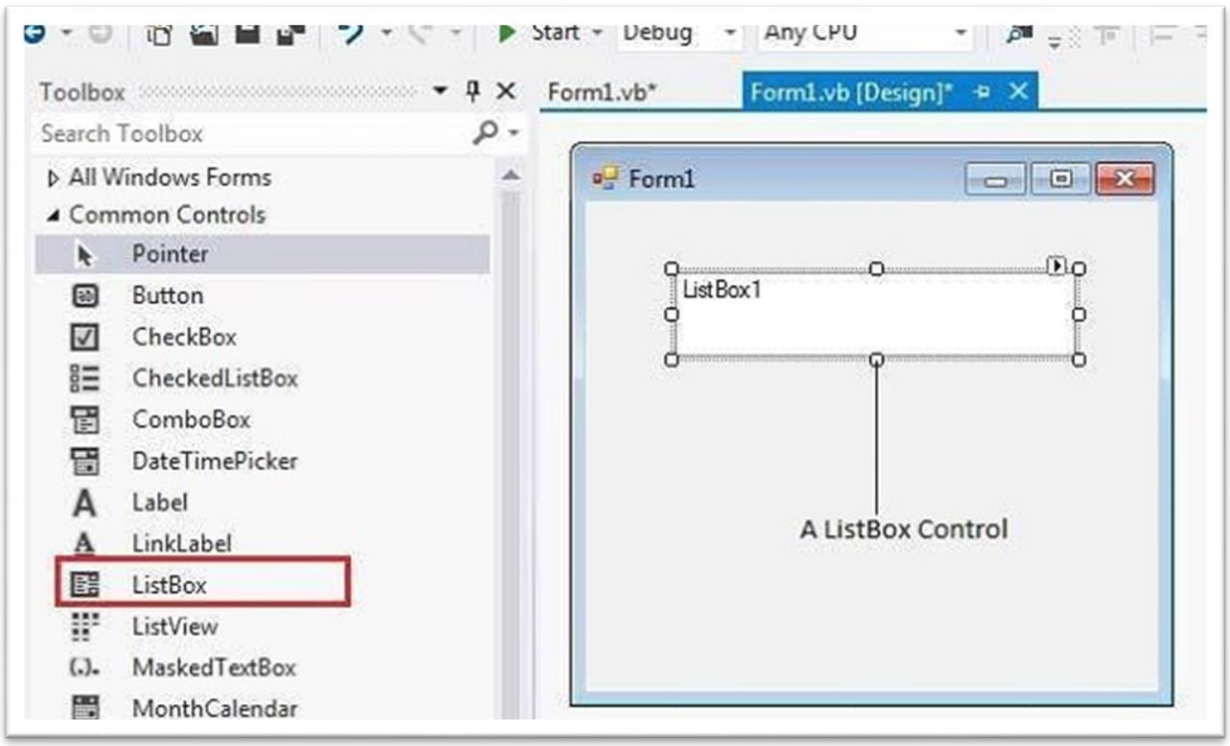
The ListBox represents a Windows control to display a list of items to a user. A user can select an item from the list. It allows the programmer to add items at design time by using the properties window or at the runtime.

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

You can populate the list box items either from the properties window or at runtime. To add items to a ListBox, select the ListBox control and get to the properties window, for the properties of this control. Click the ellipses (...) button next to the Items property. This opens the String Collection Editor dialog box, where you can enter the values one at a line.

Properties of the ListBox Control

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



Sr.No.	Property & Description
1	**AllowSelection** Gets a value indicating whether the ListBox currently enables selection of list items.
2	**BorderStyle** Gets or sets the type of border drawn around the list box.
3	**ColumnWidth** Gets of sets the width of columns in a multicolumn list box.

4	**HorizontalExtent** Gets or sets the horizontal scrolling area of a list box.
5	**HorizontalScrollBar** Gets or sets the value indicating whether a horizontal scrollbar is displayed in the list box.
6	**ItemHeight** Gets or sets the height of an item in the list box.
7	**Items** Gets the items of the list box.
8	**MultiColumn** Gets or sets a value indicating whether the list box supports multiple columns.
9	**ScrollAlwaysVisible** Gets or sets a value indicating whether the vertical scroll bar is shown at all times.
10	**SelectedIndex** Gets or sets the zero-based index of the currently selected item in a list box.
11	**SelectedIndices** Gets a collection that contains the zero-based indexes of all currently selected items in the list box.
12	**SelectedItem** Gets or sets the currently selected item in the list box.
13	**SelectedItems** Gets a collection containing the currently selected items in the list box.

14	**SelectedValue** Gets or sets the value of the member property specified by the ValueMember property.
15	**SelectionMode** Gets or sets the method in which items are selected in the list box. This property has values − • None • One • MultiSimple • MultiExtended
16	**Sorted** Gets or sets a value indicating whether the items in the list box are sorted alphabetically.
17	**Text** Gets or searches for the text of the currently selected item in the list box.
18	**TopIndex** Gets or sets the index of the first visible item of a list box.

Methods of the ListBox Control

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

Sr.No.	Method Name & Description
1	**BeginUpdate** Prevents the control from drawing until the EndUpdate method is called, while items are added to the ListBox one at a time.
2	**ClearSelected** Unselects all items in the ListBox.

3	**EndUpdate** Resumes drawing of a list box after it was turned off by the BeginUpdate method.
4	**FindString** Finds the first item in the ListBox that starts with the string specified as an argument.
5	**FindStringExact** Finds the first item in the ListBox that exactly matches the specified string.
6	**GetSelected** Returns a value indicating whether the specified item is selected.
7	**SetSelected** Selects or clears the selection for the specified item in a ListBox.
8	**OnSelectedIndexChanged** Raises the SelectedIndexChanged event.
8	**OnSelectedValueChanged** Raises the SelectedValueChanged event.

Events of the ListBox Control

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

Sr.No.	Event & Description
1	**Click** Occurs when a list box is selected.
2	**SelectedIndexChanged** Occurs when the SelectedIndex property of a list box is changed.

Eg program

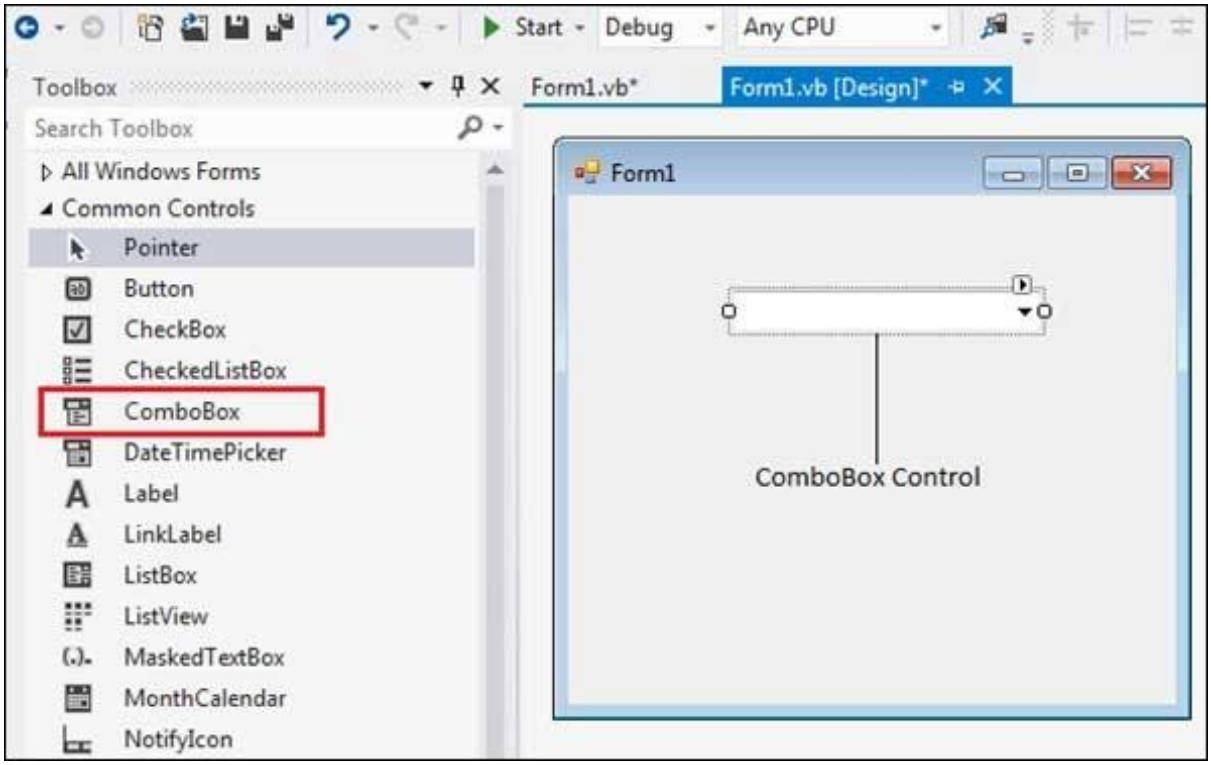
```
Public Class Form1
  Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    ' Set the caption bar text of the form.
    Me.Text = "tutorialspont.com"
    ListBox1.Items.Add("Canada")
    ListBox1.Items.Add("USA")
    ListBox1.Items.Add("UK")
    ListBox1.Items.Add("Japan")
    ListBox1.Items.Add("Russia")
    ListBox1.Items.Add("China")
    ListBox1.Items.Add("India")
  End Sub

  Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
    MsgBox("You have selected " + ListBox1.SelectedItem.ToString())
  End Sub

  Private Sub ListBox1_SelectedIndexChanged(sender As Object, e As EventArgs)
    Handles ListBox1.SelectedIndexChanged
    Label2.Text = ListBox1.SelectedItem.ToString()
  End Sub
End Class
```

OUTPUT:

6. COMBO BOX

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

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



You can populate the list box items either from the properties window or at runtime. To add items to a ComboBox, select the ComboBox control and go to the properties window for the properties of this control. Click the ellipses (...) button next to the Items property. This opens the String Collection Editor dialog box, where you can enter the values one at a line.

Properties of the ComboBox Control

The following are some of the commonly used properties of the ComboBox control –

Sr.No.	Property & Description
1	**AllowSelection** Gets a value indicating whether the list enables selection of list items.
2	**AutoCompleteCustomSource** Gets or sets a custom System.Collections .Specialized.StringCollection to use when the AutoCompleteSourceproperty is set to CustomSource.
3	**AutoCompleteMode**

	Gets or sets an option that controls how automatic completion works for the ComboBox.
4	**AutoCompleteSource** Gets or sets a value specifying the source of complete strings used for automatic completion.
5	**DataBindings** Gets the data bindings for the control.
6	**DataManager** Gets the CurrencyManager associated with this control.
7	**DataSource** Gets or sets the data source for this ComboBox.
8	**DropDownHeight** Gets or sets the height in pixels of the drop-down portion of the ComboBox.
9	**DropDownStyle** Gets or sets a value specifying the style of the combo box.
10	**DropDownWidth** Gets or sets the width of the of the drop-down portion of a combo box.
11	**DroppedDown** Gets or sets a value indicating whether the combo box is displaying its drop-down portion.
12	**FlatStyle** Gets or sets the appearance of the ComboBox.
13	**ItemHeight**

	Gets or sets the height of an item in the combo box.
14	**Items** Gets an object representing the collection of the items contained in this ComboBox.
15	**MaxDropDownItems** Gets or sets the maximum number of items to be displayed in the drop-down part of the combo box.
16	**MaxLength** Gets or sets the maximum number of characters a user can enter in the editable area of the combo box.
17	**SelectedIndex** Gets or sets the index specifying the currently selected item.
18	**SelectedItem** Gets or sets currently selected item in the ComboBox.
19	**SelectedText** Gets or sets the text that is selected in the editable portion of a ComboBox.
20	**SelectedValue** Gets or sets the value of the member property specified by the ValueMember property.
21	**SelectionLength** Gets or sets the number of characters selected in the editable portion of the combo box.
22	**SelectionStart** Gets or sets the starting index of text selected in the combo box.

23	**Sorted** Gets or sets a value indicating whether the items in the combo box are sorted.
24	**Text** Gets or sets the text associated with this control.

Methods of the ComboBox Control

The following are some of the commonly used methods of the ComboBox control –

Sr.No.	Method Name & Description
1	**BeginUpdate** Prevents the control from drawing until the EndUpdate method is called, while items are added to the combo box one at a time.
2	**EndUpdate** Resumes drawing of a combo box, after it was turned off by the BeginUpdate method.
3	**FindString** Finds the first item in the combo box that starts with the string specified as an argument.
4	**FindStringExact** Finds the first item in the combo box that exactly matches the specified string.
5	**SelectAll** Selects all the text in the editable area of the combo box.

Events of the ComboBox Control

The following are some of the commonly used events of the ComboBox control –

Sr.No.	Event & Description
1	**DropDown** Occurs when the drop-down portion of a combo box is displayed.
2	**DropDownClosed** Occurs when the drop-down portion of a combo box is no longer visible.
3	**DropDownStyleChanged** Occurs when the DropDownStyle property of the ComboBox has changed.
4	**SelectedIndexChanged** Occurs when the SelectedIndex property of a ComboBox control has changed.
5	**SelectionChangeCommitted** Occurs when the selected item has changed and the change appears in the combo box.

Eg Program:

```
Public Class Form1
  Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    ' Set the caption bar text of the form.
    Me.Text = "tutorialspont.com"
  End Sub

  'sends the selected items to the list box
  Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
    If ComboBox1.SelectedIndex > -1 Then
      Dim sindex As Integer
      sindex = ComboBox1.SelectedIndex
      Dim sitem As Object
      sitem = ComboBox1.SelectedItem
      ListBox1.Items.Add(sitem)
    End If
  End Sub
```

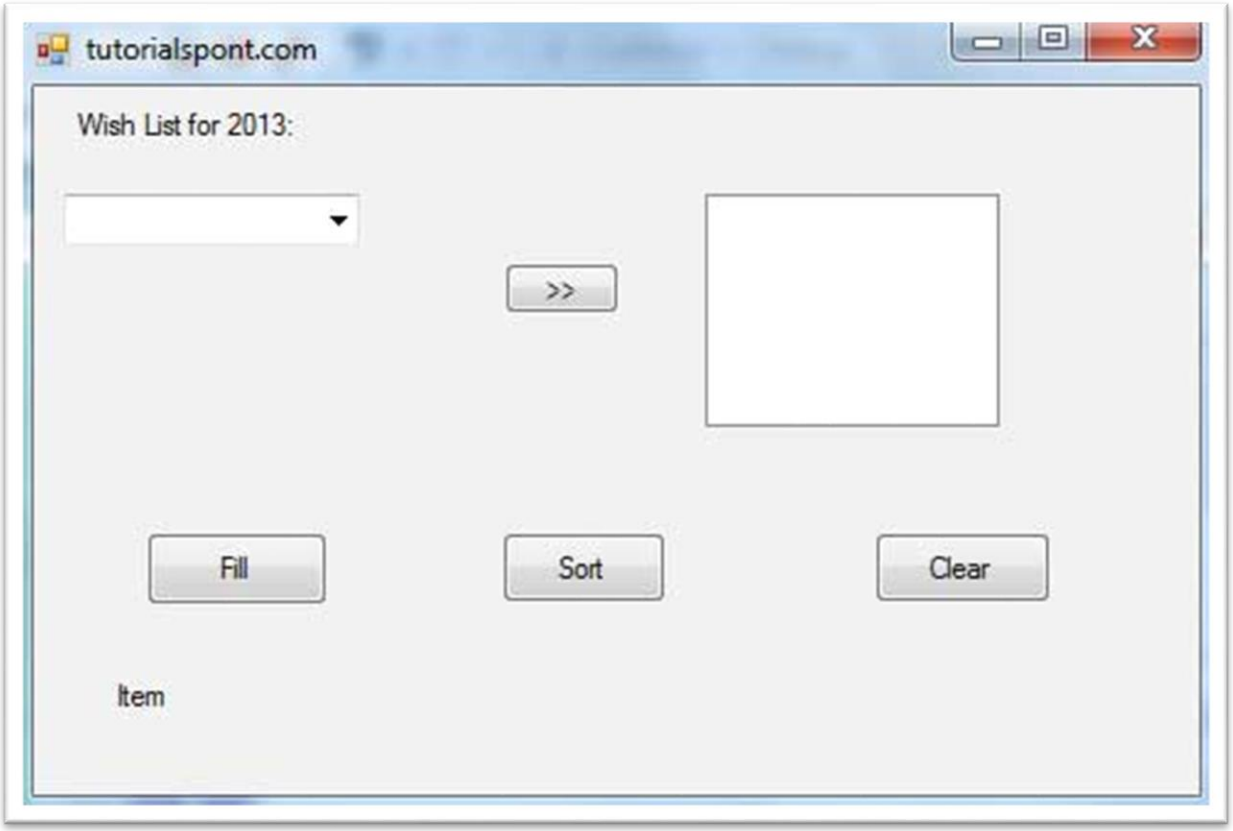
```
  'populates the list
  Private Sub Button2_Click(sender As Object, e As EventArgs) Handles Button2.Click
    ComboBox1.Items.Clear()
    ComboBox1.Items.Add("Safety")
    ComboBox1.Items.Add("Security")
    ComboBox1.Items.Add("Governance")
    ComboBox1.Items.Add("Good Music")
    ComboBox1.Items.Add("Good Movies")
    ComboBox1.Items.Add("Good Books")
    ComboBox1.Items.Add("Education")
    ComboBox1.Items.Add("Roads")
    ComboBox1.Items.Add("Health")
    ComboBox1.Items.Add("Food for all")
    ComboBox1.Items.Add("Shelter for all")
    ComboBox1.Items.Add("Industrialisation")
    ComboBox1.Items.Add("Peace")
    ComboBox1.Items.Add("Liberty")
    ComboBox1.Items.Add("Freedom of Speech")
    ComboBox1.Text = "Select from..."
  End Sub
  'sorting the list

  Private Sub Button3_Click(sender As Object, e As EventArgs)
    ComboBox1.Sorted = True
  End Sub
  'clears the list

  Private Sub Button4_Click(sender As Object, e As EventArgs)
    ComboBox1.Items.Clear()
  End Sub
  'displaying the selected item on the label

  Private Sub ComboBox1_SelectedIndexChanged(sender As Object, e As EventArgs) _
    Handles ListBox1.SelectedIndexChanged
    Label1.Text = ComboBox1.SelectedItem.ToString()
  End Sub
End Class
```

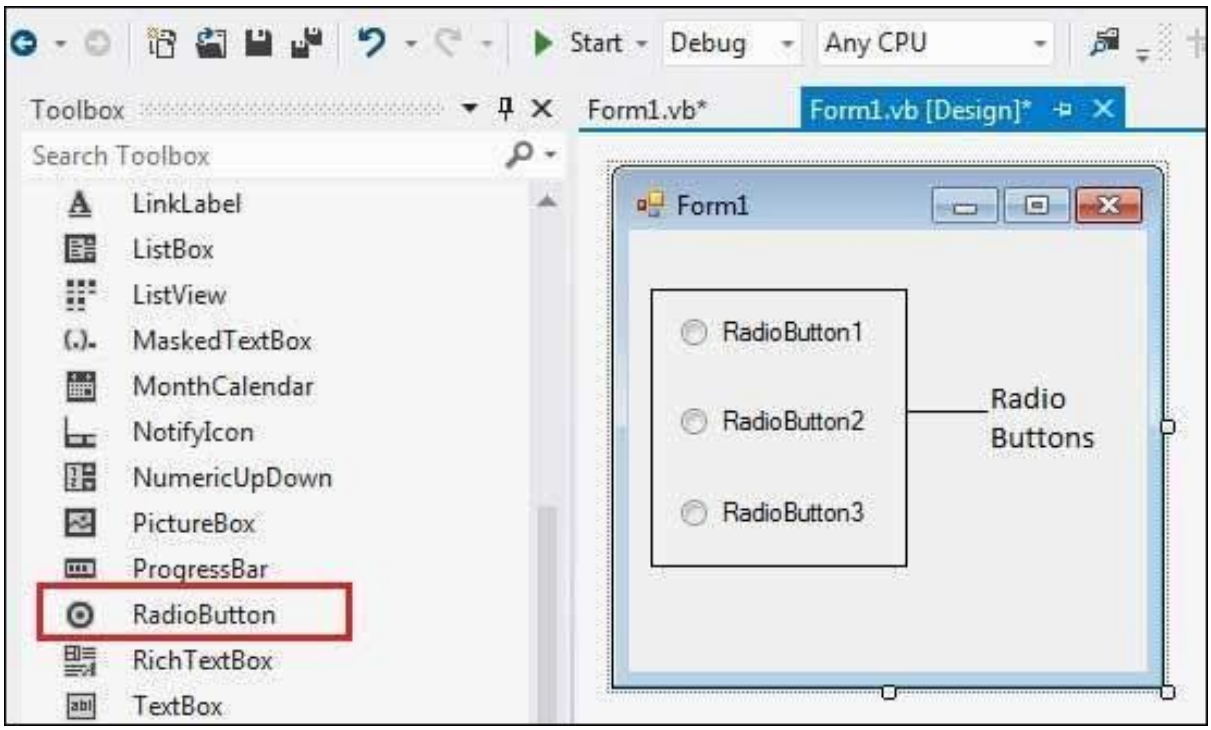
Output:

7. RADIOBUTTON

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.

Let's 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

The following are some of the commonly used properties of the RadioButton control –

Sr.No.	Property & Description
1	**Appearance** Gets or sets a value determining the appearance of the radio button.
2	**AutoCheck** Gets or sets a value indicating whether the Checked value and the appearance of the control automatically change when the control is clicked.
3	**CheckAlign** Gets or sets the location of the check box portion of the radio button.

4	**Checked** Gets or sets a value indicating whether the control is checked.
5	**Text** Gets or sets the caption for a radio button.
6	**TabStop** Gets or sets a value indicating whether a user can give focus to the RadioButton control using the TAB key.

Methods of the RadioButton Control

The following are some of the commonly used methods of the RadioButton control

–

Sr.No.	Method Name & Description
1	**PerformClick** Generates a Click event for the control, simulating a click by a user.

Events of the RadioButton Control

The following are some of the commonly used events of the RadioButton control –

Sr.No	Event & Description
1	**AppearanceChanged** Occurs when the value of the Appearance property of the RadioButton control is changed.
2	**CheckedChanged**

	Occurs when the value of the Checked property of the RadioButton control is changed.

Eg Program:

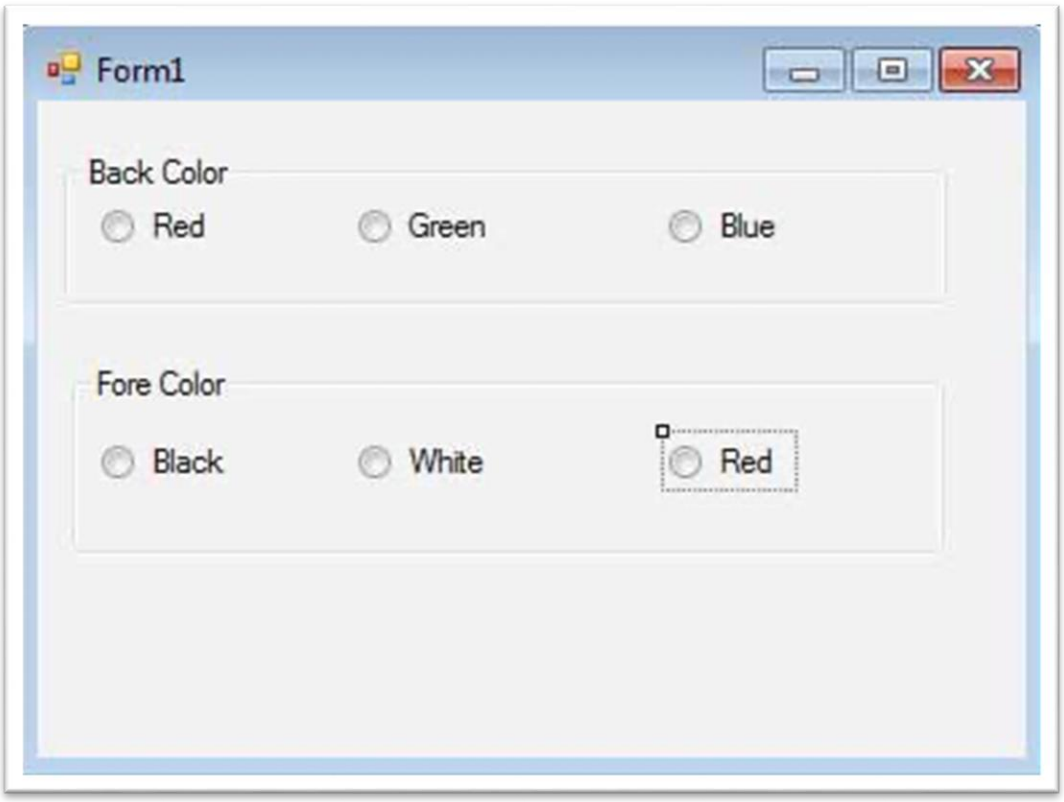
```
Public Class Form1
  Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    ' Set the caption bar text of the form.
    Me.Text = "tutorialspont.com"
  End Sub

  Private Sub RadioButton1_CheckedChanged(sender As Object, _
    e As EventArgs) Handles RadioButton1.CheckedChanged
    Me.BackColor = Color.Red
  End Sub

  Private Sub RadioButton2_CheckedChanged(sender As Object, _
    e As EventArgs) Handles RadioButton2.CheckedChanged
    Me.BackColor = Color.Green
  End Sub

  Private Sub RadioButton3_CheckedChanged(sender As Object, _
    e As EventArgs) Handles RadioButton3.CheckedChanged
    Me.BackColor = Color.Blue
  End Sub

  Private Sub RadioButton4_CheckedChanged(sender As Object, _
    e As EventArgs) Handles RadioButton4.CheckedChanged
    Me.ForeColor = Color.Black
  End Sub

  Private Sub RadioButton5_CheckedChanged(sender As Object, _
     e As EventArgs) Handles RadioButton5.CheckedChanged
    Me.ForeColor = Color.White
  End Sub

  Private Sub RadioButton6_CheckedChanged(sender As Object, _
    e As EventArgs) Handles RadioButton6.CheckedChanged
    Me.ForeColor = Color.Red
  End Sub
End Class
```

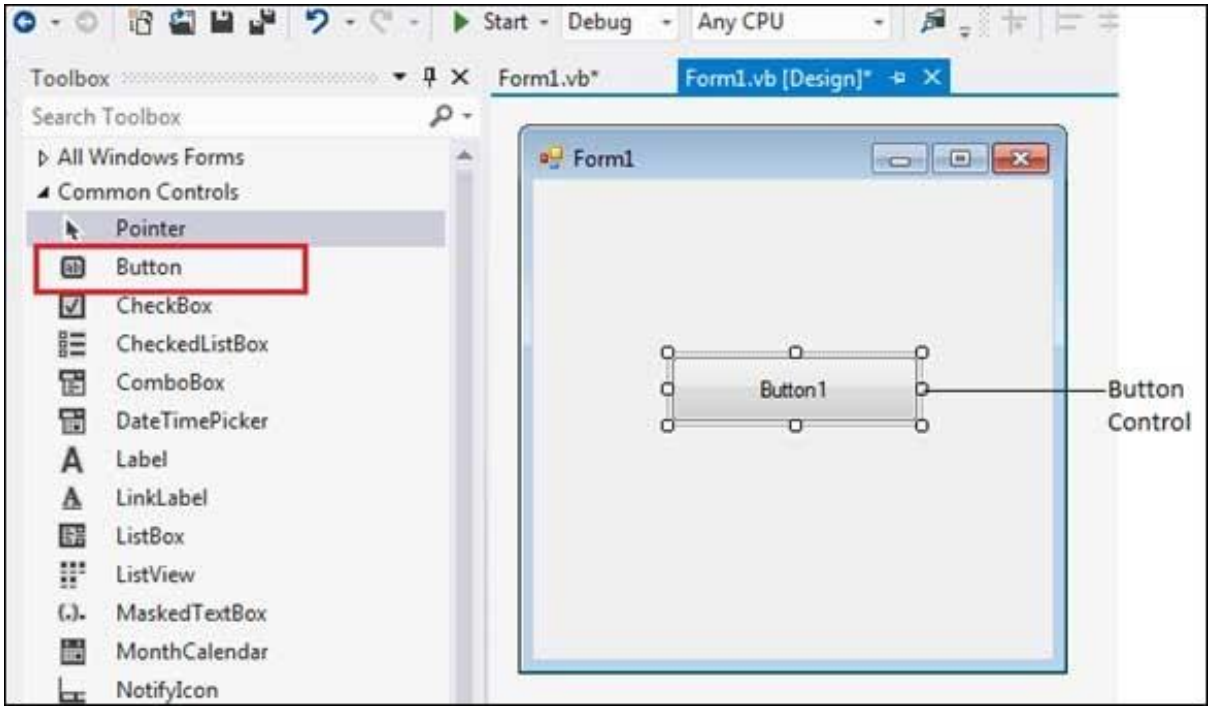
Output:

8. BUTTON

The Button control represents a standard Windows button. It is generally used to generate a Click event by providing a handler for the Click event.

Let's create a label by dragging a Button control from the Toolbox ad dropping it on the form.





Properties of the Button Control

The following are some of the commonly used properties of the Button control –

Sr.No.	Property & Description
1	**AutoSizeMode** Gets or sets the mode by which the Button automatically resizes itself.
2	**BackColor** Gets or sets the background color of the control.
3	**BackgroundImage** Gets or sets the background image displayed in the control.
4	**DialogResult** Gets or sets a value that is returned to the parent form when the button is clicked. This is used while creating dialog boxes.

5	**ForeColor** Gets or sets the foreground color of the control.
6	**Image** Gets or sets the image that is displayed on a button control.
7	**Location** Gets or sets the coordinates of the upper-left corner of the control relative to the upper-left corner of its container.
8	**TabIndex** Gets or sets the tab order of the control within its container.
9	**Text** Gets or sets the text associated with this control.

Methods of the Button Control

The following are some of the commonly used methods of the Button control –

Sr.No.	Method Name & Description
1	**GetPreferredSize** Retrieves the size of a rectangular area into which a control can be fitted.
2	**NotifyDefault** Notifies the Button whether it is the default button so that it can adjust its appearance accordingly.
3	**Select** Activates the control.

4	**ToString** Returns a String containing the name of the Component, if any. This method should not be overridden.

Events of the Button Control

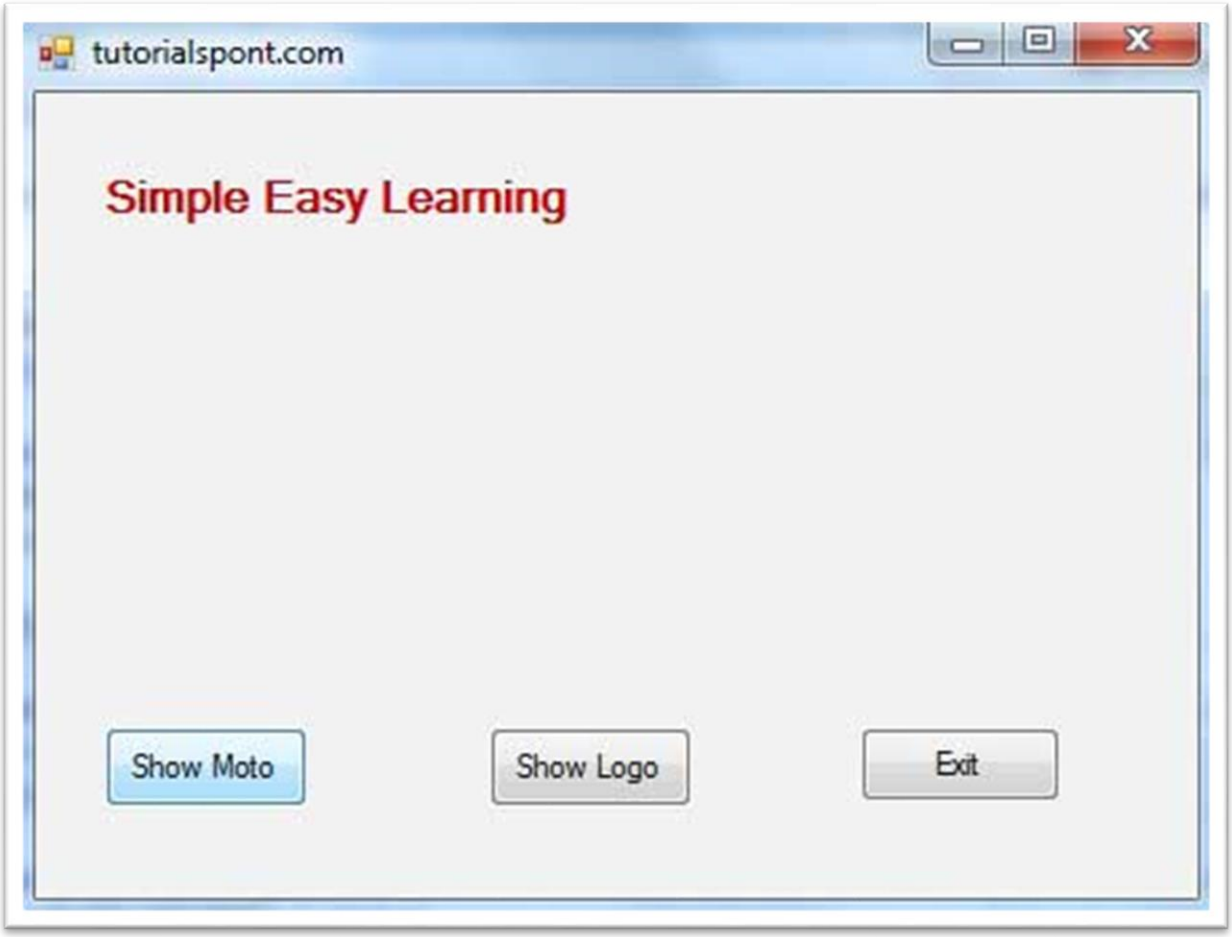
The following are some of the commonly used events of the Button control –

Sr.No.	Event & Description
1	**Click** Occurs when the control is clicked.
2	**DoubleClick** Occurs when the user double-clicks the Button control.
3	**GotFocus** Occurs when the control receives focus.
4	**TabIndexChanged** Occurs when the TabIndex property value changes.
5	**TextChanged** Occurs when the Text property value changes.
6	**Validated** Occurs when the control is finished validating.

Eg Program:

```
Public Class Form1
  Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    ' Set the caption bar text of the form.
    Me.Text = "tutorialspont.com"
    btnImage.Visible = False
  End Sub

  Private Sub btnMoto_Click(sender As Object, e As EventArgs) Handles btnMoto.Click
    btnImage.Visible = False
    Label1.Text = "Simple Easy Learning"
  End Sub

  Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
    Application.Exit()
  End Sub

  Private Sub btnLogo_Click(sender As Object, e As EventArgs) Handles btnLogo.Click
    Label1.Visible = False
    btnImage.Visible = True
  End Sub
End Class
```

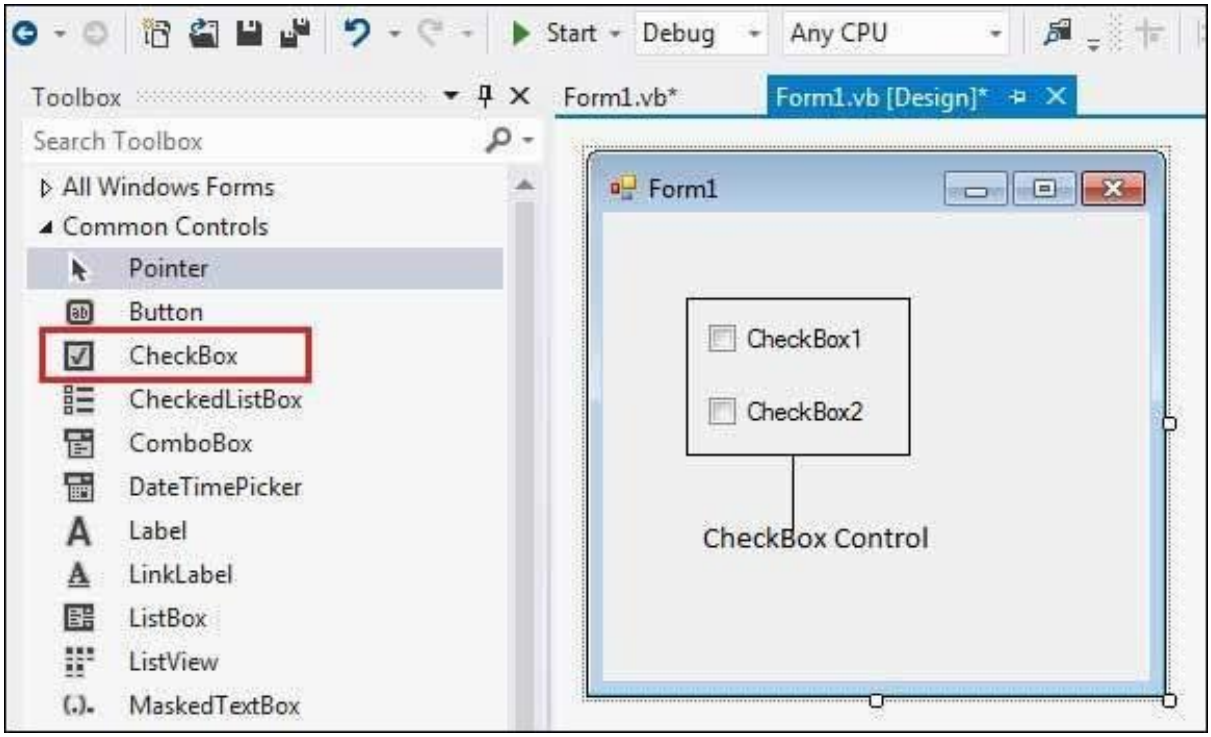
Output:



9. CHECK BOX

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.

Let's create two check boxes by dragging CheckBox controls from the Toolbox and dropping on the form.



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

The following are some of the commonly used properties of the CheckBox control –

Sr.No.	Property & Description
1	**Appearance** Gets or sets a value determining the appearance of the check box.
2	**AutoCheck** Gets or sets a value indicating whether the Checked or CheckedState value and the appearance of the control automatically change when the check box is selected.
3	**CheckAlign**

	Gets or sets the horizontal and vertical alignment of the check mark on the check box.
4	**Checked** Gets or sets a value indicating whether the check box is selected.
5	**CheckState** Gets or sets the state of a check box.
6	**Text** Gets or sets the caption of a check box.
7	**ThreeState** Gets or sets a value indicating whether or not a check box should allow three check states rather than two.

Methods of the CheckBox Control

The following are some of the commonly used methods of the CheckBox control –

Sr.No.	Method Name & Description
1	**OnCheckedChanged** Raises the CheckedChanged event.
2	**OnCheckStateChanged** Raises the CheckStateChanged event.
3	**OnClick** Raises the OnClick event.

Events of the CheckBox Control

The following are some of the commonly used events of the CheckBox control –

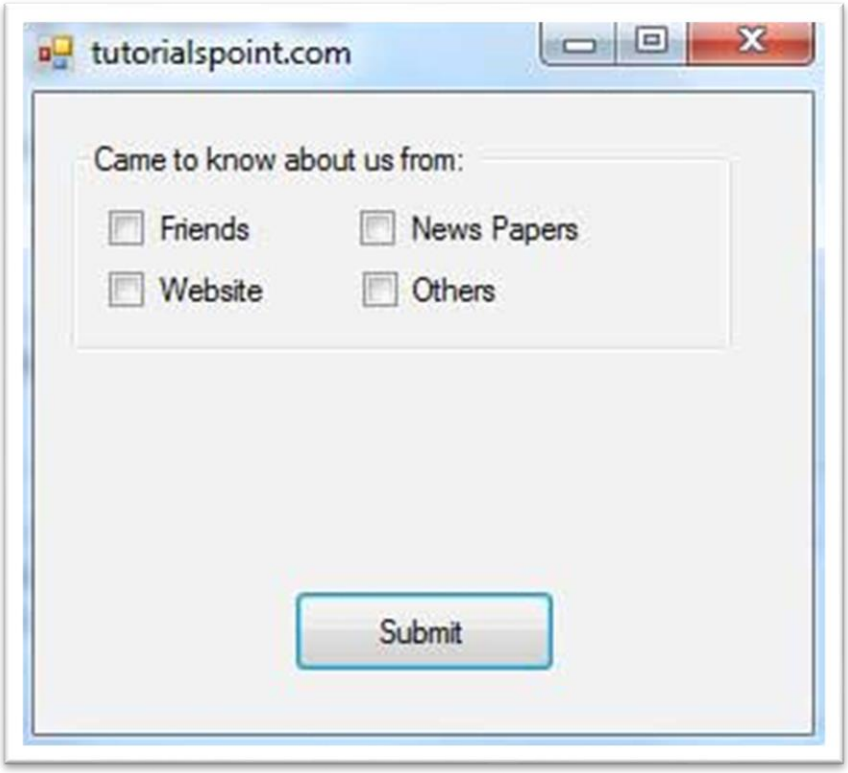
Sr.No.	Event & Description
1	**AppearanceChanged** Occurs when the value of the Appearance property of the check box is changed.
2	**CheckedChanged** Occurs when the value of the Checked property of the CheckBox control is changed.
3	**CheckStateChanged** Occurs when the value of the CheckState property of the CheckBox control is changed.

Eg Program

```
Public Class Form1
  Private Sub Form1_Load(sender As Object, e As EventArgs) _
    Handles MyBase.Load
    ' Set the caption bar text of the form.
    Me.Text = "tutorialspoint.com"
    Label1.Visible = False
    TextBox1.Visible = False
    TextBox1.Multiline = True
  End Sub

  Private Sub Button1_Click(sender As Object, e As EventArgs) _
    Handles Button1.Click
    Dim str As String
    str = " "

    If CheckBox1.Checked = True Then
      str &= CheckBox1.Text
```

```
      str &= " "
    End If

    If CheckBox2.Checked = True Then
      str &= CheckBox2.Text
      str &= " "
    End If

    If CheckBox3.Checked = True Then
      str &= CheckBox3.Text
      str &= " "
    End If

    If CheckBox4.Checked = True Then
      str &= TextBox1.Text
      str &= " "
    End If
    If str <> Nothing Then
      MsgBox(str + vbLf + "Thank you")
    End If
  End Sub

  Private Sub CheckBox4_CheckedChanged(sender As Object, _
    e As EventArgs) Handles CheckBox4.CheckedChanged
    Label1.Visible = True
    TextBox1.Visible = True
  End Sub
End Class
```

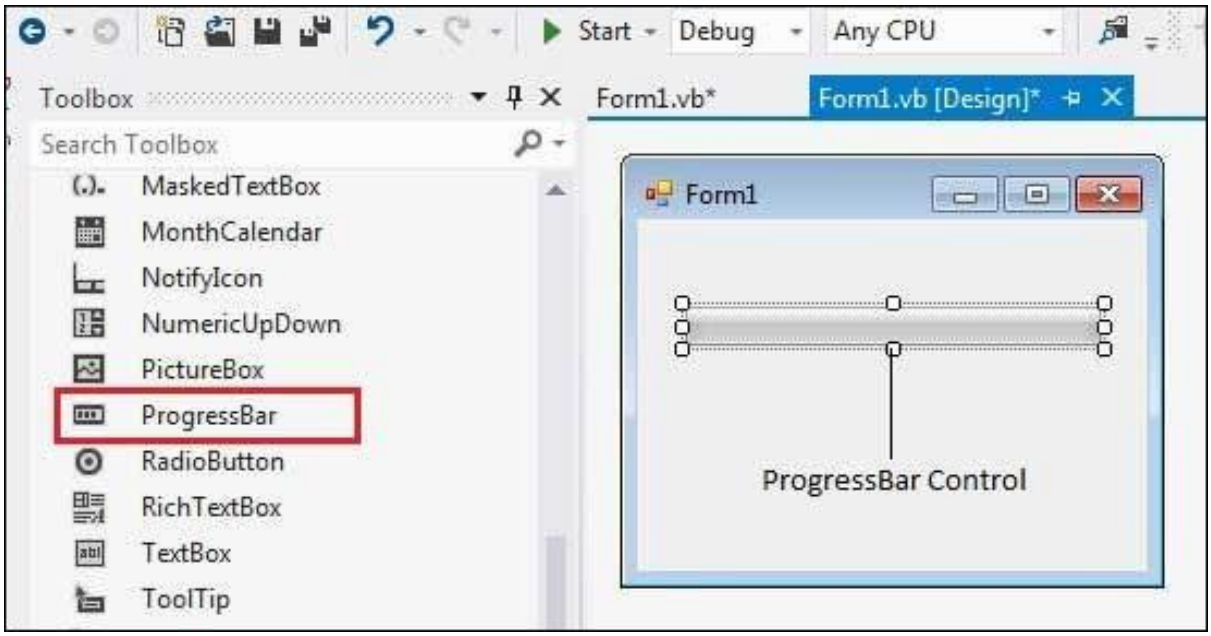
Output:



10. PROGRESS BAR

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

Let's click on a ProgressBar control from the Toolbox and place it on the form.



The main properties of a progress bar are *Value*, *Maximum* and *Minimum*. The Minimum and Maximum properties are used to set the minimum and maximum values that the progress bar can display. The Value property specifies the current position of the progress bar.

The ProgressBar control is typically used when an application performs tasks such as copying files or printing documents. To a user the application might look unresponsive if there is no visual cue. In such cases, using the ProgressBar allows the programmer to provide a visual status of progress.

Properties of the ProgressBar Control

The following are some of the commonly used properties of the ProgressBar control

–

Sr.No.	Property & Description
1	**AllowDrop** Overrides Control.AllowDrop.
2	**BackgroundImage** Gets or sets the background image for the ProgressBar control.
3	**BackgroundImageLayout** Gets or sets the layout of the background image of the progress bar.
4	**CausesValidation** Gets or sets a value indicating whether the control, when it receives focus, causes validation to be performed on any controls that require validation.
5	**Font** Gets or sets the font of text in the ProgressBar.
6	**ImeMode** Gets or sets the input method editor (IME) for the ProgressBar.

7	**ImeModeBase** Gets or sets the IME mode of a control.
8	**MarqueeAnimationSpeed** Gets or sets the time period, in milliseconds, that it takes the progress block to scroll across the progress bar.
9	**Maximum** Gets or sets the maximum value of the range of the control.v
10	**Minimum** Gets or sets the minimum value of the range of the control.
11	**Padding** Gets or sets the space between the edges of a ProgressBar control and its contents.
12	**RightToLeftLayout** Gets or sets a value indicating whether the ProgressBar and any text it contains is displayed from right to left.
13	**Step** Gets or sets the amount by which a call to the PerformStep method increases the current position of the progress bar.
14	**Style** Gets or sets the manner in which progress should be indicated on the progress bar.
15	**Value** Gets or sets the current position of the progress bar.v

Methods of the ProgressBar Control

The following are some of the commonly used methods of the ProgressBar control –

Sr.No.	Method Name & Description
1	**Increment** Increments the current position of the ProgressBar control by specified amount.
2	**PerformStep** Increments the value by the specified step.
3	**ResetText** Resets the Text property to its default value.
4	**ToString** Returns a string that represents the progress bar control.

Events of the ProgressBar Control

The following are some of the commonly used events of the ProgressBar control –

Sr.No.	Event & Description
1	**BackgroundImageChanged** Occurs when the value of the BackgroundImage property changes.
2	**BackgroundImageLayoutChanged** Occurs when the value of the BackgroundImageLayout property changes.
3	**CausesValidationChanged** Occurs when the value of the CausesValidation property changes.

4	**Click** Occurs when the control is clicked.
5	**DoubleClick** Occurs when the user double-clicks the control.
6	**Enter** Occurs when focus enters the control.
7	**FontChanged** Occurs when the value of the Font property changes.
8	**ImeModeChanged** Occurs when the value of the ImeMode property changes.
9	**KeyDown** Occurs when the user presses a key while the control has focus.
10	**KeyPress** Occurs when the user presses a key while the control has focus.
11	**KeyUp** Occurs when the user releases a key while the control has focus.
12	**Leave** Occurs when focus leaves the ProgressBar control.
13	**MouseClick** Occurs when the control is clicked by the mouse.
14	**MouseDoubleClick** Occurs when the user double-clicks the control.

15	**PaddingChanged** Occurs when the value of the Padding property changes.
16	**Paint** Occurs when the ProgressBar is drawn.
17	**RightToLeftLayoutChanged** Occurs when the RightToLeftLayout property changes.
18	**TabStopChanged** Occurs when the TabStop property changes.
19	**TextChanged** Occurs when the Text property changes.

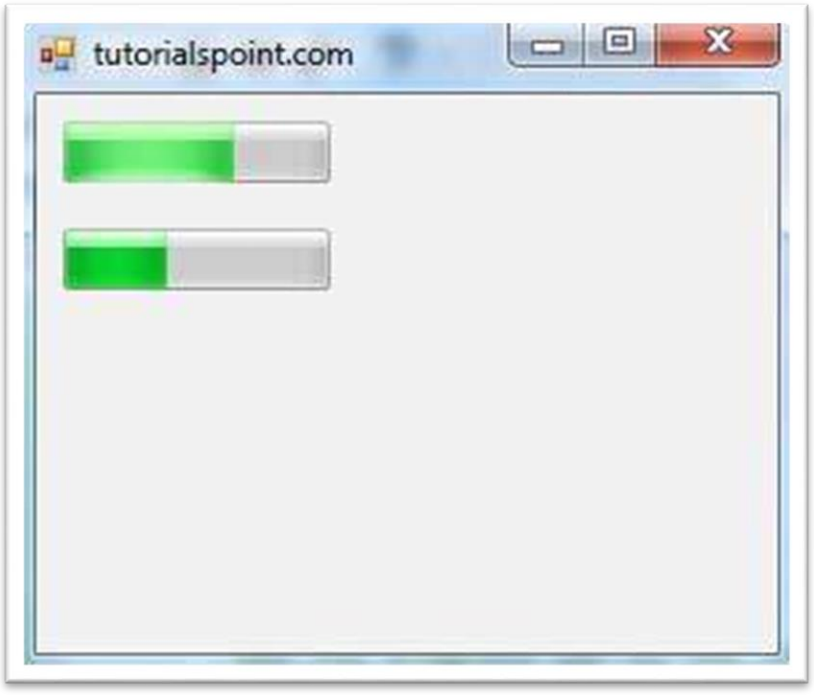
Eg Program

```
Public Class Form1
  Private Sub Form1_Load(sender As Object, e As EventArgs) _
    Handles MyBase.Load
    'create two progress bars
    Dim ProgressBar1 As ProgressBar
    Dim ProgressBar2 As ProgressBar
    ProgressBar1 = New ProgressBar()
    ProgressBar2 = New ProgressBar()
    'set position
    ProgressBar1.Location = New Point(10, 10)
    ProgressBar2.Location = New Point(10, 50)
    'set values
    ProgressBar1.Minimum = 0
    ProgressBar1.Maximum = 200
    ProgressBar1.Value = 130
    ProgressBar2.Minimum = 0
    ProgressBar2.Maximum = 100
    ProgressBar2.Value = 40
    'add the progress bar to the form
    Me.Controls.Add(ProgressBar1)
    Me.Controls.Add(ProgressBar2)
    ' Set the caption bar text of the form.
    Me.Text = "tutorialspoint.com"
  End Sub
```

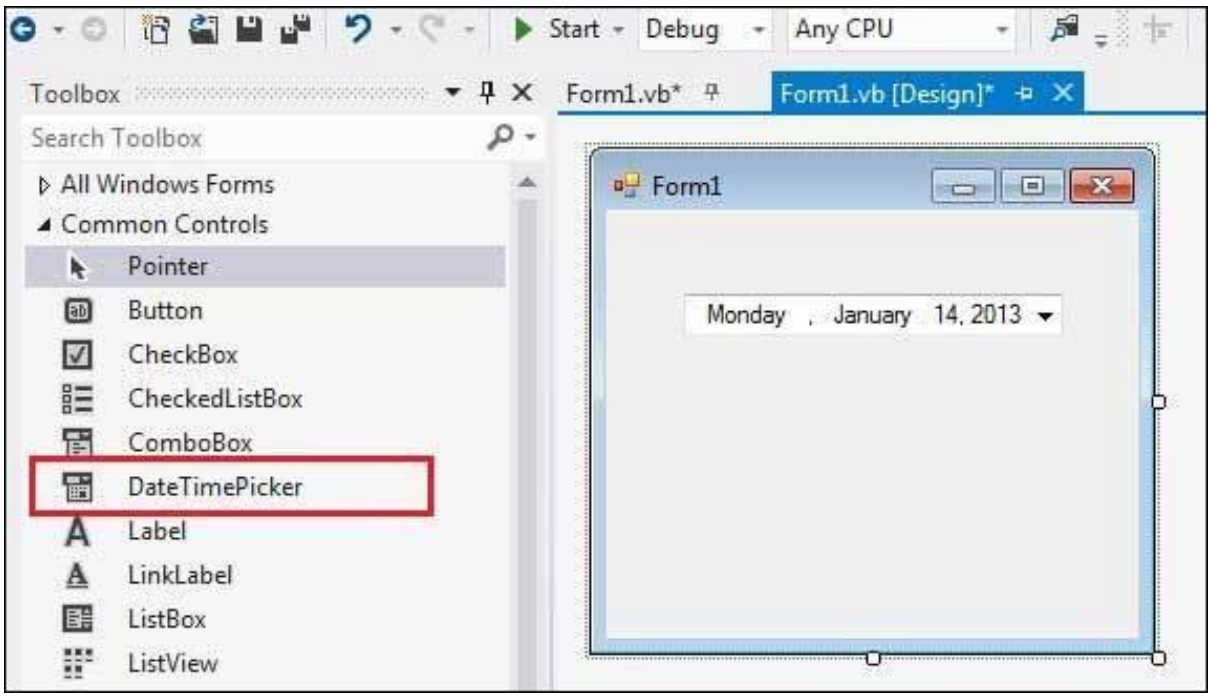
```
End Class
```

Output:

11. DATE TIME PICKER



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



The **MinDate** and the **MaxDate** properties allow you to put limits on the date range.

Properties of the DateTimePicker Control

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

Sr.No.	Property & Description
1	**BackColor** Gets or sets a value indicating the background color of the DateTimePicker control.
2	**BackgroundImage** Gets or sets the background image for the control.
3	**BackgroundImageLayout** Gets or sets the layout of the background image of the DateTimePicker control.
4	**CalendarFont**

	Gets or sets the font style applied to the calendar.
5	**CalendarForeColor** Gets or sets the foreground color of the calendar.
6	**CalendarMonthBackground** Gets or sets the background color of the calendar month.
7	**CalendarTitleBackColor** Gets or sets the background color of the calendar title.
8	**CalendarTitleForeColor** Gets or sets the foreground color of the calendar title.
9	**CalendarTrailingForeColor** Gets or sets the foreground color of the calendar trailing dates.
10	**Checked** Gets or sets a value indicating whether the Value property has been set with a valid date/time value and the displayed value is able to be updated.
11	**CustomFormat** Gets or sets the custom date/time format string.
12	**DropDownAlign** Gets or sets the alignment of the drop-down calendar on the DateTimePicker control.
13	**ForeColor** Gets or sets the foreground color of the DateTimePicker control.
14	**Format** Gets or sets the format of the date and time displayed in the control.

15	**MaxDate** Gets or sets the maximum date and time that can be selected in the control.
16	**MaximumDateTime** Gets the maximum date value allowed for the DateTimePicker control.
17	**MinDate** Gets or sets the minimum date and time that can be selected in the control.
18	**MinimumDateTime** Gets the minimum date value allowed for the DateTimePicker control.
19	**PreferredHeight** Gets the preferred height of the DateTimePicker control.
20	**RightToLeftLayout** Gets or sets whether the contents of the DateTimePicker are laid out from right to left.
21	**ShowCheckBox** Gets or sets a value indicating whether a check box is displayed to the left of the selected date.
22	**ShowUpDown** Gets or sets a value indicating whether a spin button control (also known as an up-down control) is used to adjust the date/time value.
23	**Text** Gets or sets the text associated with this control.
24	**Value** Gets or sets the date/time value assigned to the control.

Methods of the DateTimePicker Control

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

Sr.No.	Method Name & Description
1	**ToString** Returns the string representing the control.

Events of the DateTimePicker Control

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

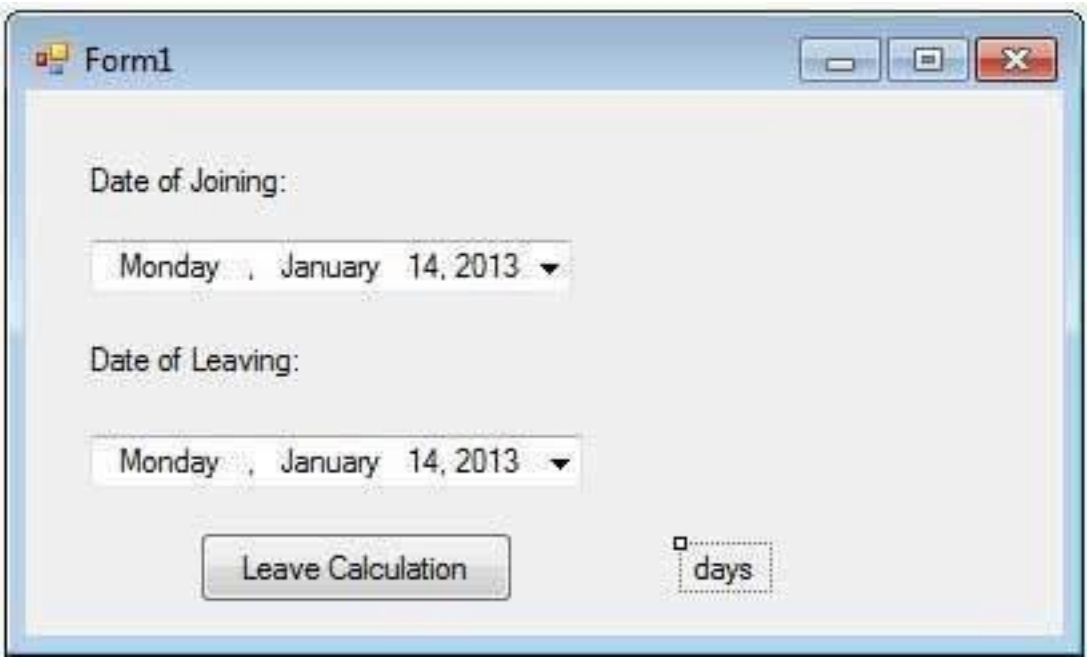
Sr.No.	Event & Description
1	**BackColorChanged** Occurs when the value of the BackColor property changes.
2	**BackgroundImageChanged** Occurs when the value of the BackgroundImage property changes.
3	**BackgroundImageLayoutChanged** Occurs when the value of the BackgroundImageLayout property changes.
4	**Click** Occurs when the control is clicked.
5	**CloseUp** Occurs when the drop-down calendar is dismissed and disappears.
6	**DoubleClick** Occurs when the control is double-clicked.
7	**DragDrop** Occurs when a drag-and-drop operation is completed.
8	**ForeColorChanged**

	Occurs when the value of the ForeColor property changes.
9	**FormatChanged** Occurs when the Format property value has changed.
10	**MouseClick** Occurs when the control is clicked with the mouse.
11	**MouseDoubleClick** Occurs when the control is double-clicked with the mouse.
12	**PaddingChanged** Occurs when the value of the Padding property changes.
13	**Paint** Occurs when the control is redrawn.
14	**RightToLeftLayoutChanged** Occurs when the RightToLeftLayout property changes.
15	**TextChanged** Occurs when the value of the Text property changes.
16	**ValueChanged** Occurs when the Value property changes.

Example

In this example, let us create a small application for calculating days of leave. Let us add two DateTimePicker controls on the form, where the user will enter the date of going on leave and the date of joining. Let us keep a button control for performing the calculation and appropriate label controls for displaying information.

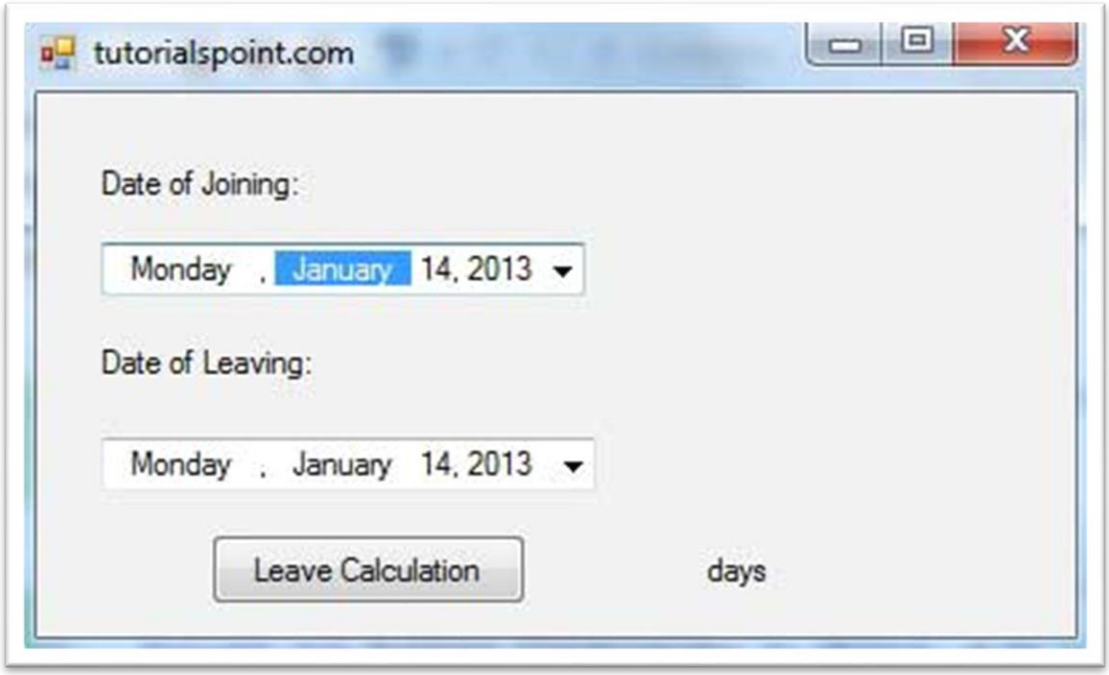
The form in design view −



Add the following code in the code editor window –

```
Public Class Form1
  Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    ' Set the caption bar text of the form.
    Me.Text = "tutorialspoint.com"
  End Sub

  Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
    Dim d1 As DateTime = DateTimePicker1.Value
    Dim d2 As DateTime = DateTimePicker2.Value
    Dim result As TimeSpan = d1.Subtract(d2)
    Dim days As Integer = result.TotalDays
    Label3.Text = days
  End Sub
End Class
```

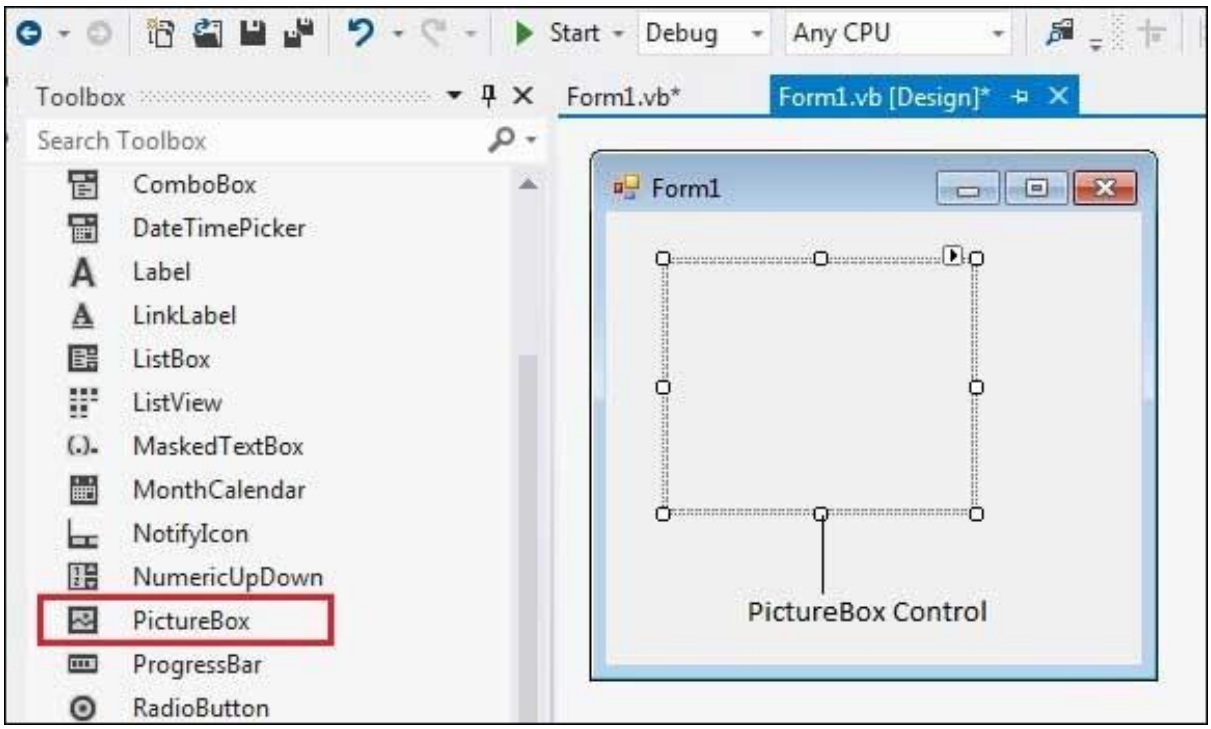
Output:



12. PICTURE BOX

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

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



Properties of the PictureBox Control

The following are some of the commonly used properties of the PictureBox control –

Sr.No.	Property & Description
1	**AllowDrop** Specifies whether the picture box accepts data that a user drags on it.
2	**ErrorImage** Gets or specifies an image to be displayed when an error occurs during the image-loading process or if the image load is cancelled.
3	**Image** Gets or sets the image that is displayed in the control.
4	**ImageLocation** Gets or sets the path or the URL for the image displayed in the control.

5	**InitialImage** Gets or sets the image displayed in the control when the main image is loaded.
6	**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.
7	**TabIndex** Gets or sets the tab index value.
8	**TabStop** Specifies whether the user will be able to focus on the picture box by using the TAB key.
9	**Text** Gets or sets the text for the picture box.
10	**WaitOnLoad** Specifies whether or not an image is loaded synchronously.

Methods of the PictureBox Control

The following are some of the commonly used methods of the PictureBox control –

Sr.No.	Method Name & Description

1	**CancelAsync** Cancels an asynchronous image load.
2	**Load** Displays an image in the picture box
3	**LoadAsync** Loads image asynchronously.
4	**ToString** Returns the string that represents the current picture box.

Events of the PictureBox Control

The following are some of the commonly used events of the PictureBox control –

Sr.No.	Event & Description
1	**CausesValidationChanged** Overrides the Control.CausesValidationChanged property.
2	**Click** Occurs when the control is clicked.
3	**Enter** Overrides the Control.Enter property.
4	**FontChanged** Occurs when the value of the Font property changes.
5	**ForeColorChanged**

	Occurs when the value of the ForeColor property changes.
6	**KeyDown** Occurs when a key is pressed when the control has focus.
7	**KeyPress** Occurs when a key is pressed when the control has focus.
8	**KeyUp** Occurs when a key is released when the control has focus.
9	**Leave** Occurs when input focus leaves the PictureBox.
10	**LoadCompleted** Occurs when the asynchronous image-load operation is completed, been canceled, or raised an exception.
11	**LoadProgressChanged** Occurs when the progress of an asynchronous image-loading operation has changed.
12	**Resize** Occurs when the control is resized.
13	**RightToLeftChanged** Occurs when the value of the RightToLeft property changes.
14	**SizeChanged** Occurs when the Size property value changes.
15	**SizeModeChanged** Occurs when SizeMode changes.

16	**TabIndexChanged** Occurs when the value of the TabIndex property changes.
17	**TabStopChanged** Occurs when the value of the TabStop property changes.
18	**TextChanged** Occurs when the value of the Text property changes.

Eg Program

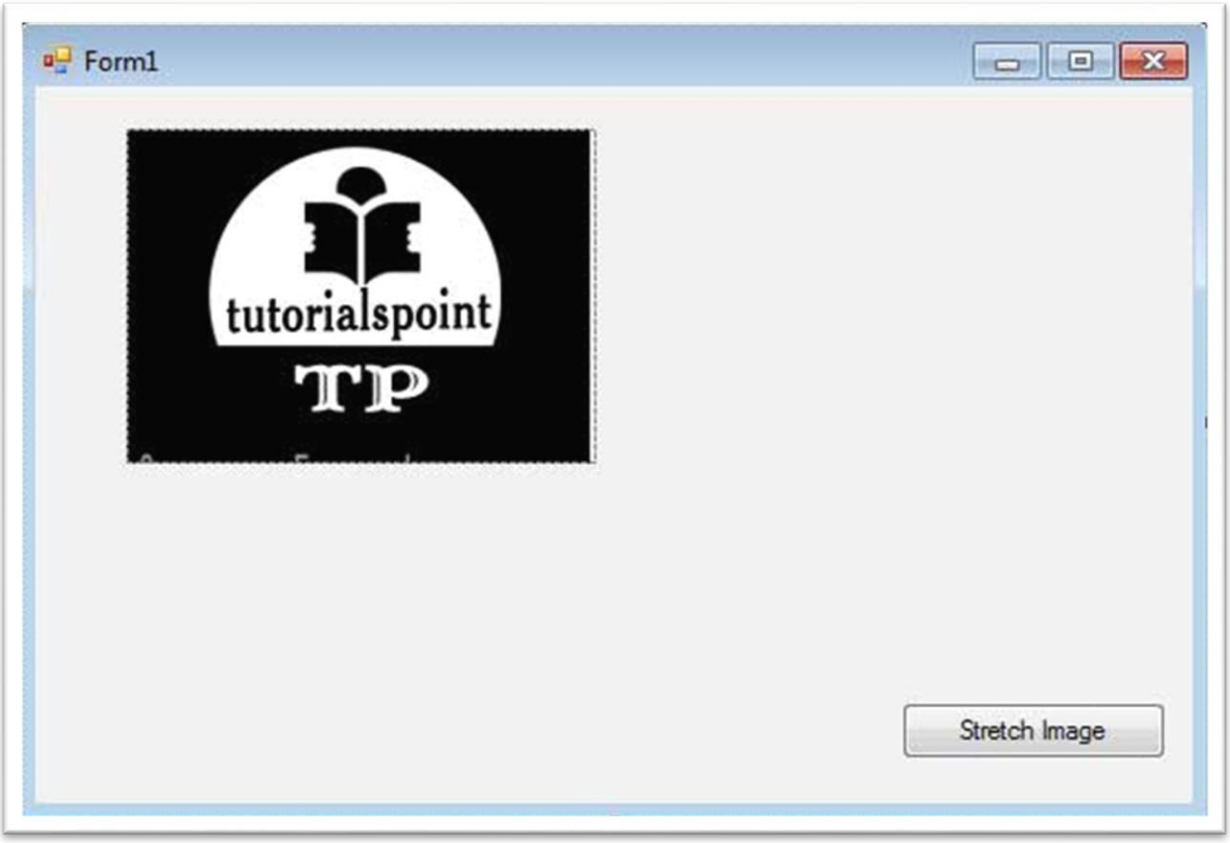
```
Public Class Form1
  Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    ' Set the caption bar text of the form.
    Me.Text = "tutorialspoint.com"
  End Sub

  Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
    PictureBox1.ClientSize = New Size(300, 300)
    PictureBox1.SizeMode = PictureBoxSizeMode.StretchImage
  End Sub
End Class
```

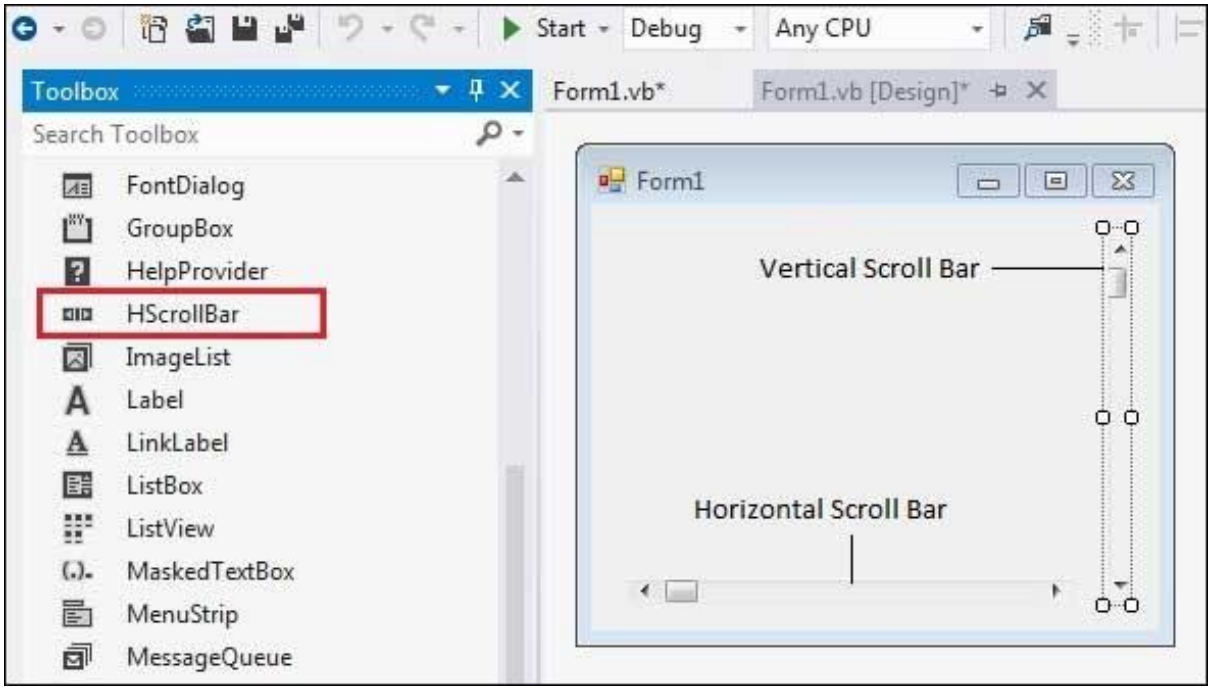
Output:

13. SCROLLBAR

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

Let's click on HScrollBar control and VScrollBar control from the Toolbox and place them on the form.





Properties of the ScrollBar Control

The following are some of the commonly used properties of the ScrollBar control –

Sr.No.	Property & Description
1	**AutoSize** Gets or sets a value indicating whether the ScrollBar is automatically resized to fit its contents.
2	**BackColor** Gets or sets the background color for the control.
3	**ForeColor** Gets or sets the foreground color of the scroll bar control.
4	**ImeMode** Gets or sets the Input Method Editor (IME) mode supported by this control.
5	**LargeChange**

	Gets or sets a value to be added to or subtracted from the Value property when the scroll box is moved a large distance.
6	**Maximum** Gets or sets the upper limit of values of the scrollable range.
7	**Minimum** Gets or sets the lower limit of values of the scrollable range.
8	**SmallChange** Gets or sets the value to be added to or subtracted from the Value property when the scroll box is moved a small distance.
9	**Value** Gets or sets a numeric value that represents the current position of the scroll box on the scroll bar control.

Methods of the ScrollBar Control

The following are some of the commonly used methods of the ScrollBar control –

Sr.No.	Method Name & Description
1	**OnClick** Generates the Click event.
2	**Select** Activates the control.

Events of the ScrollBar Control

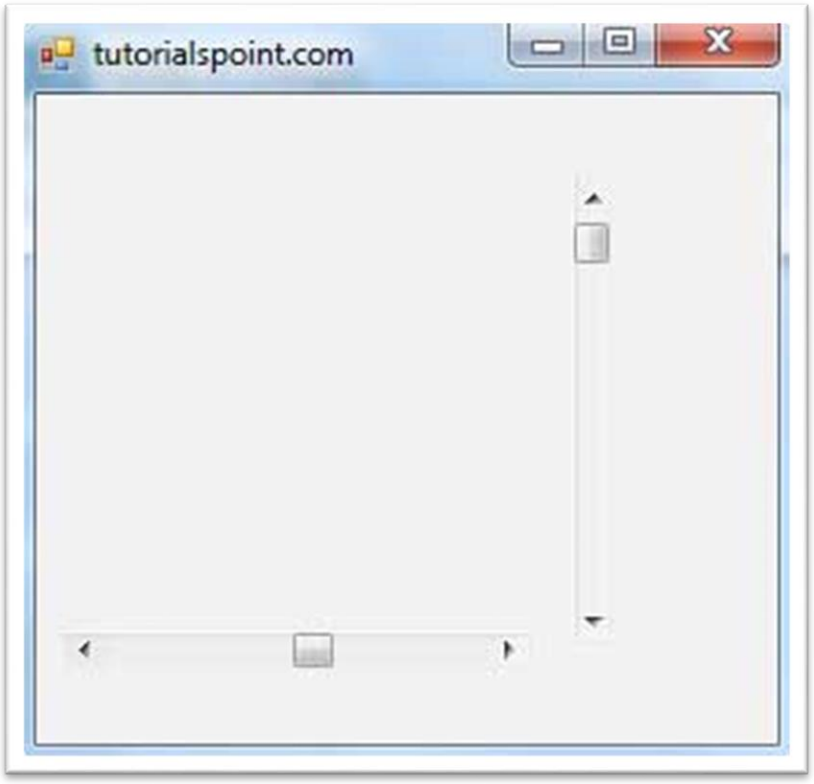
The following are some of the commonly used events of the ScrollBar control –

Sr.No.	Event & Description
1	**Click** Occurs when the control is clicked.
2	**DoubleClick** Occurs when the user double-clicks the control.
3	**Scroll** Occurs when the control is moved.
4	**ValueChanged** Occurs when the Value property changes, either by handling the Scroll event or programmatically.

Eg Program

```
Public Class Form1
  Private Sub Form1_Load(sender As Object, e As EventArgs) _
    Handles MyBase.Load

    'create two scroll bars
    Dim hs As HScrollBar
    Dim vs As VScrollBar
    hs = New HScrollBar()
    vs = New VScrollBar()

    'set properties
    hs.Location = New Point(10, 200)
    hs.Size = New Size(175, 15)
    hs.Value = 50
    vs.Location = New Point(200, 30)
    vs.Size = New Size(15, 175)
    hs.Value = 50

    'adding the scroll bars to the form
    Me.Controls.Add(hs)
    Me.Controls.Add(vs)
    ' Set the caption bar text of the form.
    Me.Text = "tutorialspoint.com"
  End Sub
End Class
```

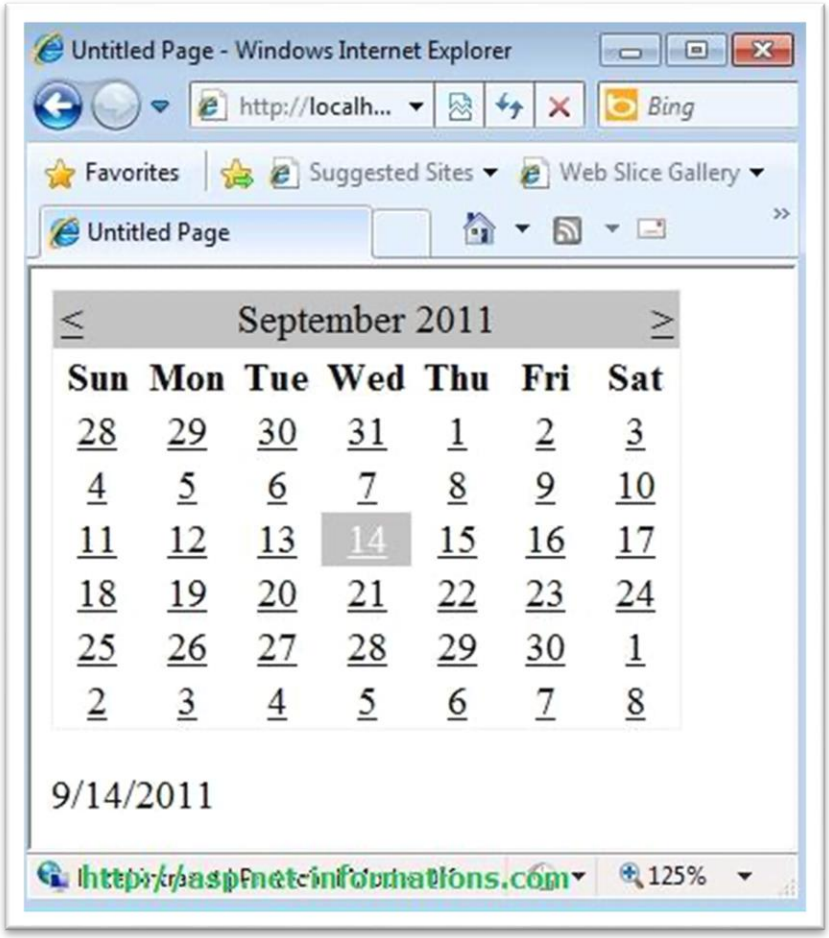
Output:



15. CALENDER

The Calendar control is used to display a calendar in the browser. The control allows you to select dates and move to the next or previous month.

You can customize the appearance of the Calendar control by setting the properties that control the style for different parts of the control. The following ASP.NET program display the selected Calender date in short date format in a label control.



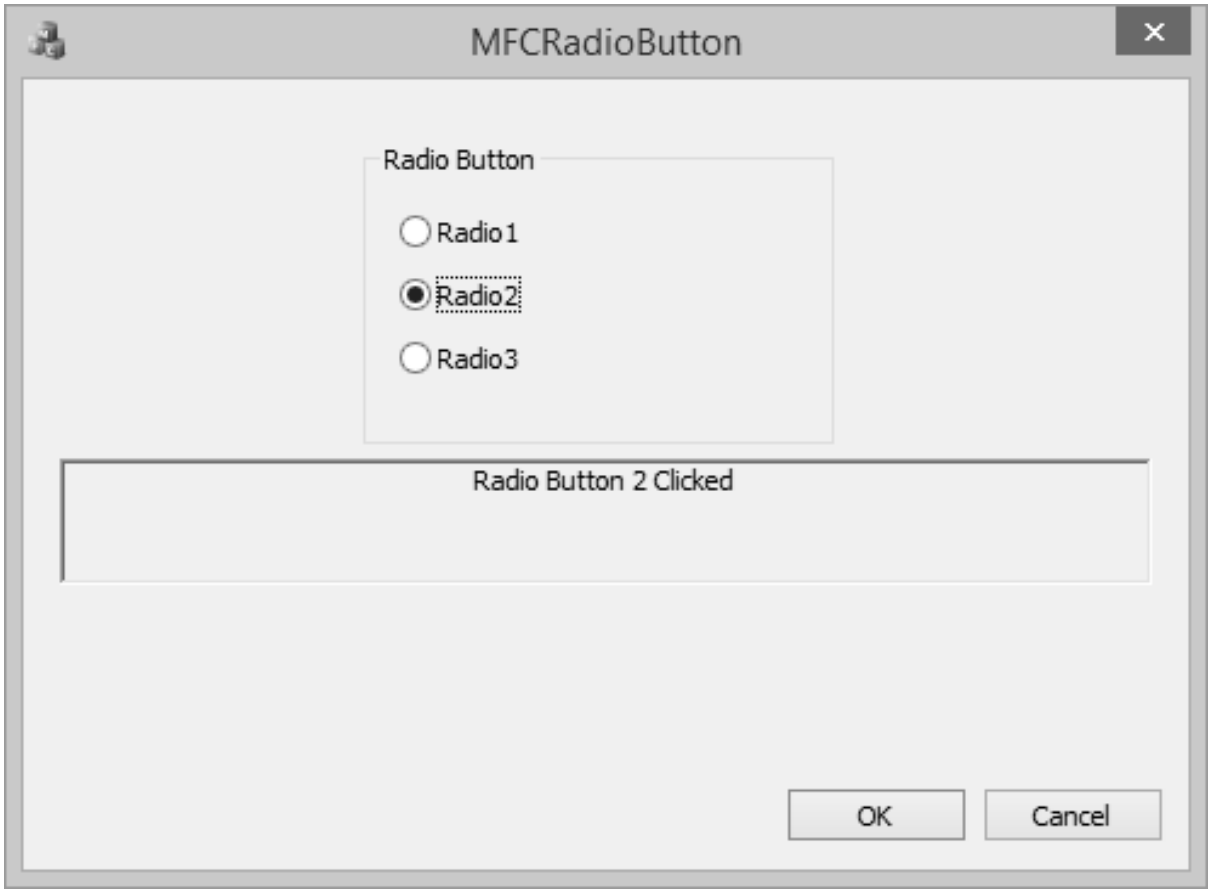

```
Imports System.Data

Imports System.Data.SqlClient Imports
System.Configuration
```

```
    Partial Class _Default

    Inherits System.Web.UI.Page

    Protected Sub Calendar1_SelectionChanged(ByVal sender
    As Object, ByVal e As System.EventArgs) Handles
    Calendar1.SelectionChanged

    Label1.Text =
    Calendar1.SelectedDate.ToShortDateString() End Sub

End Class
```

14. GROUP BOX

A **group box** is a static control used to set a visible or programmatic group of controls. The control is a rectangle that groups other controls together. The controls are grouped by drawing a border around them and displaying the given text in the upper-left corner.

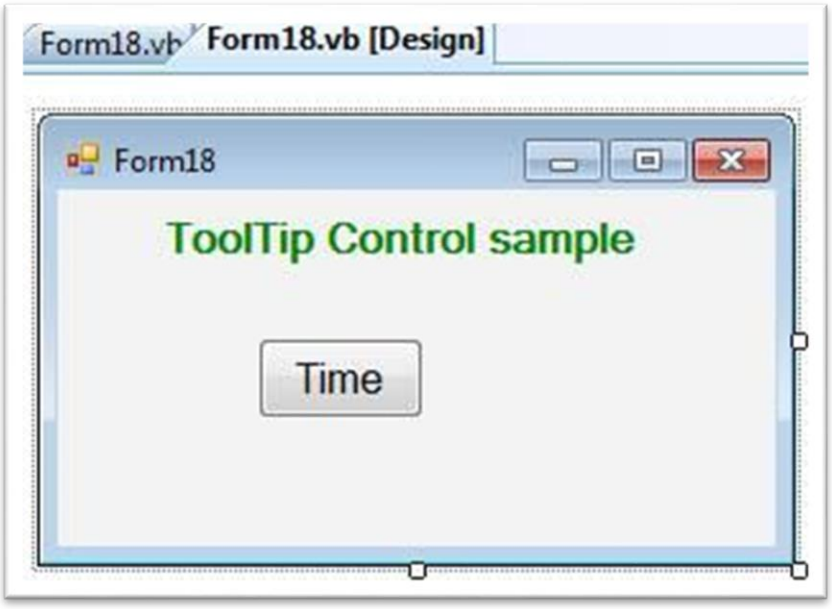
In the following dialog box, the Group box contains three radio buttons inside.

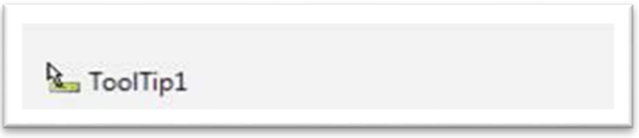


15. TOOLTIP

The ToolTip control is used to display some text when the mouse is move over a control. This control is helpful in providing quick information when the user moves the pointer over an associated control.
How to use ToolTip control

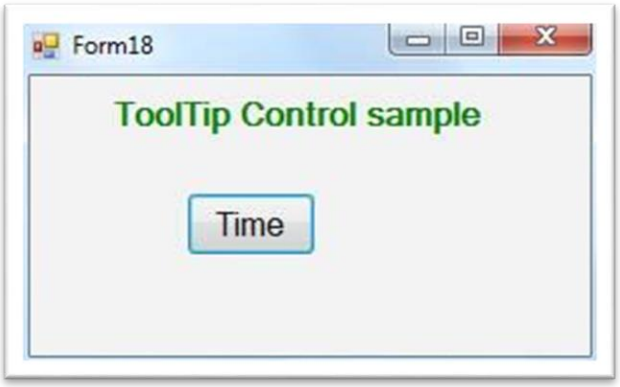
Drag and drop a Tooltip control and control on which you want to use tooltip control






```
Public Class Form18
Private Sub Button1_MouseHover(ByVal sender As System.Object, ByVal e As System.E

ventArgs) Handles Button1.MouseHover
'show tooltip on button

ToolTip1.SetToolTip(Button1, 'Click to show time') ' SetToolTip method assign a ToolTip with control End Sub

End Class
```

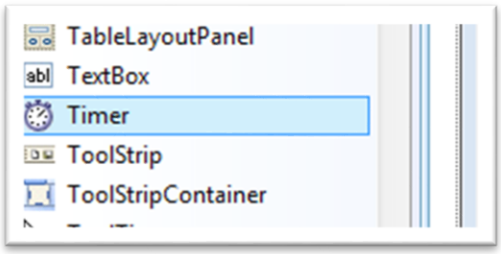
Output:



16. TIMER.

Timer Control plays an important role in the Client side programming and Server side

programming, also used in Windows Services. By using this Timer Control, windows allow you to control when actions take place without the interaction of another thread.

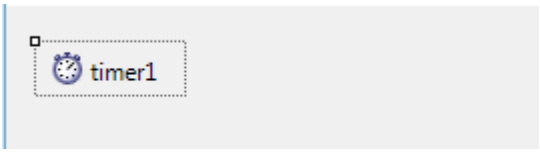


Use of Timer Control

We can use Timer Control in many situations in our development environment. If you want to run some code after a certain interval of time continuously, you can use the Timer control. As well as to start a process at a fixed time schedule, to increase or decrease the speed in an animation graphics with time schedule etc. you can

use the Timer Control. The Visual Studio toolbox has a Timer Control that allowing you to drag and drop the timer controls directly onto a Windows Forms designer. At runtime it does not have a visual representation and works as a

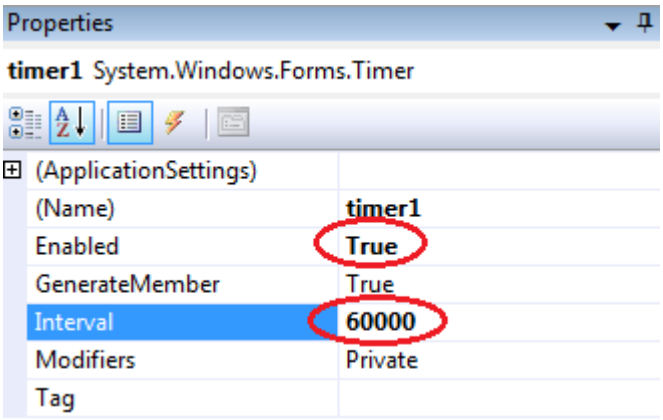
component in the background.



How to Timer Control ?

With the Timer Control, we can control programs in millisecond, seconds, minutes and even in hours. The Timer Control allows us to set Interval property in milliseconds (1 second is equal to 1000 milliseconds). For example, if we want to set an interval of two minute we set the value at Interval property as 120000, means 120x1000 .

The Timer Control starts its functioning only after its Enabled property is set to True, by default Enabled property is False.



Timer example

Here is an example for start and stop methods of the Timer Control.

In this example we run this program only 10 seconds. So we start the Timer in the Form_Load event and stop the Timer after 10 seconds. We set timer Interval property as 1000 milliseconds (1 second) and in run time the Timer will execute 10 times its Tick event .

```
Public Class Form1
Dim second As Integer

Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load

Timer1.Interval = 1000

Timer1.Start() 'Timer starts functioning End Sub

Private Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Timer1.Tick

Label1.Text = DateTime.Now.ToString
```

```
second = second + 1 If second >= 10 Then

Timer1.Stop() 'Timer stops functioning

MsgBox("Timer Stopped....") End If

End Sub End Class
```

MODULE IV

1. MENUS AND TOOLBARS

Menus are a part of every good application and provide not only an easy way to navigate within an application but also useful tools for working with that application. The MenuStrip control in Visual Studio, provides several key features. First and foremost, it provides a quick and easy way to add menus, menu items, and submenu items to your application. It also provides a built- in editor that enables you to add, edit, and delete menu items at the drop of a hat.

The menu items that you create may contain images, access keys, shortcut keys, and check marks as well as text labels.

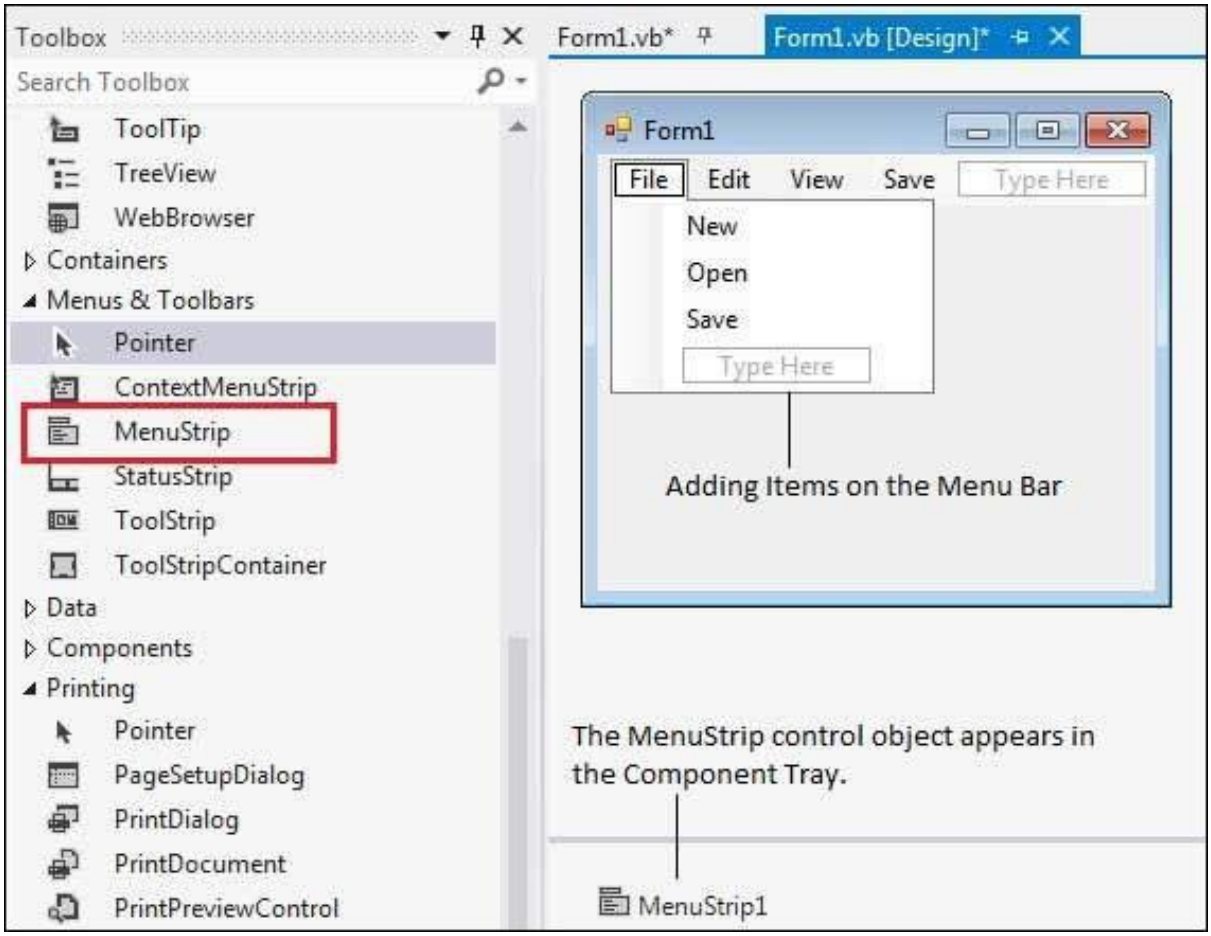
- Visual Basic has come a long way and now provides an Image property for a menu item that makes adding an image to your menu items a breeze.
- An access key (also known as an accelerator key) enables you to navigate the menus using the Alt key and a letter that is underlined in the menu item. When the access key is pressed, the menu appears on the screen, and the user can navigate through it using the arrow keys or the mouse.
- Shortcut keys enable you to invoke the menu item without displaying the menus at all. Shortcut keys usually consist of a control key and a letter, such as Ctrl+X to cut text.
- A check mark symbol can be placed next to a menu item in lieu of an image, typically to indicate that the menu item is being used.
-

1.1 MENU STRIP:

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.

The following diagram shows adding a MenuStrip control on the form –



Properties of the MenuStrip Control

The following are some of the commonly used properties of the MenuStrip control –

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.

Events of the MenuStrip Control

The following are some of the commonly used events of the MenuStrip control –

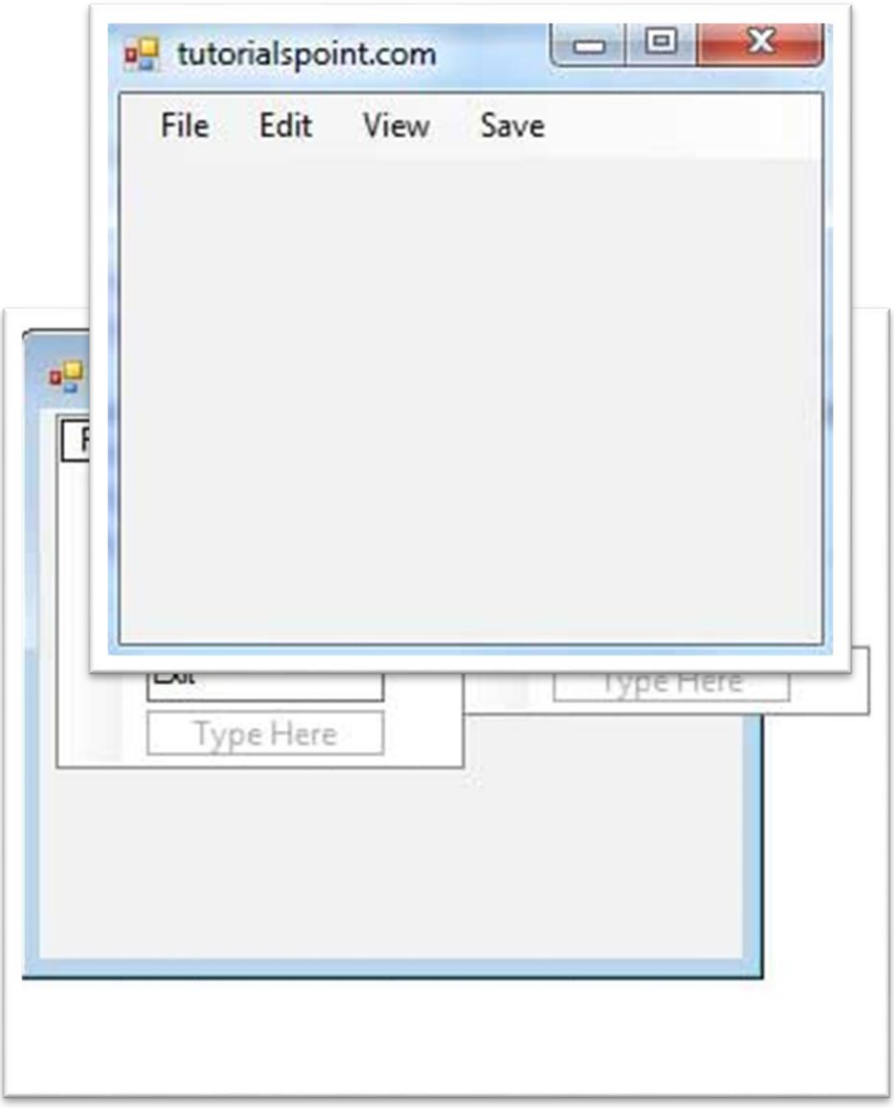
Sr.No.	Event & Description
1	**MenuActivate** Occurs when the user accesses the menu with the keyboard or mouse.
2	**MenuDeactivate** Occurs when the MenuStrip is deactivated.

Eg Program

In this example, let us add menu and sub-menu items.

Take the following steps –

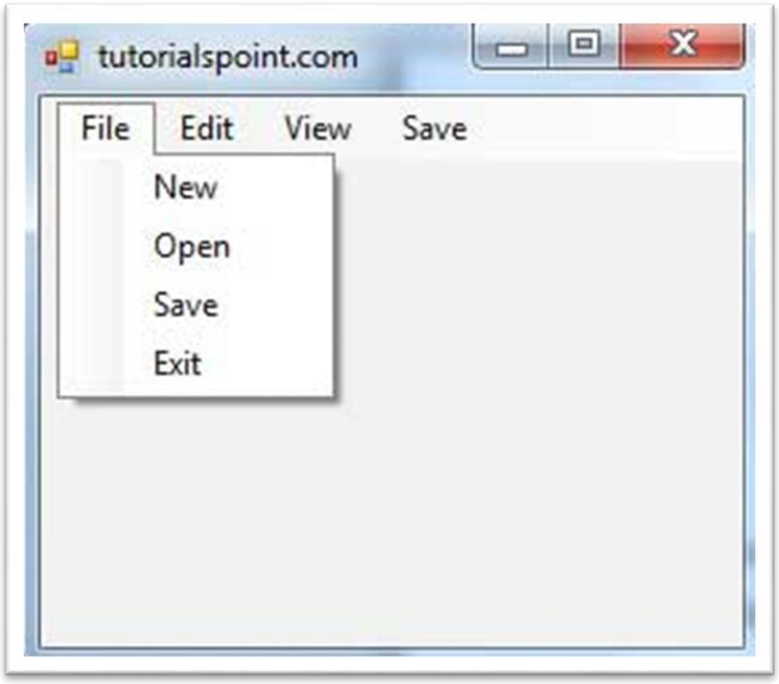
- Drag and drop or double click on a MenuStrip control, to add it to the form.
- Click the Type Here text to open a text box and enter the names of the menu items or sub-menu items you want. When you add a sub-menu, another text box with 'Type Here' text opens below it.
- Complete the menu structure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




```
Private Sub ExitToolStripMenuItem_Click(sender As Object, e As EventArgs) _
  Handles ExitToolStripMenuItem.Click
  End
End Sub
```

Output:

Click on the File -> Exit to exit from the application –



1.2 TOOL STRIP:

The **ToolStripMenuItem** class supports the menus and menu items in

a menu system. You handle these menu items through the click events in a menu system.

Properties of the ToolStripMenuItem Control

The following are some of the commonly used properties of the ToolStripMenuItem control –

Sr.No.	Property & Description
1	**Checked** Gets or sets a value indicating whether the ToolStripMenuItem is checked.
2	**CheckOnClick** Gets or sets a value indicating whether the ToolStripMenuItem should automatically appear checked and unchecked when clicked.
3	**CheckState** Gets or sets a value indicating whether a ToolStripMenuItem is in the checked, unchecked, or indeterminate state.
4	**Enabled** Gets or sets a value indicating whether the control is enabled.
5	**IsMdiWindowListEntry** Gets a value indicating whether the ToolStripMenuItem appears on a multiple document interface (MDI) window list.
6	**ShortcutKeyDisplayString**

	Gets or sets the shortcut key text.
7	**ShortcutKeys** Gets or sets the shortcut keys associated with the ToolStripMenuItem.
8	**ShowShortcutKeys** Gets or sets a value indicating whether the shortcut keys that are associated with the ToolStripMenuItem are displayed next to the ToolStripMenuItem.

Events of the ToolStripMenuItem Control

The following are some of the commonly used events of the ToolStripMenuItem control –

Sr.No.	Event & Description
1	**CheckedChanged** Occurs when the value of the Checked property changes.
2	**CheckStateChanged** Occurs when the value of the CheckState property changes.

Eg Program

In this example, let us continue with the example from the chapter 'VB.Net - MenuStrip control'. Let us –

❖ Hide and display menu items.

- Disable and enable menu items.
- Set access keys for menu items
- Set shortcut keys for menu items.

Hide and Display Menu Items

The **Visible** property of the **ToolStripMenuItem** class allows you to hide or show a menu item. Let us hide the Project Menu on the menu bar.

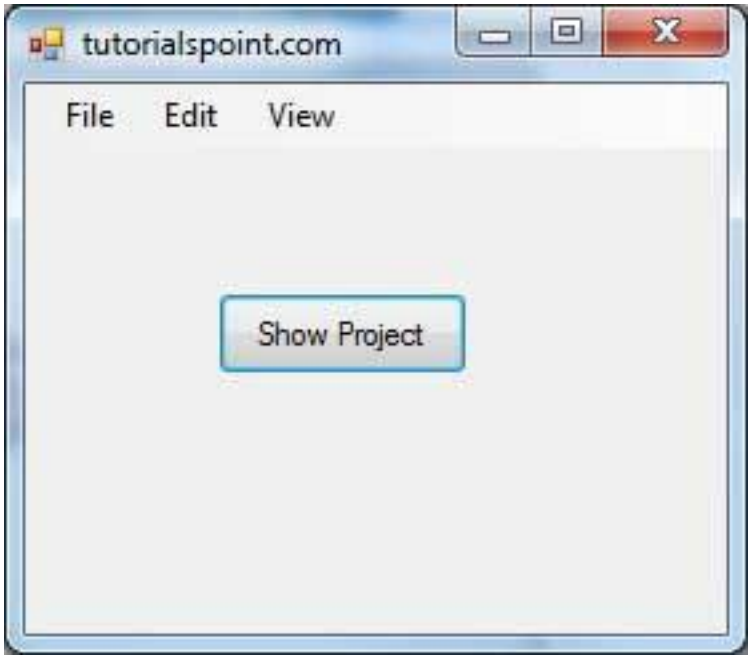
- Add the following code snippet to the Form1_Load event –

```
Private Sub Form1_Load(sender As Object, e As EventArgs) _
Handles MyBase.Load
  ' Hide the project menu
  ProjectToolStripMenuItem1.Visible = False
  ' Set the caption bar text of the form.
  Me.Text = "tutorialspoint.com"
End Sub
```

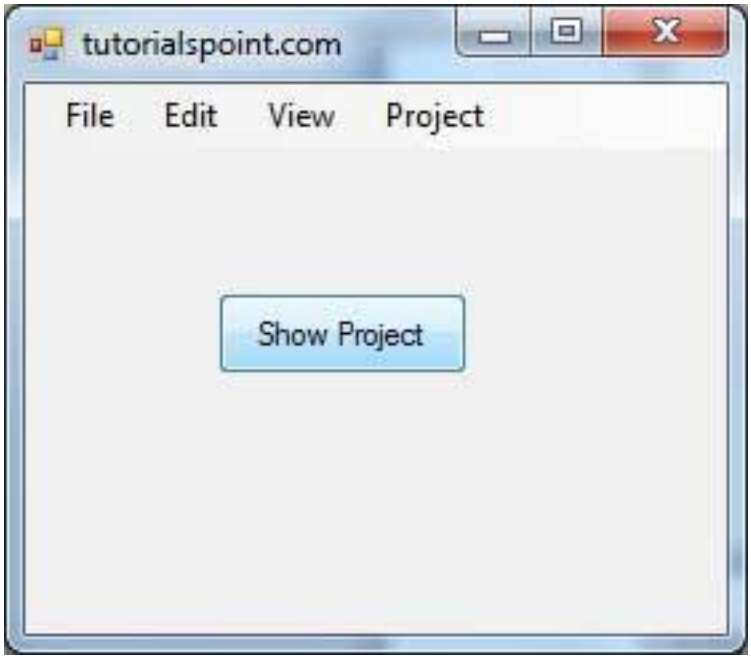
- Add a button control on the form with text 'Show Project'.
- Add the following code snippet to the Button1_Click event –

```
Private Sub Button1_Click(sender As Object, e As EventArgs) _
Handles Button1.Click
  ProjectToolStripMenuItem1.Visible = True
End Sub
```

When the above code is executed and run using **Start** button available at the Microsoft Visual Studio tool bar, it will show the following window:



Clicking on the Show Project button displays the project menu –



Disable and Enable Menu Items

The **Enabled** property allows you to disable or gray out a menu item. Let us disable the Project Menu on the menu bar.

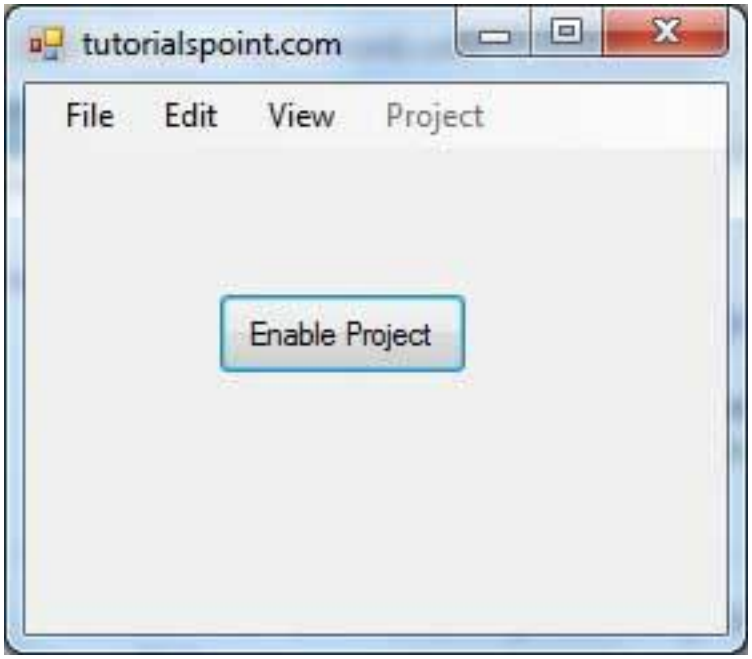
❖ Add the following code snippet to the Form1_Load event –

```
Private Sub Form1_Load(sender As Object, e As EventArgs) _
Handles MyBase.Load
  ' Disable the project menu
  ProjectToolStripMenuItem1.Enabled = False
  ' Set the caption bar text of the form.
  Me.Text = "tutorialspoint.com"
End Sub
```

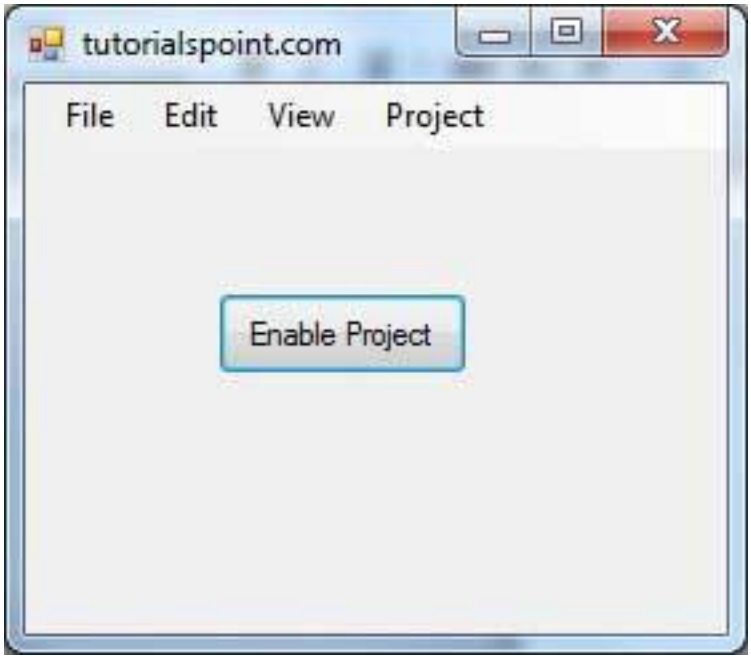
❖ Add a button control on the form with text 'Enable Project'.
❖ Add the following code snippet to the Button1_Click event –

```
Private Sub Button1_Click(sender As Object, e As EventArgs) _
Handles Button1.Click
  ProjectToolStripMenuItem1.Enabled = True
End Sub
```

When the above code is executed and run using **Start** button available at the Microsoft Visual Studio tool bar, it will show the following window –



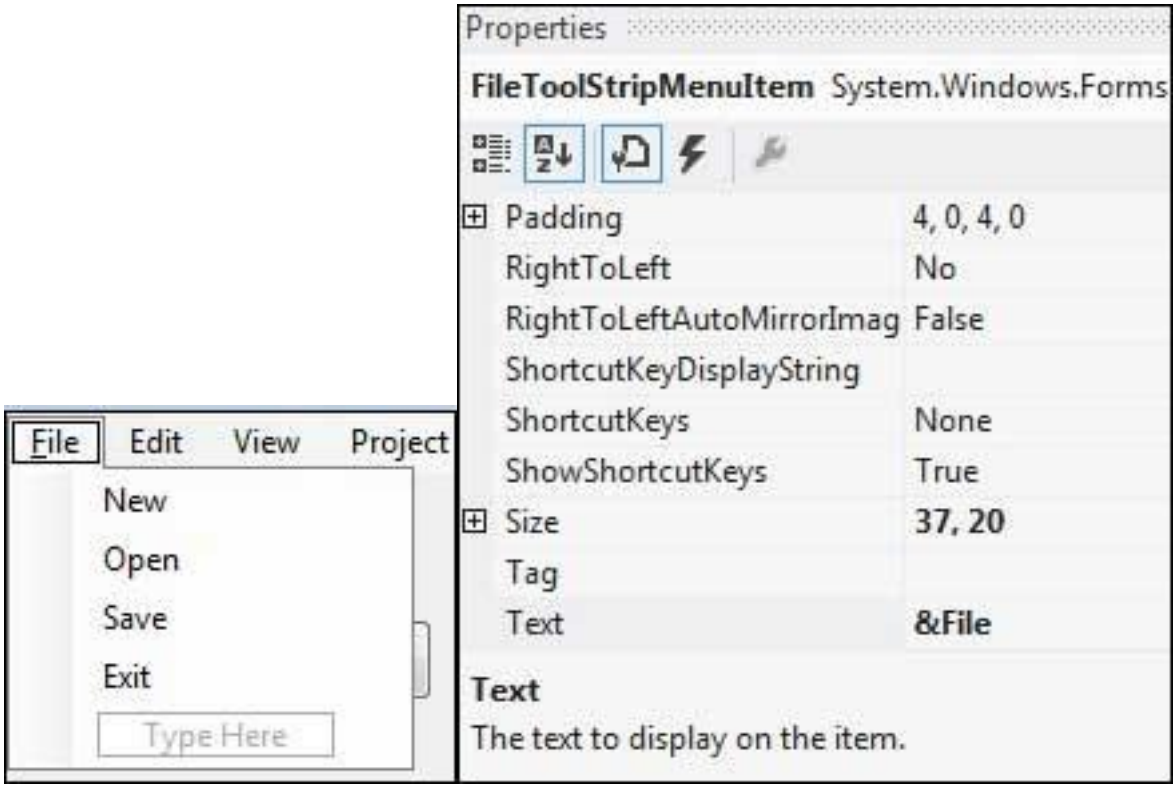
Clicking on the Enable Project button enables the project menu –



Set Access Keys for Menu Items

Setting access keys for a menu allows a user to select it from the keyboard by using the ALT key.

For example, if you want to set an access key ALT + F for the file menu, change its **Text** with an added & (ampersand) preceding the access key letter. In other words, you change the text property of the file menu to &File.

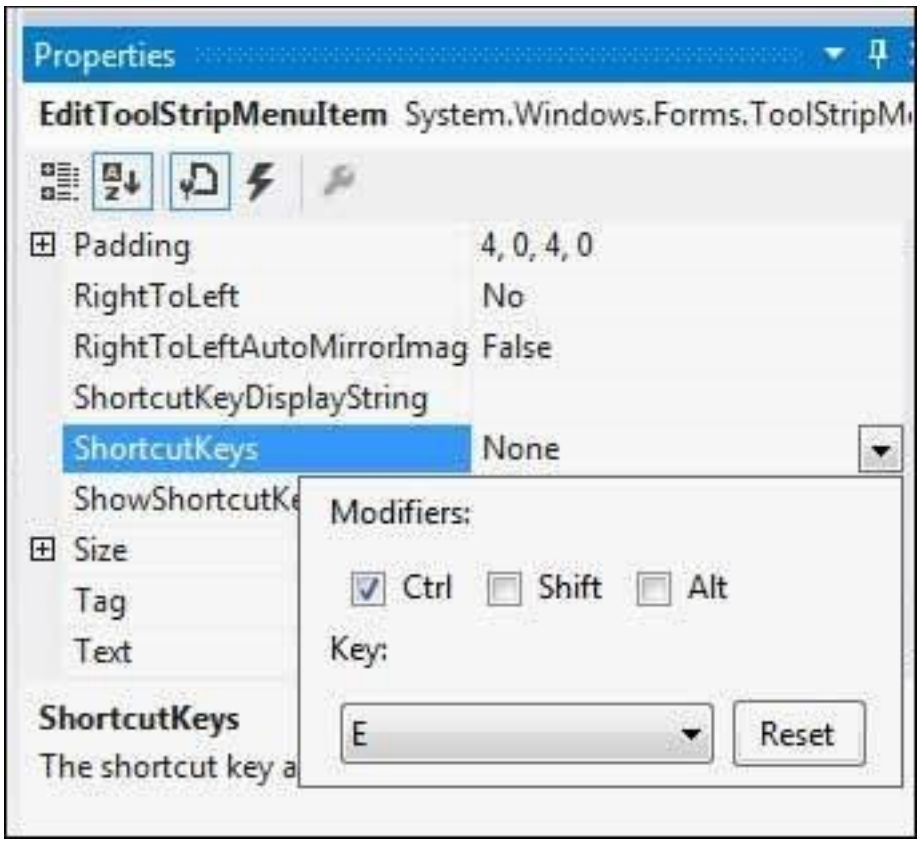


Set Shortcut Keys for Menu Items

When you set a shortcut key for a menu item, user can press the shortcut from the keyboard and it would result in occurrence of the Click event of the menu.

A shortcut key is set for a menu item using the ShortcutKeys property. For example, to set a shortcut key CTRL + E, for the Edit menu –

- Select the Edit menu item and select its ShortcutKeys property in the properties window.
- Click the drop down button next to it.
- Select Ctrl as Modifier and E as the key.



1.3 STATUS STRIP

A StatusStrip control displays information about an object being viewed on a Form, the object's components, or contextual information that relates to that object's operation within your application. Typically, a StatusStrip control consists of ToolStripStatusLabel objects, each of which displays text, an icon, or both.

The StatusStrip can also contain ToolStripDropDownButton, ToolStripSplitButton, and ToolStripProgressBar controls.

The default StatusStrip has no panels. To add panels to a StatusStrip, use the ToolStripItemCollection.AddRange method.

There is extensive support for handling StatusStrip items and common commands in Visual Studio.

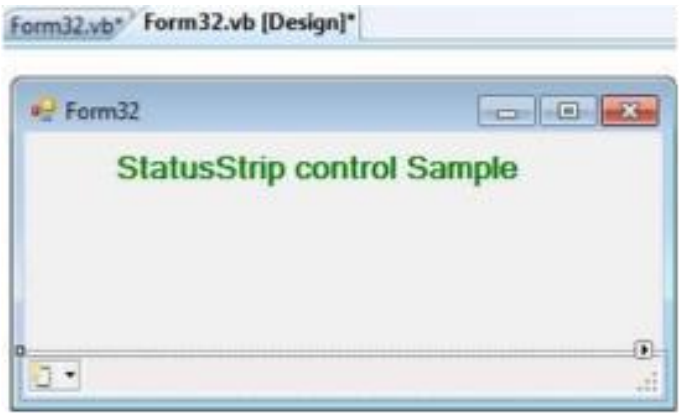
Although StatusStrip replaces and extends the StatusBar control of previous versions, StatusBar is retained for both backward compatibility and future use if you choose.

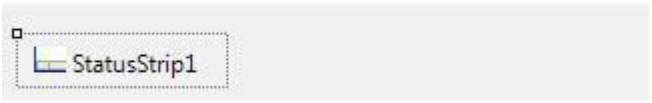
Important StatusStrip Members

NAME	DESCRIPTION
ToolStripStatusLabel	Represents a panel in a StatusStrip control.
	Displays an associated ToolStripDropDown from which the user can select a single item.
ToolStripSplitButton	Represents a two-part control that is a standard button and a drop-down menu.
ToolStripProgressBar	Displays the completion state of a process

How to use StatusStrip

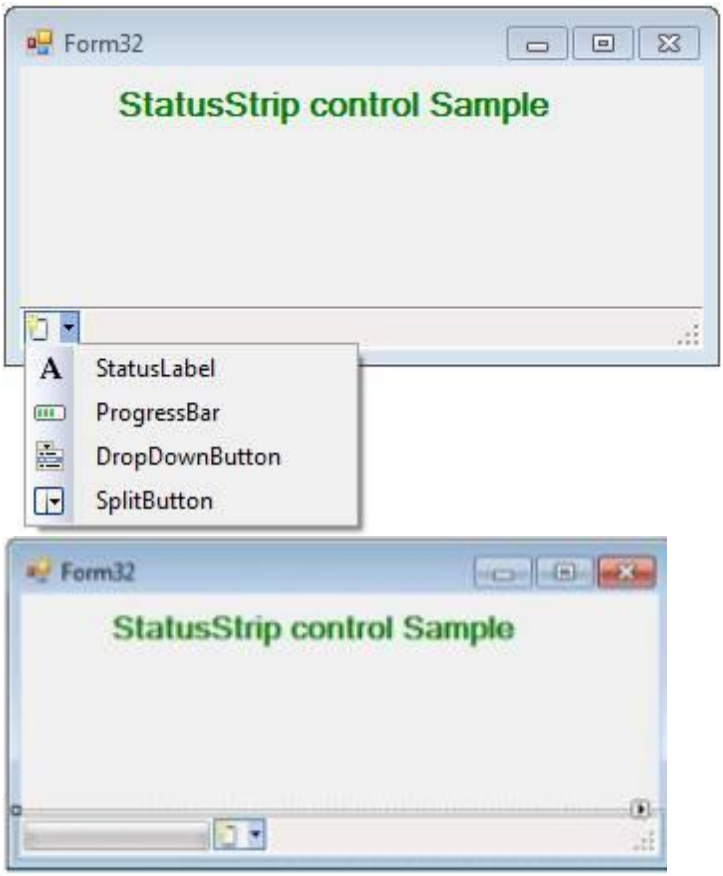
Drag and drop StatusStrip control from the toolbox on the window Form.





Select-control which you want to show in StatusStrip

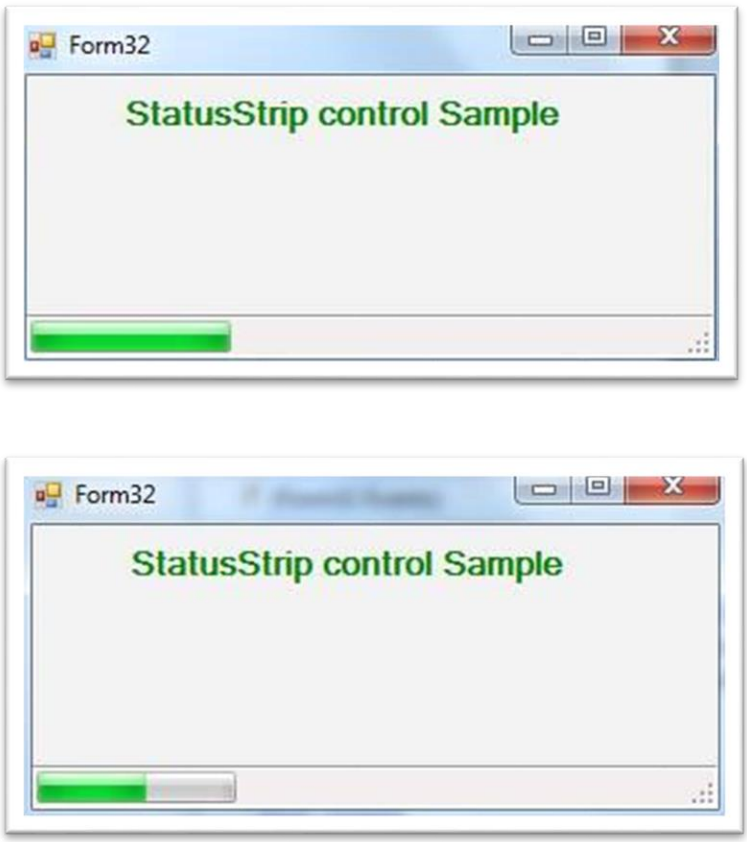
To add Status Label, ProgressBar, DropdownButton, and SplitButton Here ProgressBar is added.



CODE:

```
Private Sub Form32_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handl es MyBase.Load

While ToolStripProgressBar1.Value < 100 ToolStripProgressBar1.Value += 5

End While
```

Output:



2. BUILT-IN DIALOG BOXES

There are a number of built-in dialog boxes in Visual Basic, which is great, because developing your own file open, file save, and other dialog boxes not only takes a lot of work, but gives your program a different look from what Windows users are already used to.

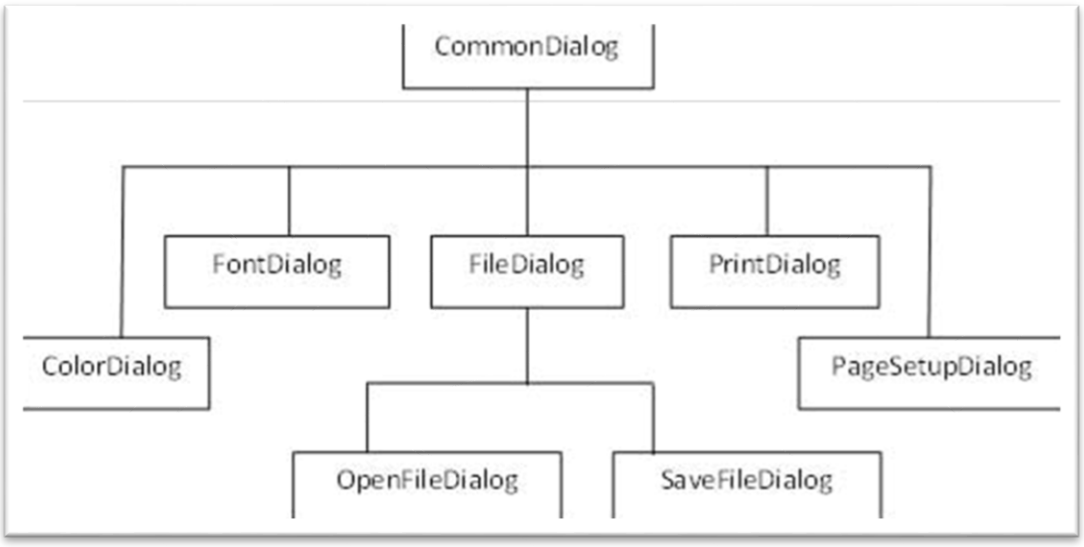
All of these dialog box control classes inherit from the **CommonDialog** class and override the **RunDialog()** function of the base class to create the specific dialog box.

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:

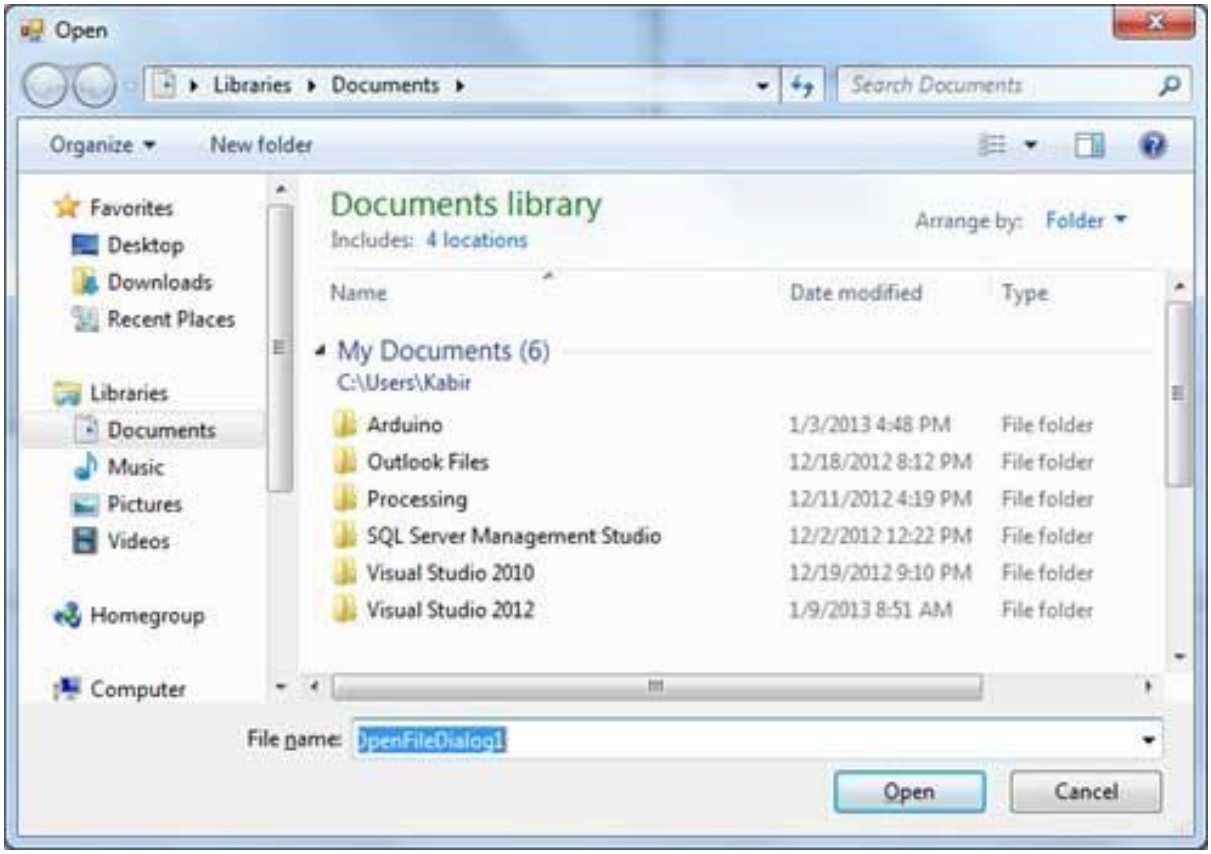


2.1 OPEN FILE DIALOGS

The **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.
7	**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).
8	**FileName** 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.
10	**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.
11	**FilterIndex** 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.
13	**Multiselect** Gets or sets a value indicating whether the dialog box allows multiple files to be selected.
14	**ReadOnlyChecked** Gets or sets a value indicating whether the read-only check box is selected.
15	**RestoreDirectory** Gets or sets a value indicating whether the dialog box restores the current directory before closing.
16	**SafeFileName** Gets the file name and extension for the file selected in the dialog box. The file name does not include the path.
17	**SafeFileNames** 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.

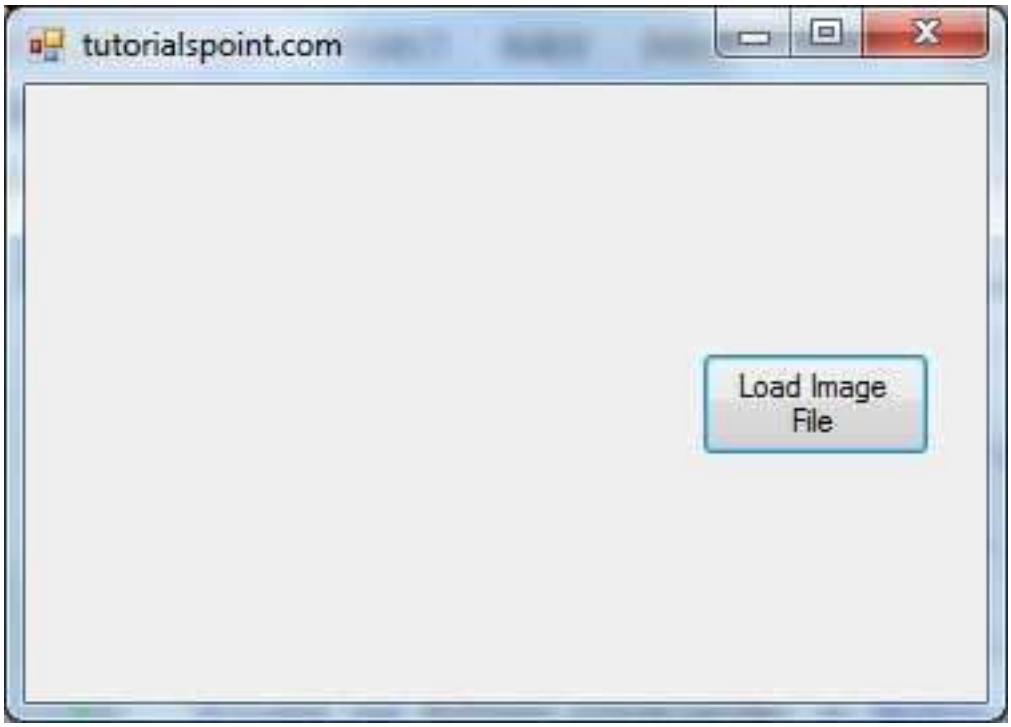
Eg Program

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

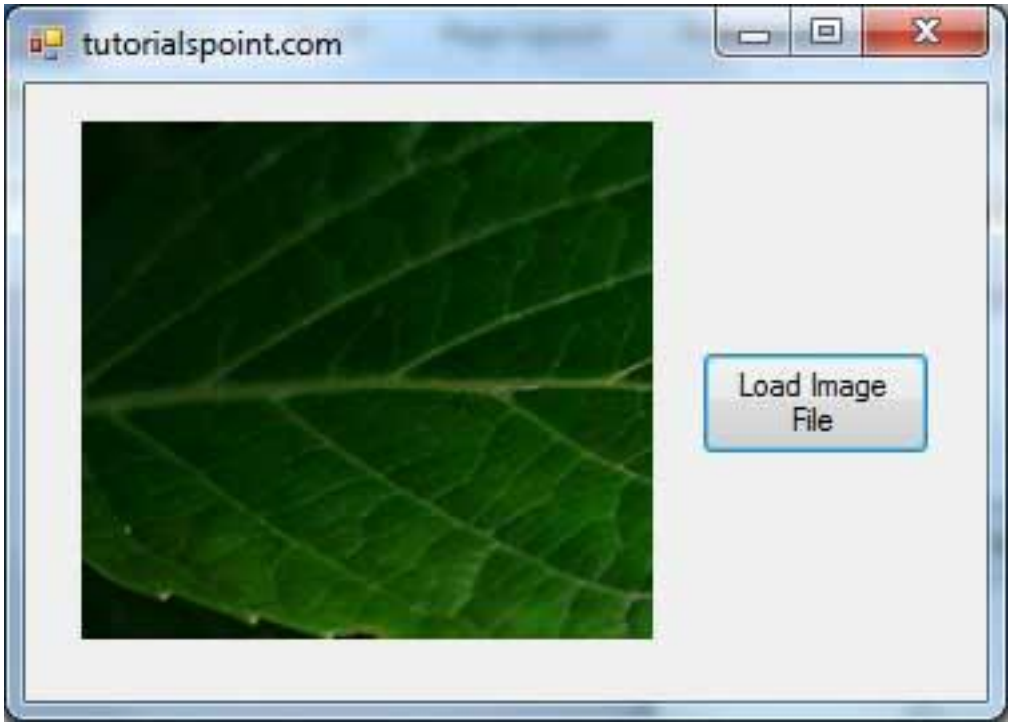
- Drag and drop a PictureBox control, a Button control and a OpenFileDialog control on the form.
- Set the Text property of the button control to 'Load Image File'.
- Double-click the Load Image File button and modify the code of the Click event:

```
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 –



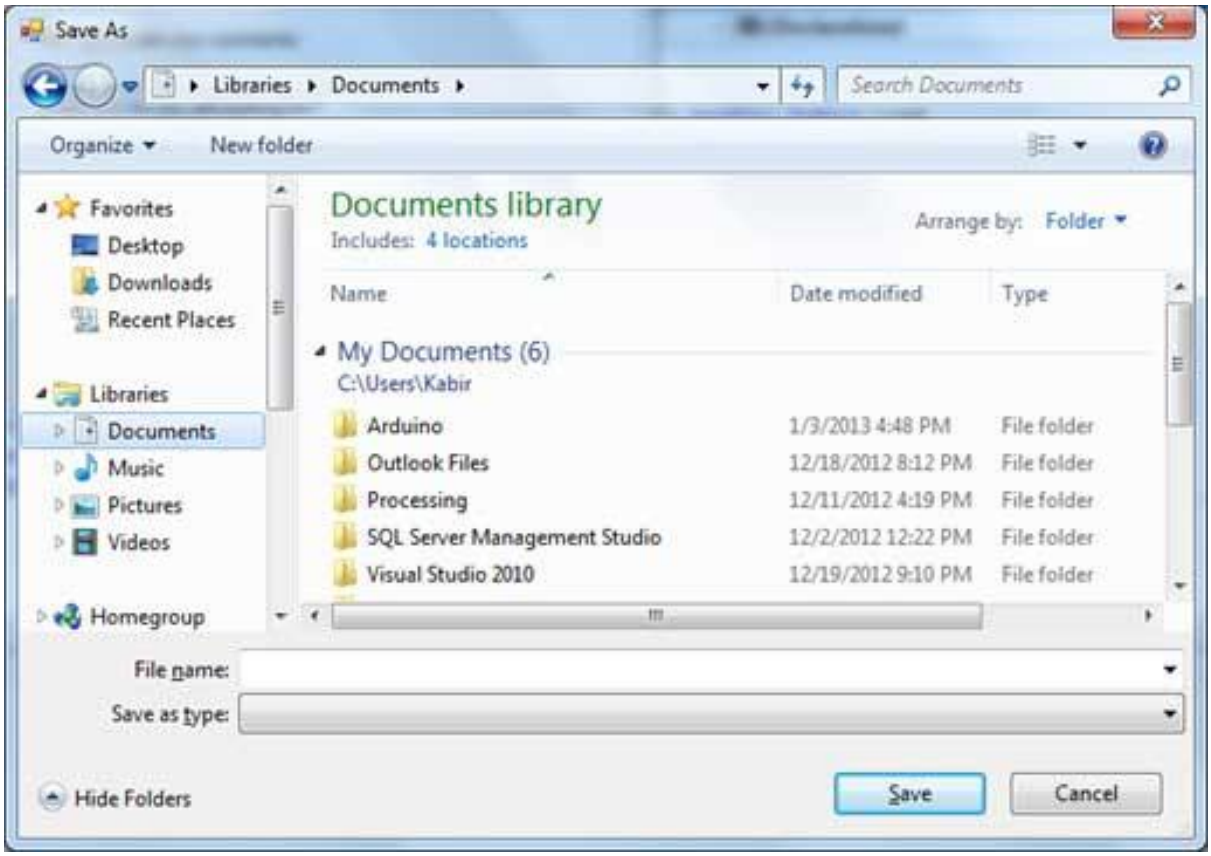
Click on the Load Image File button to load an image stored in your computer.



2.2 SAVE FILE DIALOGS

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.

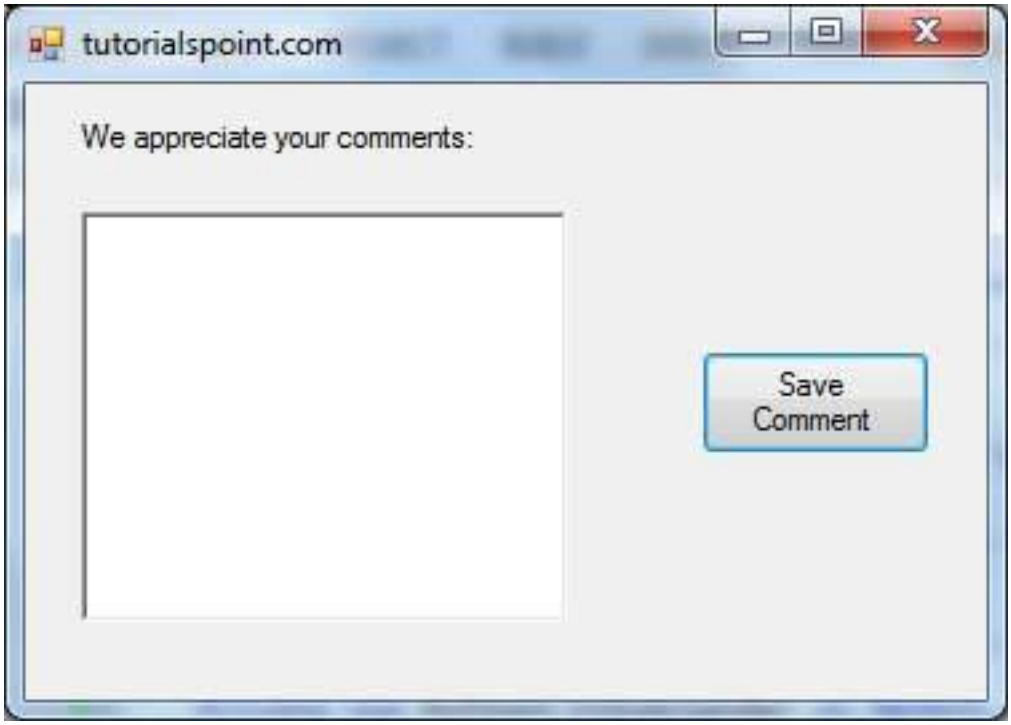
Eg Program

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 –



2.3 FONT DIALOGS

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.

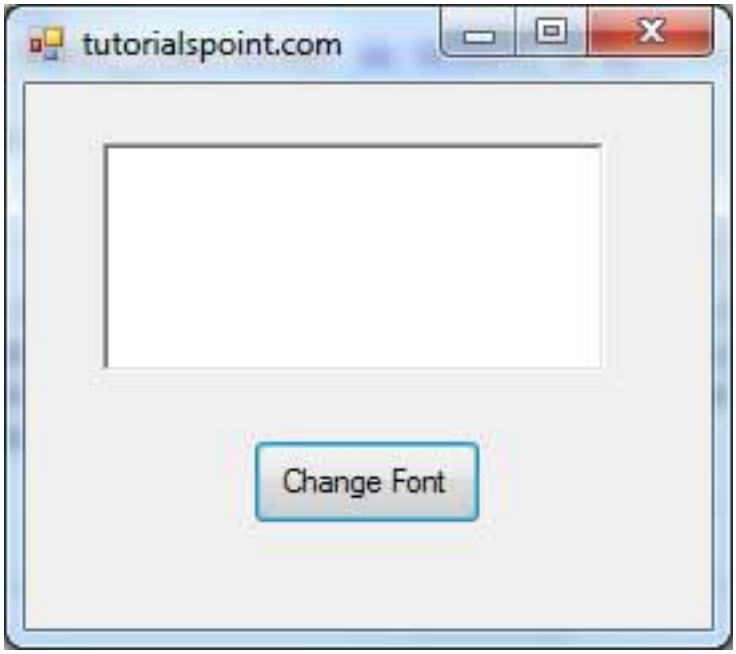
Example

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 –

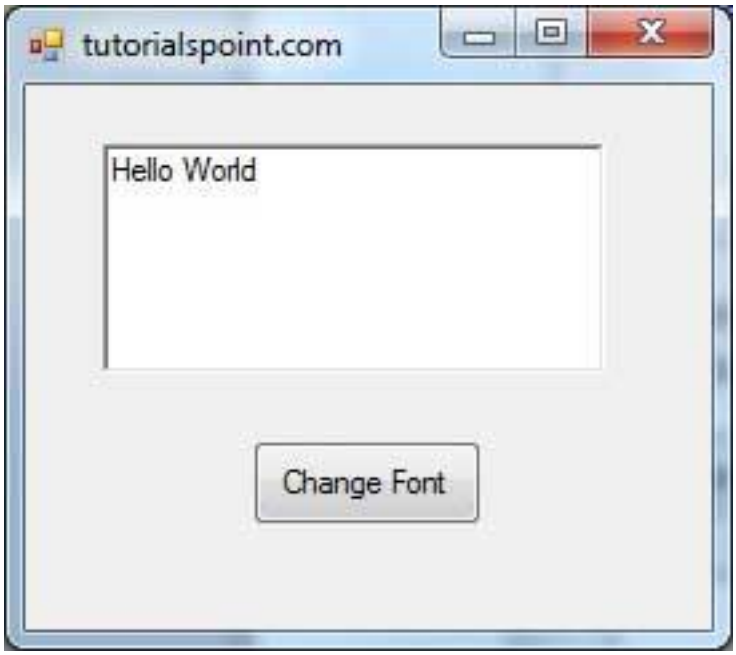
- Drag and drop a RichTextBox control, a Button control and a FontDialog control on the form.
- Set the Text property of the button control to 'Change Font'.
- Set the ShowColor property of the FontDialog control to True.
- Double-click the Change Color button and modify the code of the Click event –

```
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
```

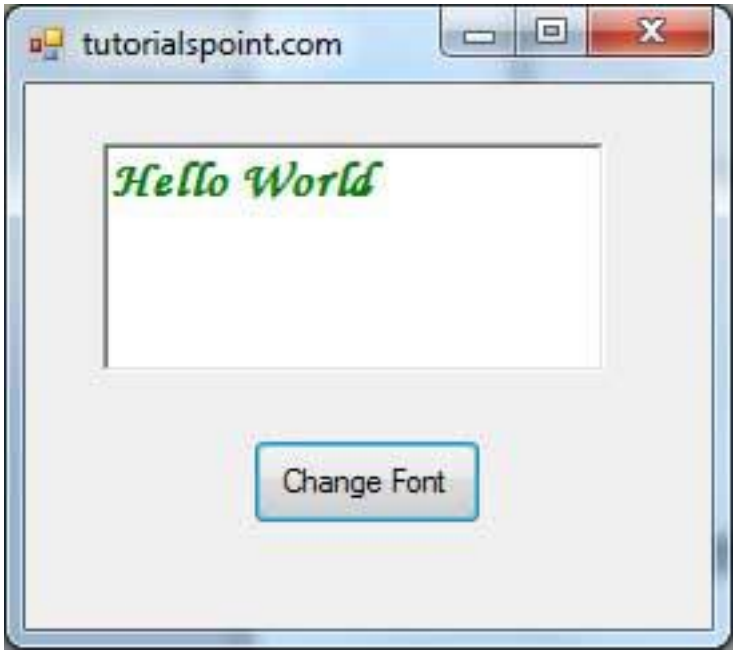
When the application is compiled and run using **Start** button available at the Microsoft Visual Studio tool bar, it will show the following window –



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.

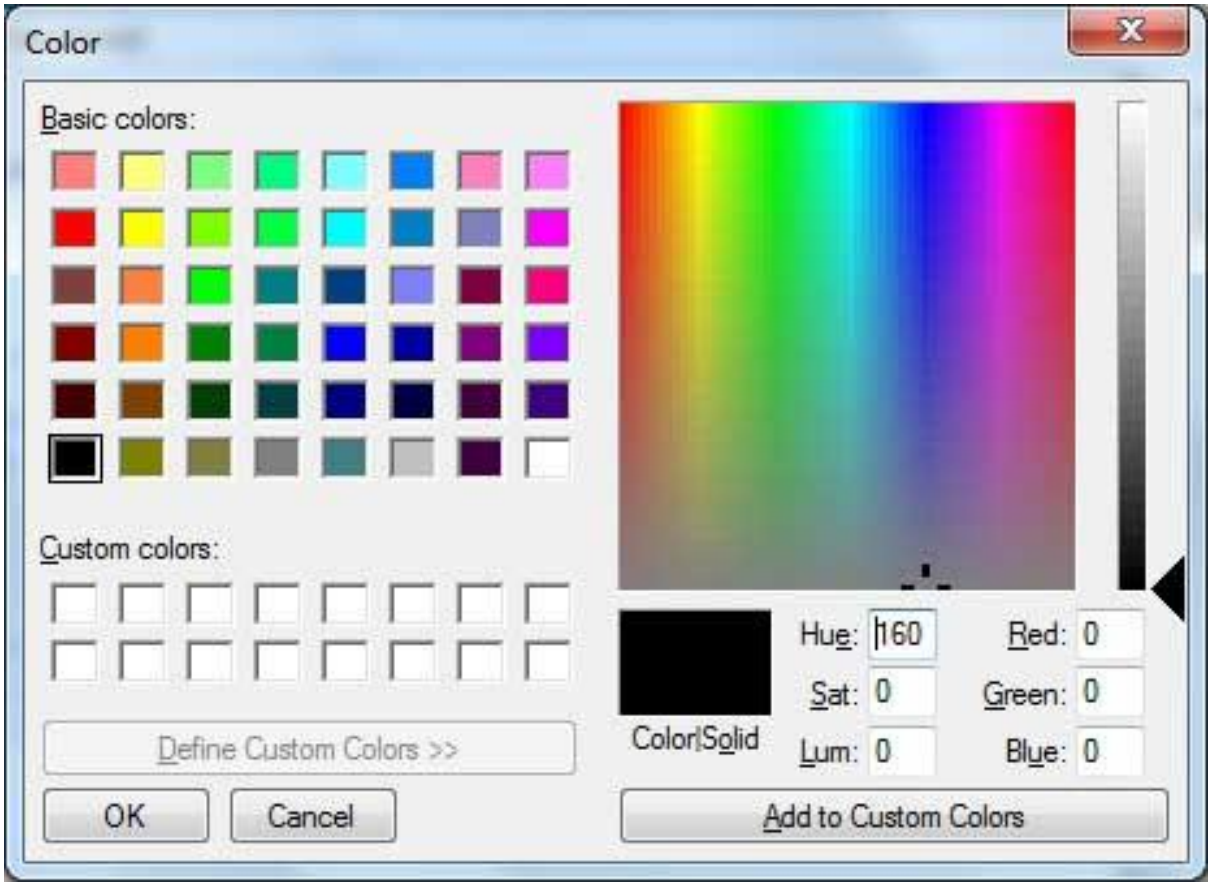


2.4 COLOR DIALOGS

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.

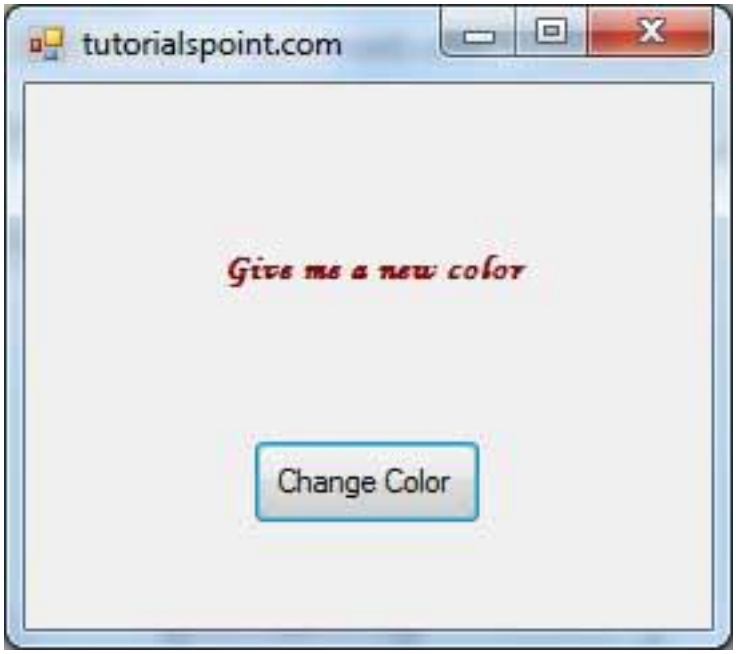
Eg Program

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 –



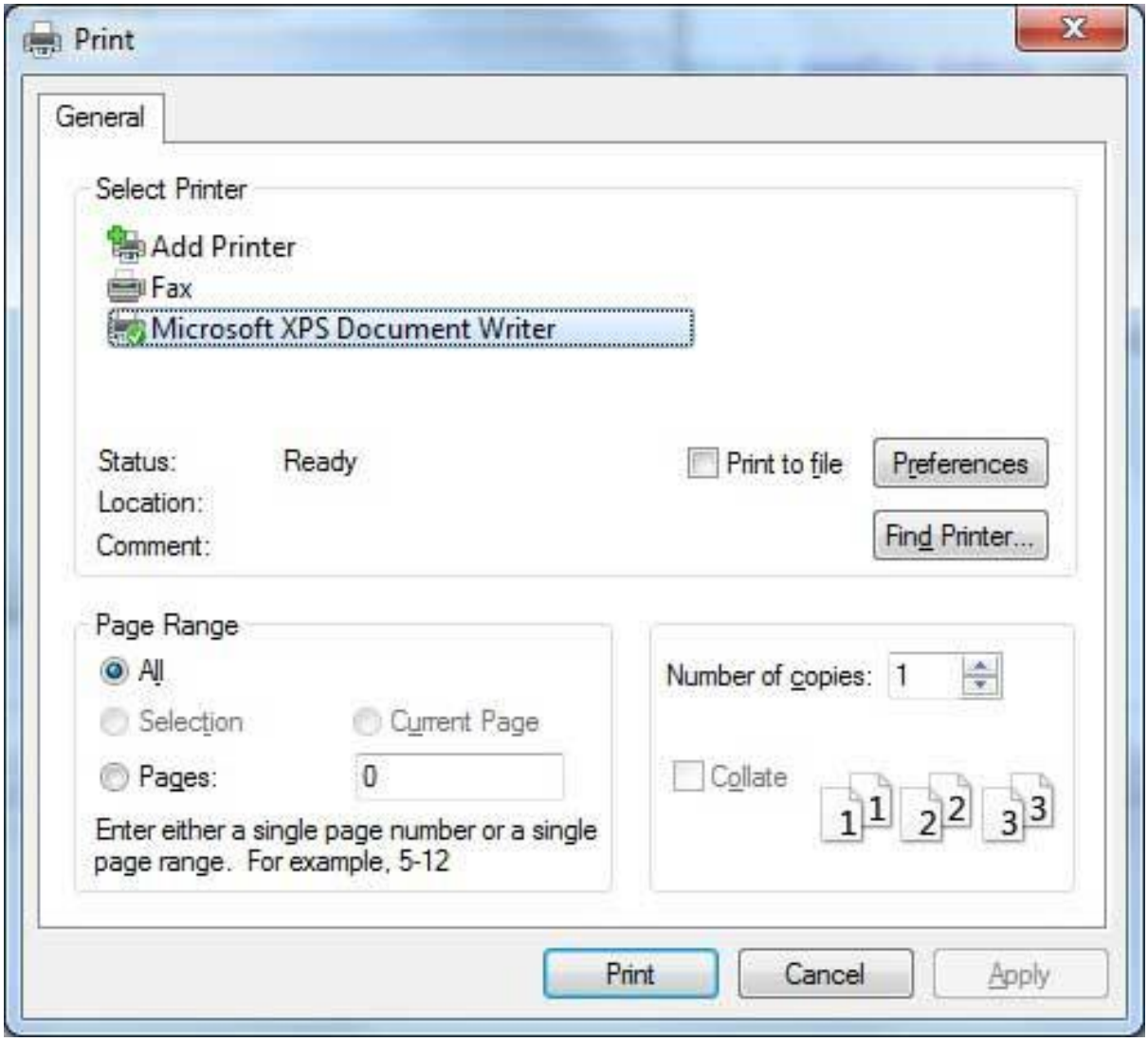
2.5 PRINT DIALOGS

The PrintDialog control lets the user to print documents by selecting a printer and choosing which sections of the document to print from a Windows Forms application.

There are various other controls related to printing of documents. Let us have a brief look at these controls and their purpose. These other controls are –

- The **PrintDocument** control – it provides support for actual events and operations of printing in Visual Basic and sets the properties for printing.
- The **PrinterSettings** control – it is used to configure how a document is printed by specifying the printer.
- The **PageSetUpDialog** control – it allows the user to specify page-related print settings including page orientation, paper size and margin size.
- The **PrintPreviewControl** control – it represents the raw preview part of print previewing from a Windows Forms application, without any dialog boxes or buttons.
- The **PrintPreviewDialog** control – it represents a dialog box form that contains a PrintPreviewControl for printing from a Windows Forms application.
-

Following is the Print dialog box –



Properties of the PrintDialog Control

The following are some of the commonly used properties of the PrintDialog control –

Sr.No.	Property & Description
1	**AllowCurrentPage** Gets or sets a value indicating whether the **Current Page** option button is displayed.
2	**AllowPrintToFile**

	Gets or sets a value indicating whether the **Print to file** check box is enabled.
3	**AllowSelection** Gets or sets a value indicating whether the **Selection** option button is enabled.
4	**AllowSomePages** Gets or sets a value indicating whether the **Pages** option button is enabled.
5	**Document** Gets or sets a value indicating the PrintDocument used to obtain PrinterSettings.
6	**PrinterSettings** Gets or sets the printer settings the dialog box modifies.
7	**PrintToFile** Gets or sets a value indicating whether the **Print to file** check box is selected.
8	**ShowHelp** Gets or sets a value indicating whether the **Help** button is displayed.
9	**ShowNetwork** Gets or sets a value indicating whether the **Network** button is displayed.

Methods of the PrintDialog Control

The following are some of the commonly used methods of the PrintDialog 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.

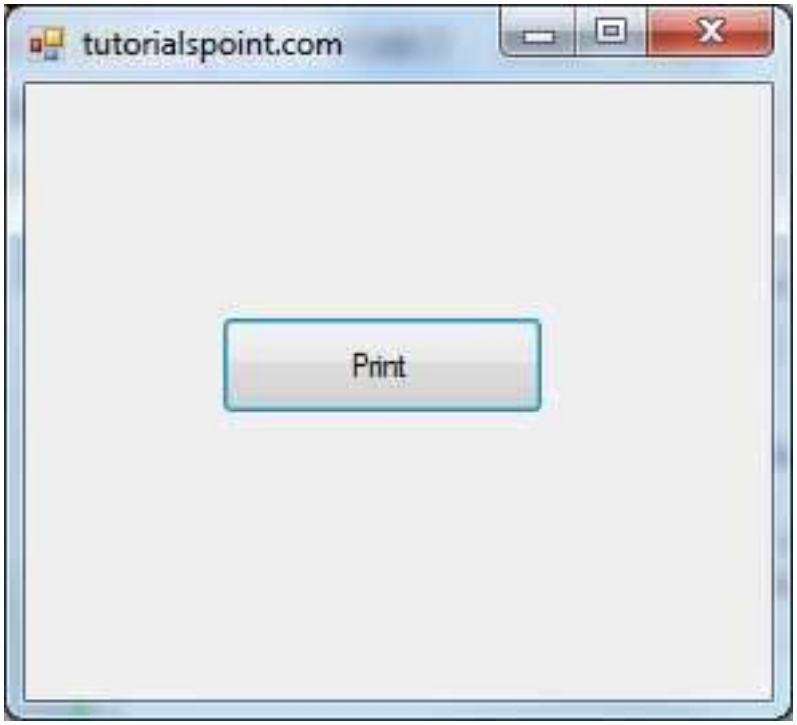
Example

In this example, let us see how to show a Print dialog box in a form. Take the following steps –

- Add a PrintDocument control, a PrintDialog control and a Button control on the form. The PrintDocument and the PrintDialog controls are found on the Print category of the controls toolbox.
- Change the text of the button to 'Print'.
- Double-click the Print button and modify the code of the Click event as shown –

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
  PrintDialog1.Document = PrintDocument1
  PrintDialog1.PrinterSettings = PrintDocument1.PrinterSettings
  PrintDialog1.AllowSomePages = True

  If PrintDialog1.ShowDialog = DialogResult.OK Then
    PrintDocument1.PrinterSettings = PrintDialog1.PrinterSettings
    PrintDocument1.Print()
  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 –



3. INPUTBOX

The **InputBox function** prompts the users to enter values. After entering the values, if the user clicks the OK button or presses ENTER on the keyboard, the InputBox function will return the text in the text box. If the user clicks the Cancel button, the function will return an empty string ("").

Syntax

```
InputBox(prompt[,title][,default][,xpos][,ypos][,helpfile,context])
```

Parameter Description

- **Prompt** – A required parameter. A String that is displayed as a message in the dialog box. The maximum length of prompt is approximately 1024 characters. If the message extends to more than a line, then the lines can be separated using a carriage return character (Chr(13)) or a linefeed character (Chr(10)) between each line.

- **Title** – An optional parameter. A String expression displayed in the title bar of the dialog box. If the title is left blank, the application name is placed in the title bar.

- **Default** – An optional parameter. A default text in the text box that the user would like to be displayed.

- **XPos** – An optional parameter. The position of **X** axis represents the prompt distance from the left side of the screen horizontally. If left blank, the input box is horizontally centered.

- **YPos** – An optional parameter. The position of **Y** axis represents the prompt distance from the left side of the screen vertically. If left blank, the input box is vertically centered.

- **Helpfile** – An optional parameter. A String expression that identifies the helpfile to be used to provide context-sensitive Help for the dialog box.

- **context** – An optional parameter. A Numeric expression that identifies the Help context number assigned by the Help author to the appropriate Help topic. If context is provided, helpfile must also be provided.

Example

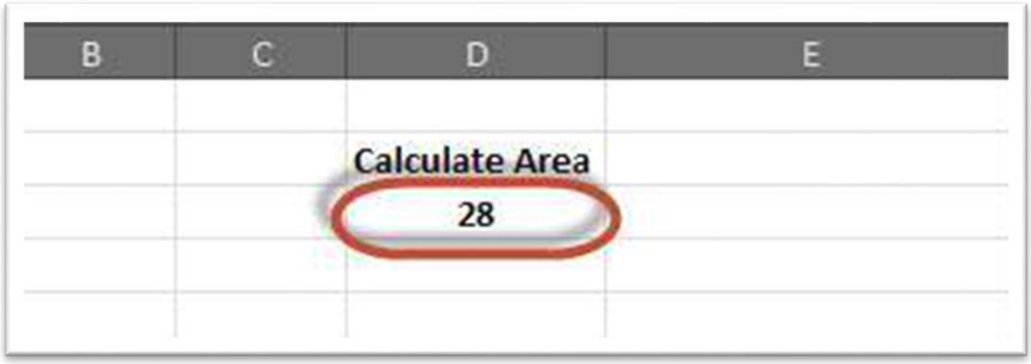
Let us calculate the area of a rectangle by getting values from the user at run time with the help of two input boxes (one for length and one for width).

```
Function findArea()
   Dim Length As Double
   Dim Width As Double

   Length = InputBox("Enter Length ", "Enter a Number")
   Width = InputBox("Enter Width", "Enter a Number")
   findArea = Length * Width
End Function
```

Output


Enter Value
Enter Length
OK
Cancel
7


Enter Value
Enter Width
OK
Cancel
4


B
C
D
E
Calculate Area
28

4.

MSG BOX

The **MsgBox function** displays a message box and waits for the user to click a button and then an action is performed based on the button clicked by the user.

Syntax

```
MsgBox(prompt[,buttons][,title][,helpfile,context])
```

Parameter Description

- **Prompt** – A Required Parameter. A String that is displayed as a message in the dialog box. The maximum length of prompt is approximately 1024 characters. If the message extends to more than a line, then the lines can be separated using a carriage return character (Chr(13)) or a linefeed character (Chr(10)) between each line.

- **Buttons** – An Optional Parameter. A Numeric expression that specifies the type of buttons to display, the icon style to use, the identity of the default button, and the modality of the message box. If left blank, the default value for buttons is 0.

- **Title** – An Optional Parameter. A String expression displayed in the title bar of the dialog box. If the title is left blank, the application name is placed in the title bar.

- **Helpfile** – An Optional Parameter. A String expression that identifies the Help file to use for providing context-sensitive help for the dialog box.

- **Context** – An Optional Parameter. A Numeric expression that identifies the Help context number assigned by the Help author to the appropriate Help topic. If context is provided, helpfile must also be provided.

The Buttons parameter can take any of the following values –

- 0 vbOKOnly - Displays OK button only.
- 1 vbOKCancel - Displays OK and Cancel buttons.
- 2 vbAbortRetryIgnore - Displays Abort, Retry, and Ignore buttons.
- 3 vbYesNoCancel - Displays Yes, No, and Cancel buttons.
- 4 vbYesNo - Displays Yes and No buttons.

- 5 vbRetryCancel - Displays Retry and Cancel buttons.
- 16 vbCritical - Displays Critical Message icon.
- 32 vbQuestion - Displays Warning Query icon.
- 48 vbExclamation - Displays Warning Message icon.
- 64 vbInformation - Displays Information Message icon.
- 0 vbDefaultButton1 - First button is default.
- 256 vbDefaultButton2 - Second button is default.
- 512 vbDefaultButton3 - Third button is default.
- 768 vbDefaultButton4 - Fourth button is default.
- 0 vbApplicationModal Application modal - The current application will not work until the user responds to the message box.
- 4096 vbSystemModal System modal - All applications will not work until the user responds to the message box.

The above values are logically divided into four groups: The **first group** (0 to 5) indicates the buttons to be displayed in the message box. The **second group** (16, 32, 48, 64) describes the style of the icon to be displayed, the **third group** (0, 256, 512, 768) indicates which button must be the default, and the **fourth group** (0, 4096) determines the modality of the message box.

Return Values

The MsgBox function can return one of the following values which can be used to identify the button the user has clicked in the message box.

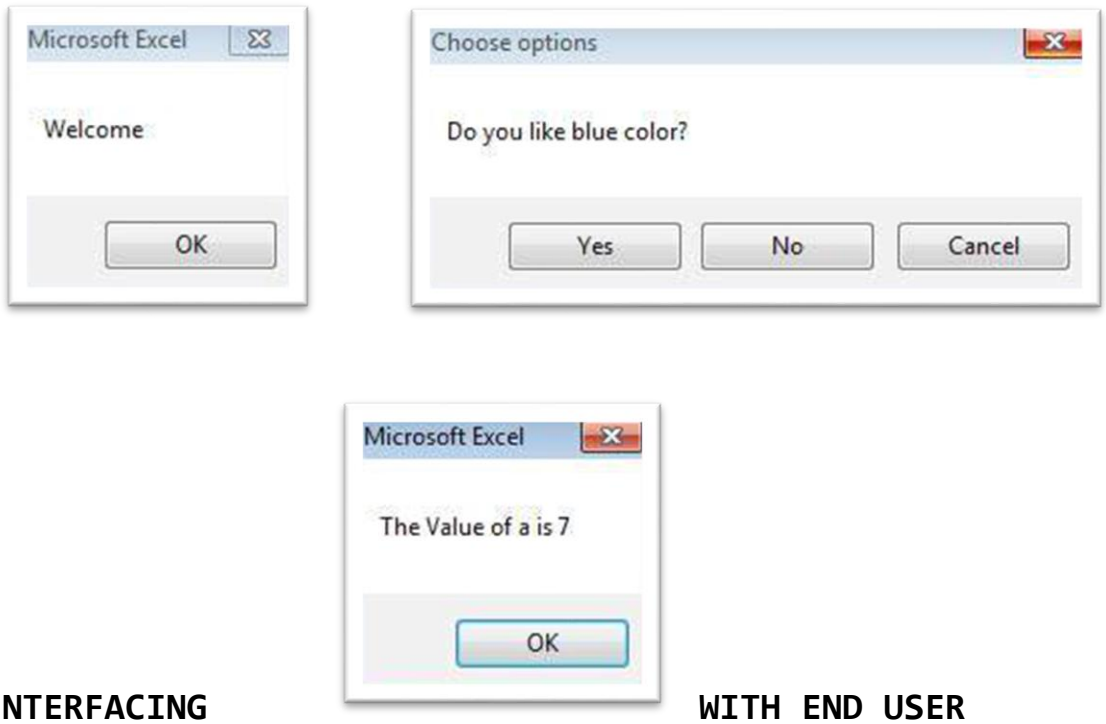
- 1 - vbOK - OK was clicked
- 2 - vbCancel - Cancel was clicked
- 3 - vbAbort - Abort was clicked
- 4 - vbRetry - Retry was clicked
- 5 - vbIgnore - Ignore was clicked
- 6 - vbYes - Yes was clicked
- 7 - vbNo - No was clicked

Eg Program

```
Function MessageBox_Demo()
  'Message Box with just prompt message
  MsgBox("Welcome")

  'Message Box with title, yes no and cancel Butttons
  int a = MsgBox("Do you like blue color?",3,"Choose options")
```

```
  ' Assume that you press No Button
  msgbox ("The Value of a is " & a)
End Function
```

Output:



5. INTERFACING WITH END USER

Multiple-document interface (MDI) applications enable you to display multiple documents at the

same time, with each document displayed in its own window. MDI applications often have a

Window menu item with submenus for switching between windows or documents.

5.1 CREATING MDI PARENT AND CHILD.

How to: Create MDI Parent Forms

The foundation of a Multiple-Document Interface (MDI) application is the MDI parent form. This is the form that contains the MDI child windows, which are the sub-windows wherein the user interacts with the MDI application. Creating an MDI parent form is easy, both in the Windows Forms Designer and programmatically.

Create an MDI parent form at design time

1. Create a Windows Application project in Visual Studio.
2. In the Properties window, set the IsMdiContainer property to true.

This designates the form as an MDI container for child windows.

1. From the Toolbox, drag a MenuStrip control to the form. Create a top-level menu item with
2. the Text property set to &File with submenu items called &New and &Close. Also create a top-level menu item called &Window.
3. The first menu will create and hide menu items at run time, and the second menu will keep track of the open MDI child windows. At this point, you have created an MDI parent window.
4. Press F5 to run the application. For information about creating MDI child windows that operate within the MDI parent form, see How to: Create MDI Child Forms.

How to: Create MDI child forms

MDI child forms are an essential element of Multiple-Document Interface (MDI) applications, as these forms are the center of user interaction.

Create MDI child forms

1. Create a new Windows Forms application project in Visual Studio. In the Properties window for the form, set its IsMdiContainer property to true and its WindowsState property to Maximized.

This designates the form as an MDI container for child windows.

2. From the Toolbox, drag a MenuStrip control to the form. Set its Text property to File.

3. Click the ellipsis (...) next to the Items property, and click Add to add two child tool strip menu items. Set the Text property for these items to New and Window.

4. In Solution Explorer, right-click the project, and then select Add > New Item.

5. In the Add New Item dialog box, select Windows Form (in Visual Basic or in Visual C# or Windows Forms Application (.NET) (in Visual C++) from the Templates pane. In the Name box, name the form Form2. Select Open to add the form to the project.

   This form will be the template for your MDI child forms.

The Windows Forms Designer opens, displaying Form2.

6. From the Toolbox, drag a RichTextBox control to the form.

7. In the Properties window, set the Anchor property to Top, Left and the Dock property to Fill.

This causes the RichTextBox control to completely fill the area of the MDI child form, even when the form is resized.

8. Double click the New menu item to create a Click event handler for it.

9. Insert code similar to the following to create a new MDI child form when the user clicks the New menu item.

10. In the drop-down list at the top of the Properties window, select the menu strip that corresponds to the File menu strip and set the MdiWindowListItem property to the Window ToolStripMenuItem.

This enables the Window menu to maintain a list of open MDI child windows with a check mark next to the active child window.

11. Press F5 to run the application. By selecting New from the File menu, you can create new MDI child forms, which are kept track of in the Window menu item.

MDI VS SDI

BASIS OF COMPARISON	MDI	SDI
Description	MDI is a type of graphic user interface which is able to show more than a single document at a time on the screen.	SDI is a Graphic User Interface which is able to show one document at a time on the screen.
Limits	Child windows per documents are allowed in MDI.	One document per window is enforced in SDI.
Container Control	MDI is a container control.	SDI is not a container control.
Operation	MDI contains multiple documents which at a time appear as child window.	SDI contains one window only at a time.
Interface	MDI supports many interfaces which means you can handle many applications at a time according to user's requirement.	SDI supports one interface which means you can handle only one application at a time.

Switching Between Documents	For switching between documents MDI uses special interface inside the parent window.	For switching between documents SDI uses Task Manager for that.
Grouping	In MDI grouping is implemented naturally.	SDI grouping is possible through special window managers.
Maximizing Documents	For maximizing documents, parents window is maximized by MDI.	For maximizing documents, parent windows are maximized through a special code or window manager.
Flexibility	Switching focus to the specific document can be easily handled by MDI.	Switching focus to specific document is difficult to implement in SDI.

MODULE V

1. INTRODUCTION TO ADO.NET

ADO.NET is a technology used for data and is provided by the Microsoft .NET Framework.

It is a part of the .NET framework that supports the communication between relational and non-relational systems with the help of a set of software components. It supports disconnected architecture using which programmers are allowed to access data and data services from a database without depending on the data source.

ADO.NET is comprised of a group of built-in classes that are useful for establishing the database connection, for gaining access to XML, relational data, and application data, and for retrieval of a result. It can be used in various programming languages such as Visual Basic.NET, Visual C++, etc., that are supported by the .NET framework.

Advantages of ADO.NET

ADO.NET has various advantages which can be categorized into the following categories:

- Interoperability:It provides the ability to communicate across heterogeneous environments, once the connection has been established between them.
- Scalability: It provides the ability to serve an increasing number of clients without reducing the performance of the system. So we can say that ADO.NET is highly scalable because it is flexible enough to be easily expanded when there is a requirement for the same.
- Productivity: It provides the ability to rapidly develop robust applications for data access using rich and extensible component object models provided by the ADO.NET.
- Performance: An improvement over earlier ADO.NET versions because of the disconnected data model. It can establish connections quickly to fetch data without any delay.

Scope of ADO.NET

ADO.NET being one of the products of Microsoft, it is good enough to position itself strongly in the market. ADO.NET has massive community support, so it is definitely having a large scope ahead. You could learn ADO.NET along with hands-on experience on the .Net frame work in order to have a good scope. Any full-stack developer

who has a better grip over both front-end and back-end technology can precisely learn ADO.NET.

1.1 ADO.NET OBJECT MODEL

ADO.NET is based on an Object Model where data residing in the database is accessed using a data provider. It is a technology of data access given by the Microsoft .Net Framework, which helps to communicate between relational and non-relational systems using a common group of components.

The components of ADO.NET architecture are:

- Data Provider: It provides data to all the applications that perform the database updates. The application can access data through the DataSet or DataReader object. A data provider is a having group of components such as Command, Connection, DataReader, and DataAdapter objects. Command and Connection objects are the necessary components irrespective of the operations like Insert, Delete, Select, and Update.

- Connection: The connection object is needed to connect with the database such as SQL Server, MySQL, Oracle, etc. To create a connection object, you must know about where the database is located(Ex: IP address or machine name, etc.) and the security credentials(Ex: user name and password-based authentication or windows authentication).

- Command: The command object is the component where you will write the SQL queries. Then by using the command object, execute the queries over the connection. By using the command object and SQL queries, you will be able to fetch the data or send the data to the database.

- DataReader: DataReader is a connected read-only RecordSet that is helpful in reading the records in the forward-only mode.

- **DataAdapter:** The DataAdapter acts as a bridge between the dataset and command object. It receives the data from the command object and puts it into the data set.

- **DataSet:** The DataSet is a disconnected RecordSet that can be browsed in both forward and backward directions. We can also update the data using the dataset. DataSet is filled by using DataAdapter.

- **DataView Class:** A DataView allows you to create various views of data from DataTable, which can be used for data-binding applications. Using this, you can display the table with different order of sorting or you can filter the data based on a filter expression or by row state, etc.

- **XML:** It is possible to create an XML representation of a dataset. In the dataset's XML representation, data is represented in XML format and the database schema is represented in XML Schema Definition(XSD) language.

1.1.1 DATA PROVIDER

Data providers are used to transferring the data between the client application and the data store. It encapsulates the database-specific details. Data providers are helpful for database connection, data retrieval, storing the data in a dataset, reading the retrieved data, and updating the database.

The data providers that comes along with the ADO.NET Framework are:

- • OLE DB: The OLEDB provider is available under System.Data.OleDb namespace. This provider can be used to access Microsoft Access, DB2/400, SyBase, and SQL Server 6.5 and earlier.
- • ODBC: The ODBC provider is available under System.Data.Odbc namespace. This provider is used when there will not be any newer provider is available.
- • SQL Server:The Microsoft SQL Server provider is available under System.Data.SqlClient namespace. Classes available under this provider will provide the same functionality as the generic OLEDB provider.

1.1.2 DATASET

The DataSet is a collection of database tables(row and column format) that contain the data. It is helpful for fetching the data without any need for Data Source interaction, that is why it is called a disconnected data access method.

 It is an in-memory data store that can contain multiple tables at the same time. DataRelation objects can be used to relate these tables.

 For creating a DataSet object, ADO.NET provides a DataSet class that consists of constructors and methods to carry out data-related operations.

 It can be used with various data sources, with XML data, or to manage the application's local data. The DataSet will include related tables, data constraints, and relationships among the tables.

1.2 CONNECTING TO DATABASE

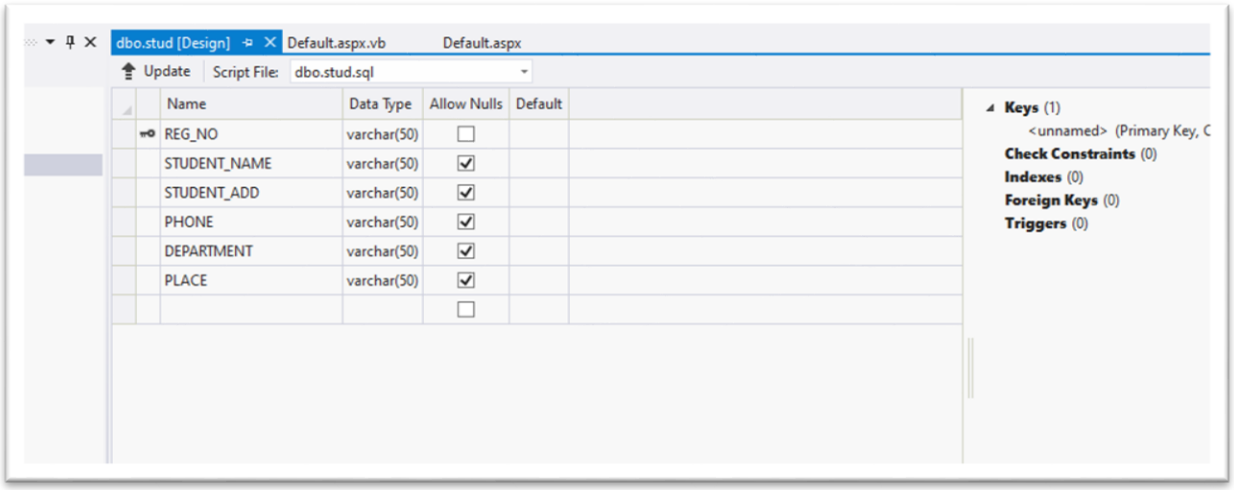
To connect with SQL Server, we must have it installed in our system. We are using Microsoft

SQL Server Management Tool to connect with the SQL Server. We can use this tool to handle database. Now, follow the following steps to connect with SQL Server.

Establish connection and create a table

```
CREATE TABLE [dbo].[stud] (
    [REG_NO]       VARCHAR (50) NOT NULL,
    [STUDENT_NAME] VARCHAR (50) NULL,
    [STUDENT_ADD]  VARCHAR (50) NULL,
    [PHONE]        VARCHAR (50) NULL,
    [DEPARTMENT]   VARCHAR (50) NULL,
    [PLACE]        VARCHAR (50) NULL,
    PRIMARY KEY CLUSTERED ([REG_NO] ASC)
```

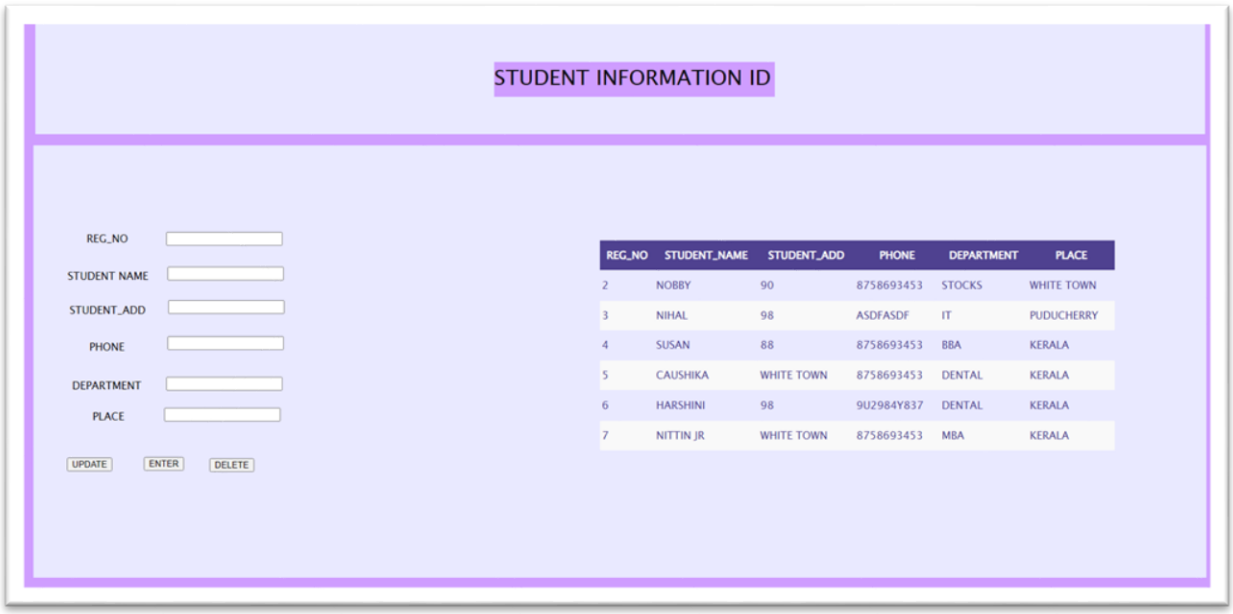
```
);
```



1.INSERT DATA INTO THE TABLE
2.UPDATE DATA IN TABLE
3.DELETE DATA FROM TABLE

```
Imports System.Configuration
Imports System.Data
Imports System.Data.SqlClient

Partial Class _Default
    Inherits System.Web.UI.Page

    Protected Sub Button1_Click(sender As Object, e As EventArgs)
Handles Button1.Click
        Dim cs =
ConfigurationManager.ConnectionStrings("ConnectionString").ToString
        Dim con As SqlConnection = New SqlConnection(cs)
        con.Open()
        Dim Sql = "insert into stud values('" + TextBox7.Text +
"','" + TextBox2.Text + "','" + TextBox3.Text + "','" +
TextBox4.Text + "','" + TextBox5.Text + "','" + TextBox6.Text + "')"
        Dim cmd As SqlCommand = New SqlCommand(Sql, con)
        cmd.ExecuteNonQuery()
        con.Close()
        Response.Redirect("Default.aspx")
```

```
    End Sub

    Protected Sub Button2_Click(sender As Object, e As EventArgs)
Handles Button2.Click
        Dim cs =
ConfigurationManager.ConnectionStrings("ConnectionString").ToString
        Dim con As SqlConnection = New SqlConnection(cs)
        con.Open()
        Dim A = "delete from stud where STUDENT_NAME = '" +
TextBox2.Text + "'"
        Dim cmd As SqlCommand = New SqlCommand(A, con)
        cmd.ExecuteNonQuery()
        con.Close()
        Response.Redirect("Default.aspx")


    End Sub

    Protected Sub Button3_Click(sender As Object, e As EventArgs)
Handles Button3.Click
        Dim cs =
ConfigurationManager.ConnectionStrings("ConnectionString").ToString
        Dim con As SqlConnection = New SqlConnection(cs)
        con.Open()
        Dim B = "update stud set STUDENT_NAME = '" + TextBox3.Text +
"' where STUDENT_NAME = '" + TextBox2.Text + "'"
        Dim C = "update stud set STUDENT_ADD = '" + TextBox4.Text +
"' where STUDENT_ADD = '" + TextBox3.Text + "'"
        Dim cmd As SqlCommand = New SqlCommand(B, con)
        Dim cXs As SqlCommand = New SqlCommand(C, con)
        cXs.ExecuteNonQuery()
        cmd.ExecuteNonQuery()
        con.Close()
        Response.Redirect("Default.aspx")
    End Sub
End Class
```


STUDENT INFORMATION ID
REG_NO
STUDENT NAME
STUDENT_ADD
PHONE
DEPARTMENT
PLACE
UPDATE
ENTER
DELETE
REG_NO STUDENT_NAME STUDENT_ADD PHONE DEPARTMENT PLACE
2 NOBBY 90 8758693453 STOCKS WHITE TOWN
3 NIHAL 98 ASDFASDF IT PUDUCHERRY
4 SUSAN 88 8758693453 BBA KERALA
5 CAUSHIKA WHITE TOWN 8758693453 DENTAL KERALA
6 HARSHINI 98 9U2984Y837 DENTAL KERALA
7 NITTIN JR WHITE TOWN 8758693453 MBA KERALA