



**JAMAL MOHAMED COLLEGE**

**(AUTONOMOUS) TIRUCHIRAPPALLI.**

**DEPARTMENT OF COMPUTER SCIENCE &IT**

**VB .NET**

**PREPARED BY**

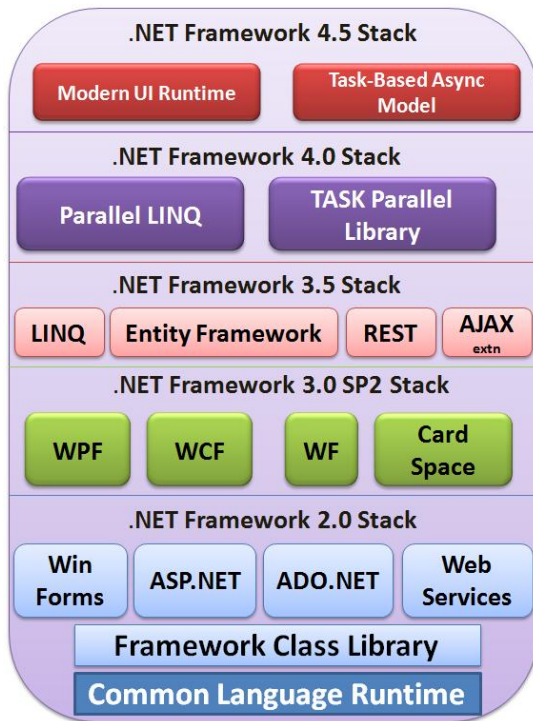
S. Benazir Butto

Assistant Professor

Department of CS&IT (SF-Women)

## UNIT 1

### THE .NET FRAMEWORK AND THE COMMON LANGUAGE RUNTIME:



.NET Framework (Pronounced dot net) is a software framework developed by Microsoft that runs primarily on Microsoft windows.

VB. NET is only one component of a revolution in windows-the .NET framework.

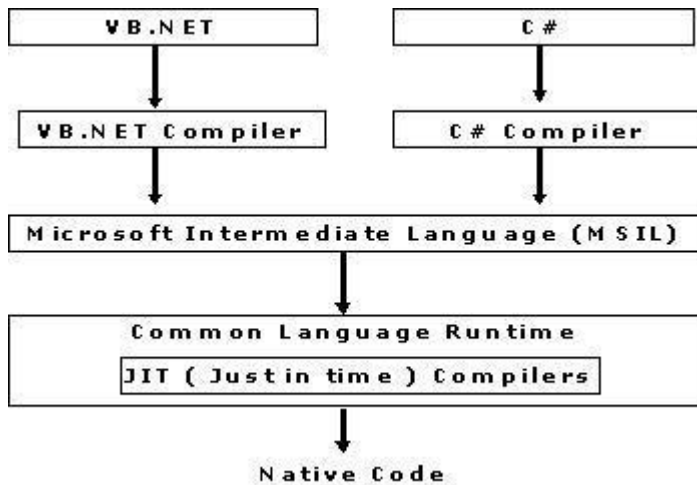
NET framework execute in a SOFTWARE environment. (As contrasted to HARDWARE environment) known as Common Language Runtime, an application virtual machine that provides services such as security memory management, and exception handling.

.NET framework is intended to be used by most new application created for Windows platform. Microsoft also produces an INTEGRATED DEVELOPMENT ENVIRONMENT largely for .NET software called visual studio.

At the base of the .NET framework in the common language runtime (CLR) The CLR provides additional services including memory management, type safety, exception handling, garbage collection, security and thread management.

The .NET Framework class library is the second major part of the .NET Framework. The class library hold's on immense amount of prewritten code that all the application you create with VISUAL BASIC, VISUAL C++, VISUAL C# and other Visual studio language are build on.

All this assumes that you're working on a machine that has the .NET Framework, and therefore the CLR and the .NET Framework class library, installed. The code for all elements we use in a VB.NET application –forms, buttons, menus, and all the rest-all comes from the class library.



The CLR converts CIL (Common Intermediate Language) to native code.

### **BUILDING VB.NET APPLICATIONS:**

To build applications in VB.NET, we have to get some terminology under our belts, because the .NET framework requires a new structure for applications.

#### **i) Assemblies:**

An assembly holds the intermediate language modules for your applications. When you create an application in VB.NET and run it, VB.NET creates one or more assemblies, which are run by the CLR..

#### **ii) Solutions and Projects:**

When you created applications in Visual Basic 6.0, you created projects. Each project held the code and data for an application, ActiveX control, or whatever else you wanted to build. If you wanted to combine projects together, you create default a project group.

#### **iii) File Extensions Used in VB.NET:**

- ❖ .vb --- can be basic windows form, a code file, a module file for storing functions, a user control, a data form.
- ❖ .xsd--- An XML schema provided to create typed datasets.
- ❖ .xml---An XML document file.
- ❖ .htm---An HTML document.
- ❖ .txt--- A text file.

- ❖ .xslt---An XSLT stylesheetfile,used to transform XMLdocuments and XML schemas.
- ❖ .css---A cascading stylesheet file.
- ❖ .rpt---A crystal Report.
- ❖ .bmp---A bitmap file.
- ❖ .js---A javascript file.
- ❖ .vbs---A VBScript file.
- ❖ .wsf---A Windows scripting file.
- ❖ .aspx---A Web form.
- ❖ .asp---An active server page.
- ❖ .asmx---A Web service class.
- ❖ .resx---A resource file used to store resource information.

#### **iv) Debug and Release Versions:**

In a debug of your program,Visual Basic stores a great deal of data needed to interface with the debugger in your prigramwhen it runs,and this not only makes the corresponding assembly larger,but also slower. When you create a new solution,Visual Basic creates it in debug mode,

- Select the configuration Manager item in the Build menu  
And the click ok.
- Select the solutions you want to set the mode for by clicking it in the Solutions Explorer,and find its Active Config property in the properties window.
- Select the Solutions you want to set the mode for by clicking it in the Solutions Explorer,and select the properties item in the Project menu.
- Probably the easiest way to set the solution mode to release or debug is simply to use the drop-down list box that appears in the Visual Basic.NET standard toolbar,at the top of the IDE.

### **THE VISUAL BASIC INTEGRATED DEVELOPMENT ENVIRONMENT**

i) The Start Page: The start page to select from recent projects;bydefault,the get started item is selected in the start page at upper left.You can also create the new project here by clicking the New project button.

ii) The Menu System: File->new->project.

Menu item to bring up the new project dialog box.

There are hundreads of menu items here,and many useful ones that will quickly become favorites,such as file|new|project that you use to create a new project.

iii) Tool Bars: Tool bars provide a quick way to select many items,and although I personally usually stick to using the menu system,there's no doubt that tool bar buttons can be quicker.

For eg.

To save the file you are currently working on, you only need to click the diskette button in the standard tool bar, or the stacked diskette button to save all the files in the solution.

iv) The New Project Dialog Box: In addition to letting you select from all the possible types of projects you can create in the Visual Basic, you can also set the name of the project, and its location; for windows projects, the location is a folder on disk,

But for web projects you specify a server running IIS.

v) Graphic designers: You are actually looking at a Windows form designer, you can manipulate the form, as well as add controls to it and so on.

There are several different types of graphical designers, including:

- Windows form designers.
- Web form designers.
- Component designers.
- XML designers.

v) Code Designers: let you edit the code for a component, and you can see a code designer.

You can use the tabs At the top center of the IDE to switch between graphical designers such as the tabs,

Form1.vb which displays a graphical designer

Form1.vb tab which displays the corresponding code designer.

vi) Intellisense: Listing all the possible options and even completing your typing for you

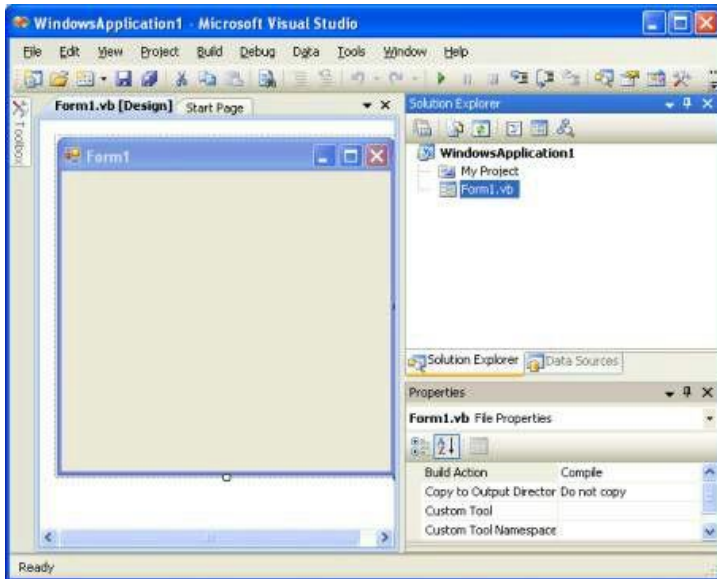
Intellisense is made up of a number of options, including:

- List Members---Lists the members of the object.
- Parameter info---Lists the arguments of the procedure calls.
- Quick Info---Displays information in tool tips as the mouse rests on elements in your code.
- Complete Word---Completes typed words.
- Automatic Brace Matching—Adds parentheses or braces as needed.

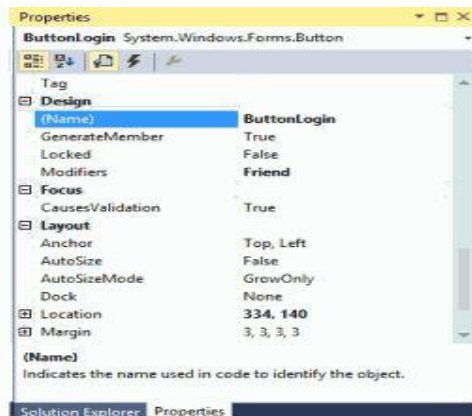
vii) The Tool Box : You can see these tabs, marked Data, Components, Window Forms and General,. The Data, Components, Window Forms and General tabs appears when you're Working with Windows form in a Windows form designer.



viii) The Solution Explorer: To add new items, you can use the menu item in the project menu, such as Window Form and Add User Control. TO add new empty modules and classes to project, you can use the project/Add New Items menu item.

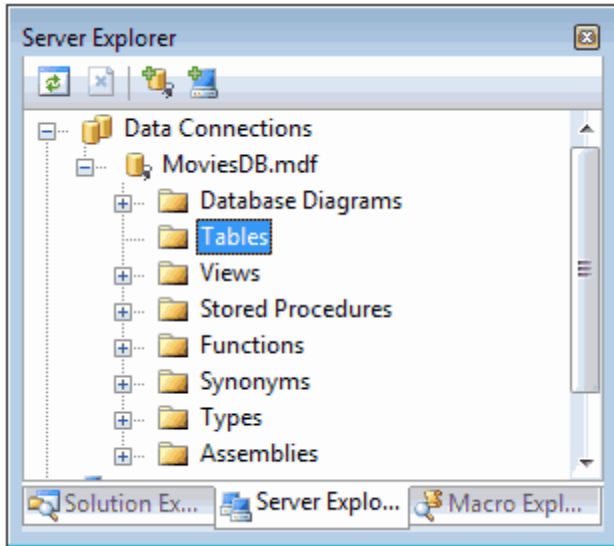


ix) The Properties Window: The properties window is divided into two columns of text, with the properties on the left, and their setting on the right. The object you're setting properties for appears in the drop-down list box at the top of the Properties Window, and you can select from all the available objects using that list box.



x) Component Tray: The component wasn't visible at the run time-such as a timer control-the timer would still appear on the form at design time. That's changed in VB.NET. now, when you add components that are invisible at run time, they'll appear in a component tray.

xi) Server Explorer: To explore what's going on in a server, and it's a great tool to help make distant servers feel less distant, because you can see everything you need in an easy graphical environment.



## VISUAL BASIC STATEMENTS

- Keywords- Words reserved for Visual Basic's use.
- Operators-Symbols used to perform operations, like +, which performs addition operations;- , which performs subtraction operations; and so on.
- Variables-Symbolic names given to values stored in memory and declared with the dim keyword. For example, if you've declared a variable named temperature as an Integer type, you can store integer values like 72 or 83 in it.
- Literal values-Simple values, like 5 or "Hello".
- Constants-The same as variables, except that constants are assigned a value that cannot be altered.
- Expressions-Combinations of terms and/or keywords that yield a value. For example, if the variable temperature holds the value 72, then the expression temperature +3 yields the value 75.

## ALL ABOUT STATEMENT SYNTAX

Each statement has its own syntax, and there are a few conventions and terms you should be aware of before getting started:

In the formal definition of each statement, you use brackets, [ and ], for optional items, and curly braces, { and }, to indicate that you select one of the enclosed items, like this for the Dim statement:

```
[ <attrlist> ] [ { Public | Protected | Friend | Protected Friend | Private | Static } ] [ Shared ] [ Shadows ] [ ReadOnly ] Dim [ WithEvents ] name [ (boundlist) ] [As [ New ] type ] [ = initexpr ]
```

- **attrlist** – A list of attributes that apply to the variables you're declaring in the statement.

- **Public** – Gives variables public access, which means there are no restrictions on their accessibility.
- **Protected**- Gives variables protected access, which means they are accessible only from within their own class or from a class derived from that class.
- **friend**— Gives variables friend access, which means they are accessible from within the program that contains their declaration, as well as from anywhere else in the same assembly.
- **Protected Friend**— Gives variables both protected and friend access. which means they can be used by code in the same assembly, as well as by code in derived classes.
- **Private**— Gives variable private access, which means they are accessible only from within their declaration context (usually a class) ,including any nested procedures.
- **Static**— Makes variables static, which means they retain their values, even after the procedure in which they're declared ends.
- **Shared**—Declares a shared variable, which means it is not associated with a specific instance of a class or structure, but can be shared across many instances.
- **Shadows**—Makes this variable a shadow of an identically named programming element in a base class. A shadowed element is unavailable in the derived class that shadows it.
- **readOnly**--Means this variable only can be read and not written. This can be useful for creating constant members of reference types, such as an object variable with preset data members.
- **WithEvents**—Specifies that this variable is used to respond to events caused by the instance that was assigned to the variable. Note that you cannot specify both WithEvents and New in the same variable declaration.
- **name**—The name of the variable. You separate multiple variables by commas. If you specify multiple variables. each variable is declared of the data type given in the first As clause encountered after its name part.
- **boundlist**- Used to declare arrays; gives the upper bounds of the dimensions of an any variable. Multiple upper bounds are separated by commas.
- **New**—Means you want to create a new object immediately.
- **type**—The data type of the variable.
- **initexpr**—An initialization expression that is evaluated and the result is assigned to the variable when it is created,.

**DECLARING VARIABLES:** Declaring variable which is used to store some data in program.

```
<attrlist> [ { Public | Protected | Friend | Protected Friend | Private | Static } ] [ Shared ] [ Shadows  
[ ReadOnly ] Dim [ WithEvents ] name[(boundlist) ] [ As [ New] type ] [ = initexpr]
```



- **attrlist** – A list of attributes that apply to the variables you’re declaring in the statement.
- **Public** – Gives variables public access, which means there are no restrictions on their accessibility.
- **Protected**- Gives variables protected access, which means they are accessible only from within their own class or from a class derived from that class.
- **friend**— Gives variables friend access, which means they are accessible from within the program that contains their declaration, as well as from anywhere else in the same assembly.
- **Protected Friend**— Gives variables both protected and friend access. which means they can be used by code in the same assembly, as well as by code in derived classes.
- **Private**— Gives variable private access, which means they are accessible only from within their declaration context (usually a class) ,including any nested procedures.
- **Static**— Makes variables static, which means they retain their values, even after the procedure in which they're declared ends.
- **Shared**—Declares a shared variable, which means it is not associated with a specific instance of a class or structure, but can be shared across many instances.
- **Shadows**—Makes this variable a shadow of an identically named programming element in a base class. A shadowed element is unavailable in the derived class that shadows it.
- **readOnly**--Means this variable only can be read and not written. This can be useful for creating constant members of reference types, such as an object variable with preset data members.
- **WithEvents**—Specifies that this variable is used to respond to events caused by the instance that was assigned to the variable. Note that you cannot specify both WithEvents and New in the same variable declaration.
- **name**—The name of the variable. You separate multiple variables by commas. If you specify multiple variables. each variable is declared of the data type given in the first As clause encountered after its name part.
- **boundlist**- Used to declare arrays; gives the upper bounds of the dimensions of an any variable. Multiple upper bounds are separated by commas.
- **New**—Means you want to create a new object immediately.
- **type**—The data type of the variable.
- **initexpr**—An initialization expression that is evaluated and the result is assigned to the variable when it is created,.

Each attribute in the attrlist list must use this syntax:

```
attrname[({attrargs | attrinit})]>
```

Here are the parts of the attrlist List:

- attrname- Name of the attribute.
- attrargs -List of arguments for this attribute. Separate multiple arguments with commas
- attrinit—List of field or property initializers for this attribute.

```
Dim EmployeeID As Integer = 1

Dim Employee As String =" Slob Owen"

Dim EmployeeAddress As String
```

Default values:

- 0 for all numeric types (including Byte).
- Binary 0 for Char.
- Nothing for all reference types (including Object, String, and all arrays). Nothing means there is no object associated with the reference.
- False for Boolean.
- 12:00 AM of January, 1 of the year 1 for Date.

**VARIABLE PREFIXES**

Ddata type	Prefix
Bboolean	Bln
Bbyte	Byt
Ccollection object	Col
Ddate(Time)	Dtm
Ddouble	Dbl
Eerror	Err
Iinteger	Int
Llong	Lng
oObject	Obj
SSingle	Sng
String	Str
User-defined type	Udt

**DECLARING CONSTANT:** class, structure, procedure, or block level to declare constant for use in place of literal values.

```
[ <attrlist> ] [{ Public | Protected | Friend | Protected Friend | Private }] [ Shadows ]const name[ AS type ] = initexpr
```

Here are the various parts of this statement:

- **attrlist**—a list of attributes that apply to the constant you're declaring in this statement you separate multiple attribute with commas.
- **Public**—gives constant public access, which means there are no restriction on their accessibility.
- **Protected**—gives constant protected access, which means they are accessible only from within their own class or from a class derived from that class.
- **Friends**—gives constant friend access, which means they are accessible from within the program that contain their declaration, as well as anywhere else in the same assembly.
- **protected friend**—Gives constants both protected and friend access, which means they can be used by code in the same assembly, as well as by code in derived classes.
- **Private**—Gives constants private access, which means they are accessible only from within their declaration context (usually a class), including any nested procedures.
- **Shadows**—Makes this constant a shadow of an identically named programming element in a base class. A shadowed element is unavailable in the derived class that shadows it.
- **name**—The name of the constant. You can declare as many constants as you like in the same declaration statement, specifying the name and initexpr parts for each one.
- **type**—The data type of the constant.
- **initexpr**—An initialization expression. Can consist of a literal, another constant, a member of an enumeration, or any combination of literals, constants, and enumeration members.

Each attribute in the attrlist must use this syntax:

```
<attrname [( { attrargs | attrint } )]>
```

Each attribute in the attrlist list must use this syntax:

- **attrname**—Name of the attribute.
- **attrargs**—List of arguments for this attribute. Separate multiple arguments with commas.

•**attrinit**—List of field or property initializers for this attribute. Separate multiple arguments with commas.

```
Imports System.Console
```

```
Module Modu1
```

```
Sub Main()
```

```
Const pi = 3.14159
```

```
Radius, Area As Single
```

```
Radius = 1
```

```
Area = Pi * Radius * Radius
```

```
WriteLine("Area = " & Str(Area))
```

```
End Sub
```

```
End Module
```

## DATA TYPES IN VB.Net

Visual Basic .NET (VB .NET) lets you get right to the basics without having to wade through translators. The following table shows you the data types VB .NET uses, as well as their CLR structure, storage size, and value ranges. Use the info for good, not evil!

Visual Basic Type	Common Language Runtime Type Structure	Storage Size
Boolean	System.Boolean	2 bytes
Byte	System.Byte	1 byte
Char	System.Char	2 bytes
Date	System.DateTime	8 bytes
Decimal	System.Decimal	16 bytes
Double (double-precision floating-point)	System.Double	8 bytes
Integer	System.Int32	4 bytes
Long (long integer)	System.Int64	8 bytes
Object	System.Object (class)	4 bytes
Short	System.Int16	2 bytes

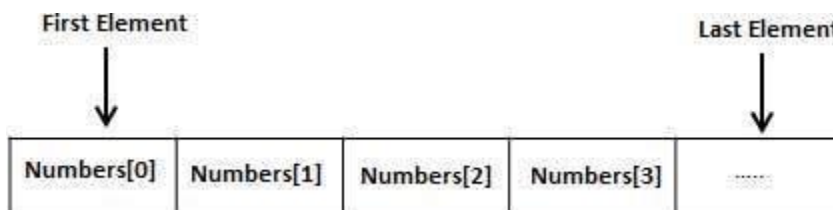
Single (single-precision floating-point)	System.Single	4 bytes
String (variable-length)	System.String (class)	Depends on implementing platform
User-Defined Type (structure)	(inherits from System.ValueType)	Sum of the sizes of its members

## ARRAYS:

An array stores a fixed-size sequential collection of elements of the same type.

An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.



## Creating Arrays in VB.Net

To declare an array in VB.Net, you use the Dim statement. For example,

```
Dim intData(30) ' an array of 31 elements
```

```
Dim strData(20) As String ' an array of 21 strings
```

```
Dim twoDarray(10, 20) As Integer ' a two dimensional array of integers
```

```
Dim ranges(10, 100) ' a two dimensional array
```

## Declaring Values in Arrays in VB.Net

```
Dim intData() As Integer = {12, 16, 20, 24, 28, 32}
```

```
Dim names() As String = {"Karthik", "Sandhya", "Shivangi", "Ashwitha", "Somnath"}
```

```
Dim miscData() As Object = {"Hello World", 12d, 16ui, "A"c}
```

Example:

```
Module arrayApl
```

```
Sub Main()
```

```
Dim n(10) As Integer ' n is an array of 11 integers '
```

```
Dim i, j As Integer
```

```
' initialize elements of array n '
```

```
For i = 0 To 10
```

```
    n(i) = i + 100 ' set element at location i to i + 100
```

```
Next i
```

```
' output each array element's value '
```

```
For j = 0 To 10
```

```
    Console.WriteLine("Element({0}) = {1}", j, n(j))
```

```
Next j
```

```
Console.ReadKey()
```

```
End Sub
```

```
End Module
```

## Dynamic Arrays

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)

Where,

- 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.

```
Module arrayApl
```

```
Sub Main()
```

```
Dim marks() As Integer
```

```
ReDim marks(2)
```

```
marks(0) = 85
```

```

marks(1) = 75
marks(2) = 90
ReDim Preserve marks(10)
marks(3) = 80
marks(4) = 76
marks(5) = 92
marks(6) = 99
marks(7) = 79
marks(8) = 75
For i = 0 To 10
    Console.WriteLine(i & vbTab & marks(i))
Next i
Console.ReadKey()
End Sub
End Module

```

## HANDLING STRING

Strings are supported by the .NET **String** class in Visual Basic. You declare a string this way:

```
Dim strText As String
```

As with other types of variables, you can also initialize a string when you declare it, like this:

For example, you use **Left**, **Mid**, and **Right** to divide a string into substrings, you find the length of a string with **Len**, and so on.

.NET framework functions are built into the **String** class that VB .NET uses.

For example, the Visual Basic **UCase** function will convert strings to upper case, and so will the **String** class's **ToUpper** method.

Example:

```

Option Strict On
Module Module1
    Sub Main()
        Dim strText1 As String = "welcome to visual basic"
        Dim strText2 As String

```

*Dim strText3 As String*

*strText2 = UCase(strText1)*

*strText3 = strText1.ToUpper*

*System.Console.WriteLine(strText2)*

*System.Console.WriteLine(strText3)*

*End Sub*

*End Module*

To do this	Use this
Concatenate two strings	&, +, String.Concat, String.Join
Compare two strings	StrComp, String.Compare, String.Equals, String.CompareTo
Convert strings	StrConv, CStr, String.ToString
Copying strings	=, String.Copy
Convert to lowercase or uppercase	Format, LCase, UCase, String.Format, String.ToUpper, String.ToLower
Convert to and from numbers	Str, Val.Format, String.Format
Create string of a repeating character	Space, String, String.String
Create an array of strings from one string	String.Split
Find length of a string	Len, String.Length
Format a string	Format, String.Format
Get a substring	Mid, String.SubString
Insert a substring	String.Insert
Justify a string with padding	LSet, Rset, String.PadLeft, String.PadRight
Manipulate strings	Str, Left, LTrim, Mid, Right, RTrim, Trim, String.Trim, String.TrimEnd, String.TrimStart
Remove text	Mid, String.Remove
Replace text	Mid, String.Replace
Set string comparison rules	Option Compare
Search strings	InStr, String.Chars, String.IndexOf, String.IndexOfAny,



To do this	Use this
	String.LastIndexOf, String.LastIndexOf Any

## FIXED-LENGTH STRINGS

**VB6.FixedLengthString**,—that supports fixed-length strings; for example, this declaration in VB6, which declares a string of 1000 characters:

```
Dim strString1 As String * 1000 (OR)
```

```
Dim strString1 As New VB6.FixedLengthString(1000)
```

## OPERATORS:

An operator is a symbol that 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, Comparison Operators, Logical/Bitwise Operators, Assignment Operators, Miscellaneous Operators

### Arithmetic Operators

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

Operator	Description
^	Raises one operand to the power of another
+	Adds two operands
-	Subtracts second operand from the first
*	Multiplies both operands
/	Divides one operand by another and returns a floating point result
\	Divides one operand by another and returns an integer result
MOD	Modulus Operator and remainder of after an integer division

### Comparison Operators

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

<b>Operator</b>	<b>Description</b>
=	Checks if the values of two operands are equal or not; if yes, then condition becomes true.
<>	Checks if the values of two operands are equal or not; if values are not equal, then condition becomes true.
>	Checks if the value of left operand is greater than the value of right operand; if yes, then condition becomes true.
<	Checks if the value of left operand is less than the value of right operand; if yes, then condition becomes 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.
<=	Checks if the value of left operand is less than or equal to the value of right operand; if yes, then condition becomes true.

- **Is Operator** - It compares two object reference variables refer to the same object without performing value comparisons. If object1 and object2 both refer to the exact same object instance, result is **True**; otherwise, result is False.
- **IsNot Operator** - It also compares two object reference variables and determines if two object references refer to different objects. If object1 and object2 both refer to the exact same object instance, result is **False**; otherwise, result is True.
- **Like Operator** - It compares a string against a pattern.

### Logical/Bitwise Operators

Assume variable A holds Boolean value True and variable B holds Boolean value False, then:

<b>Operator</b>	<b>Description</b>
And	If both the operands are true, then condition becomes true.
Or	If any of the two operands is true, then condition becomes true.
Not	. If a condition is true, then Logical NOT operator will make false.
Xor	It returns True if both expressions are True or both expressions are False; otherwise it returns False.
AndAlso	It works only on Boolean data. It performs short-circuiting.
OrElse	It works only on Boolean data. It performs short-circuiting.
IsFalse	It determines whether an expression is False.

IsTrue	It determines whether an expression is True.
--------	--

### Assignment Operators

There are following assignment operators supported by VB.Net:

perator	Description
=	Simple assignment operator, Assigns values from right side operands to left side operand
+=	It adds right operand to the left operand and assigns the result to left operand
-=	It subtracts right operand from the left operand and assigns the result to left operand
*=	It multiplies right operand with the left operand and assigns the result to left operand
/=	It divides left operand with the right operand and assigns the result to left operand (floating point division)
\=	It divides left operand with the right operand and assigns the result to left operand (Integer division)
^=	Exponentiation and assignment operator. It raises the left operand to the power of the right operand and assigns the result to left operand.
<<=	Left shift AND assignment operator
>>=	Right shift AND assignment operator
&=	Concatenates a String expression to a String variable or property and assigns the result to the variable or property.

### CONTROL STRUCTURES:

#### DECISION MAKING:

Decision making structures require that the programmer specify one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.

##### i) SIMPLE IF STMT:

It is the simplest form of control statement, frequently used in decision making and changing the control flow of the program execution.

*Syntax: If condition Then*

*[Statement(s)]*  
**End If**

Example of an If-Then statement is:

```
If (a <= 20) Then  
c = c + 1  
End If
```

**Example:**

*Module decisions*

```
Sub Main()  
    Dim a As Integer = 10  
    If (a < 20) Then  
        Console.WriteLine("a is less than 20")  
    End If  
    Console.WriteLine("value of a is : {0}", a)  
    Console.ReadLine()  
End Sub
```

*End Module*

**ii) IF .....ELSE STMT**

An **If** statement can be followed by an optional **Else** statement, which executes when the Boolean expression is false.

Syntax:    ***If(boolean\_expression)Then***  
                  *'statement(s) will execute if the Boolean expression is true*  
          ***Else***  
                  *'statement(s) will execute if the Boolean expression is false*  
          ***End If***

**Example:**

*Module decisions*

```
Sub Main()  
    'local variable definition '
```

*Dim a As Integer = 100*

*' check the boolean condition using if statement*

*If (a < 20) Then*

*' if condition is true then print the following*

*Console.WriteLine("a is less than 20")*

*Else*

*' if condition is false then print the following*

*Console.WriteLine("a is not less than 20")*

*End If*

*Console.WriteLine("value of a is : {0}", a)*

*Console.ReadLine()*

*End Sub*

*End Module*

### **iii) THE IF...ELSE IF...ELSE STATEMENT**

An **If** statement can be followed by an optional **Else if...Else** statement, which is very useful to test various conditions using single If...Else If statement.

When using If... Else If... Else statements, there are few points to keep in mind.

- An If can have zero or one Else's and it must come after an Else If's.
- An If can have zero to many Else If's and they must come before the Else.
- Once an Else if succeeds, none of the remaining Else If's or Else's will be tested.

**Syntax:** *If(boolean\_expression 1)Then*

*' Executes when the boolean expression 1 is true*

*ElseIf( boolean\_expression 2)Then*

*' Executes when the boolean expression 2 is true*

*ElseIf( boolean\_expression 3)Then*

*' Executes when the boolean expression 3 is true*

*Else*

*' executes when the none of the above condition is true*

*End If*

**Example:**

## Module decisions

```
Sub Main()  
    Dim a As Integer = 100  
    If (a = 10) Then  
        Console.WriteLine("Value of a is 10") '  
    ElseIf (a = 20) Then  
        Console.WriteLine("Value of a is 20") '  
    ElseIf (a = 30) Then  
        Console.WriteLine("Value of a is 30")  
    Else  
        Console.WriteLine("None of the values is matching")  
    End If  
  
        Console.WriteLine("Exact value of a is: {0}", a)  
        Console.ReadLine()  
  
End Sub  
End Module
```

## iv) SELECT CASE

A **Select Case** statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each select case.

### Syntax:

```
Select [ Case ] expression  
    [ Case expressionlist  
        [ statements ] ]  
    [ Case Else  
        [ elsestatements ] ]  
End Select
```

**expression**: is an expression that must evaluate to any of the elementary data type in VB.Net, i.e., Boolean, Byte, Char, Date, Double, Decimal, Integer, Long, Object, SByte, Short, Single, String, UInteger, ULong, and UShort.

- **expressionlist**: List of expression clauses representing match values for *expression*. Multiple expression clauses are separated by commas.

- **statements**: statements following Case that run if the select expression matches any clause in *expressionlist*.
- **elsetatements**: statements following Case Else that run if the select expression does not match any clause in the *expressionlist* of any of the Case statements.

### **Example:**

#### *Module decisions*

```

Sub Main()
    Dim grade As Char
    grade = "B"
    Select grade
        Case "A"
            Console.WriteLine("Excellent!")
        Case "B", "C"
            Console.WriteLine("Well done")
        Case "D"
            Console.WriteLine("You passed")
        Case "F"
            Console.WriteLine("Better try again")
        Case Else
            Console.WriteLine("Invalid grade")
    End Select
    Console.WriteLine("Your grade is {0}", grade)
    Console.ReadLine()
End Sub
End Module

```

## **LOOPING STATEMENT**

A loop statement allows us to execute a statement or group of statements multiple times and following is the general form of a loop statement in most of the programming languages:

### **LOOP CONTROL STATEMENTS:**

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.

## **i) DO LOOP:**

It repeats the enclosed block of statements while a Boolean condition is True or until the condition becomes True. It could be terminated at any time with the Exit Do statement.

**Syntax:**

```
Do { While | Until } condition  
    [ statements ]  
    [ Continue Do ]  
    [ statements ]  
    [ Exit Do ]  
    [ statements ]  
Loop  
  
-or-  
  
Do  
    [ statements ]  
    [ Continue Do ]  
    [ statements ]  
    [ Exit Do ]  
    [ statements ]  
Loop { While | Until } condition
```

## **Example:**

*Module loops*

*Sub Main()*

*' local variable definition*

*Dim a As Integer = 10*

*'do loop execution*

*Do*

*Console.WriteLine("value of a: {0}", a)*

*a = a + 1*

*Loop While (a < 20)*

*Console.ReadLine()*

*End Sub*

*End Module*



## ii) FOR LOOP:

It repeats a group of statements a specified number of times and a loop index counts the number of loop iterations as the loop executes.

**Syntax:**

```
For counter [ As datatype ] = start To end [ Step step ]  
    [ statements ]  
    [ Continue For ]  
    [ statements ]  
    [ Exit For ]  
    [ statements ]  
Next [ counter ]
```

### Example

*Module loops*

```
Sub Main()  
    Dim a As Byte  
    For a = 10 To 20  
        Console.WriteLine("value of a: {0}", a)  
    Next  
    Console.ReadLine()  
End Sub  
End Module
```

## iii) FOR EACH ....NEXT LOOP

It repeats a group of statements for each element in a collection. This loop is used for accessing and manipulating all elements in an array or a VB.Net collection.

**Syntax:**

```
For Each element [ As datatype ] In group  
    [ statements ]  
    [ Continue For ]  
    [ statements ]  
    [ Exit For ]  
    [ statements ]  
Next [ element ]
```

## Example

*Module loops*

```
Sub Main()  
    Dim anArray() As Integer = {1, 3, 5, 7, 9}  
    Dim arrayItem As Integer  
    For Each arrayItem In anArray  
        Console.WriteLine(arrayItem)  
    Next  
    Console.ReadLine()  
End Sub  
End Module
```

## iv) WHILE LOOP:

It executes a series of statements as long as a given condition is True.

**Syntax:**    *While condition*  
              *[ statements ]*  
              *[ Continue While ]*  
              *[ statements ]*  
              *[ Exit While ]*  
              *[ statements ]*  
              *End While*

## Example

*Module loops*

```
Sub Main()  
    Dim a As Integer = 10  
    While a < 20  
        Console.WriteLine("value of a: {0}", a)  
        a = a + 1  
    End While  
    Console.ReadLine()  
End Sub  
End Module
```

## v) WITH STATEMENT

It is not exactly a looping construct. It executes a series of statements that repeatedly refers to a single object or structure.

**Syntax:**                 *With object*  
  *[ statements ]*  
  *End With*

### Example

*Module loops*

*Public Class Book*

*Public Property Name As String*

*Public Property Author As String*

*Public Property Subject As String*

*End Class*

*Sub Main()*

*Dim aBook As New Book*

*With aBook*

*.Name = "VB.Net Programming"*

*.Author = "Zara Ali"*

*.Subject = "Information Technology"*

*End With*

*With aBook*

*Console.WriteLine(.Name)*

*Console.WriteLine(.Author)*

*Console.WriteLine(.Subject)*

*End With*

*Console.ReadLine()*

*End Sub*

*End Module*

## UNIT – 2

### DEFINING SUB PROCEDURES:

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

*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

**Example : Refer Note**

### DEFINING FUNCTIONS:

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

*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:** specifies the data type of the variable the function returns

**Example: Refer Note**

### UNDERSTANDING SCOPE:

The scope of a variable or constant is the set of all code that can refer to it without qualifying its name. A variable's scope is determined by where the variable is declared.

**Block Scope:** an element declared within a block can be used only within that block.

**Procedure Scope:** An element declared within a procedure is not available outside that procedure, and only the procedure that contains the declaration can use it. Elements at this level are also called local elements, and you declare them with the Dim or Static statement.

**Module Scope:** When you make a declaration at the module level, the accessibility you choose determines the scope. The namespace that contains the module, class, or structure also affects the scope.

**Namespace Scope:** If you declare an element at module level using the Friend or Public statement, it becomes available to all procedures throughout the entire namespace in which it is declared. Note that an element accessible in a namespace is also accessible from inside any namespace nested inside that namespace.

**Example: Refer Note**

#### **EXCEPTION HANDLING:**

i) **Structured Exception:** Structured exception handling is based on a particular statement, the Try...Catch...Finally statement, which is divided into a Try block, optional Catch blocks, and an optional Finally block.

The Try block contains code where exceptions can occur,

The Catch block contains code to handle the exceptions that occur. If an exception occurs in the Try block, the code throws the exception—actually an object based on the Visual Basic Exception class—so it can be caught and handled by the appropriate Catch statement. After the rest of the statement finishes, execution is always passed to the Finally block, if there is one

#### **Syntax and Example Refer Note**

ii) **Unstructured Exception:** It revolves around the On Error GoTo statement

The On Error GoTo statement enables exception handling and specifies the location of the exception-handling code within a procedure.

Syntax:

**On Error { GoTo [ line | 0 | -1 ] | Resume Next }**

**GoTo line:** Enables the exception-handling code that starts at the line specified in the required line argument.

**GoTo 0:** Disables enabled exception handler in the current procedure and resets it to Nothing.

**GoTo -1:** Same as GoTo 0.

**Resume Next:** Specifies that when an exception occurs, execution skips over the statement that caused the problem and goes to the statement immediately following. Execution continues from that point

**Example: Refer Note**

**CREATING SUBPROCEDURES:** Declare Sub procedures with the Sub statement:

*Syntax:*

*[ <attrlist> ] [ { Overloads | Overrides | Overridable | NotOverridable MustOverride | Shadows | Shared } ] [ { Public | Protected | Friend | Protected Friend | Private } ]*

*Sub name [(arglist)]*

*[ statements ]*

*[ Exit Sub ]*

*[ statements ]*

*End Sub*

Here are the parts of this statement:

**attrlist**-List of attributes for this procedure. You separate multiple attributes with commas.

**Overloads**-Specifies that this Sub procedure overloads one (or more) procedures defined with the same name in a base class.

**Overrides**-Specifies that this Sub procedure overrides a procedure with the same name in a base class. Note that the number and data types of the arguments must match those of the procedure in the base class.

**Overridable**-Specifies that this Sub procedure can be overridden by a procedure with the same name in a derived class.

**NotOverridable**-Specifies that this Sub procedure may not be overridden in a derived class.

**MustOverride**-Specifies that this Sub procedure is not implemented. Instead, this procedure must be implemented in a derived class. If it is not, that class will not be creatable.

**Shadows**-Makes this Sub procedure a shadow of an identically named programming element in a base class. A shadowed element is unavailable in the derived class that shadows it.

**Shared**-Specifies that this Sub procedure is a shared procedure. As a shared procedure, it is not associated with a specific instance of a class or structure, and you can call it by qualifying it either with the class or structure name, or with the variable name of a specific instance of the class or structure.

**Public**-Procedures declared Public have public access. There are no restrictions on the accessibility of public procedures.

**Protected**-Procedures declared Protected have protected access. They are accessible only from within their own class or from a derived class.

**Protected Friend**-Procedures declared Protected Friend have both protected and friend accessibility. They can be used by code in the same assembly, as well as by code in derived classes.

**Private**-Procedures declared Private have private access. They are accessible only within their declaration context, including from any nested procedures.

**name**-Name of the Sub procedure.

**arglist**-List of expressions (which can be single variables or simple values) representing arguments that are passed to the Sub procedure when it is called.

**statements**-The block of statements to be executed within the Sub procedure.

Each argument in the arglist part has the following syntax and parts:

[ <attrlist> ] [ **Optional** ] [ { **ByVal** | **ByRef** } ] [ **ParamArray** ] **argn**

[ **As argtype** ] [ = **defaultvalue** ]

Here are the parts of the arglist:

**attrlist**-List of attributes that apply to this argument. Multiple attributes are separated by commas.

**Optional**-Specifies that this argument is not required when the procedure is called.

**ByVal**-Specifies passing by value. In this case, the procedure cannot replace or reassign the underlying variable element in the calling code (unless the argument is a reference type). ByVal is the default in Visual Basic.

**ByRef**-Specifies passing by reference. In this case, the procedure can modify the underlying variable in the calling code the same way the calling code itself can.

**ParamArray**-Used as the last argument in arglist to indicate that the final argument is an optional array of elements of the specified type. The ParamArray keyword allows you to pass an arbitrary number of arguments to the procedure. A ParamArray argument is always passed ByVal.

**argname**-Name of the variable representing the argument.

**argtype**-This part is optional unless Option Strict is set to On, and holds the data type of the argument passed to the procedure. Can be Boolean, Byte, Char, Date, Decimal, Double, Integer, Long, Object, Short, Single, or String; or the name of an enumeration, structure, class, or interface.

**defaultvalue**-Required for Optional arguments. Any constant or constantexpression that evaluates to the data type of the argument. Note that if the type is Object, or a class, interface, array, or structure, the default value can be only Nothing.

Each attribute in the attrlist part has the following syntax and parts:  
<attrname [( { attrargs | attrinit } )]>

Here are the parts of attrlist:

**attrname**-Name of the attribute.

**attrargs**-List of positional arguments for this attribute. Multiple arguments are separated by commas.

**attrinit**-List of field or property initializers for this attribute. Multiple initializers are separated by commas.

## CREATING FUNCTIONS:

### Syntax:

```
[ <attrlist> ] [ { Overloads | Overrides | Overridable | NotOverridable MustOverride | Shadows | Shared } ] [ { Public | Protected | Friend | Protected Friend | Private } ] Function name [(arglist)] [ As type ]
```

```
[ statements ]
```

```
[ Exit Function ]
```

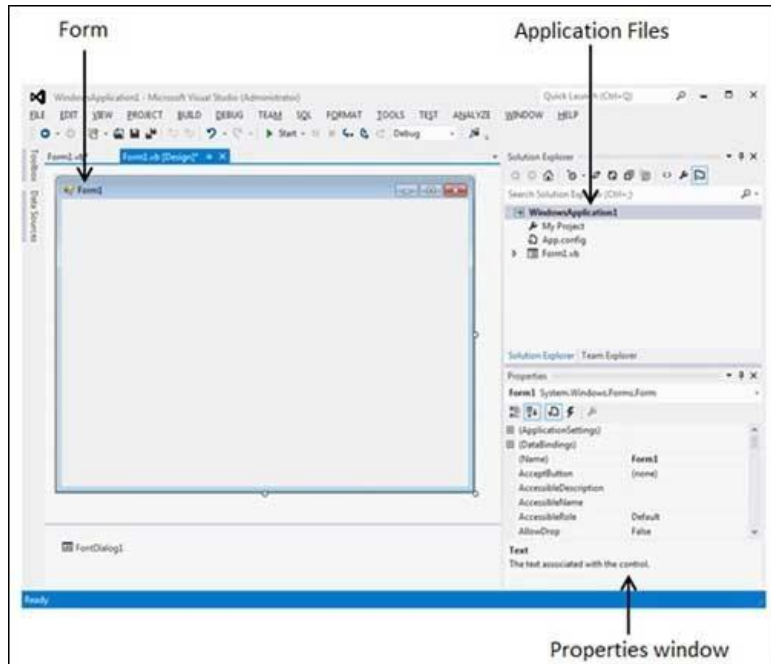
```
[ statements ]
```

```
End Function
```



# WINDOWS FORM APPLICATIONS

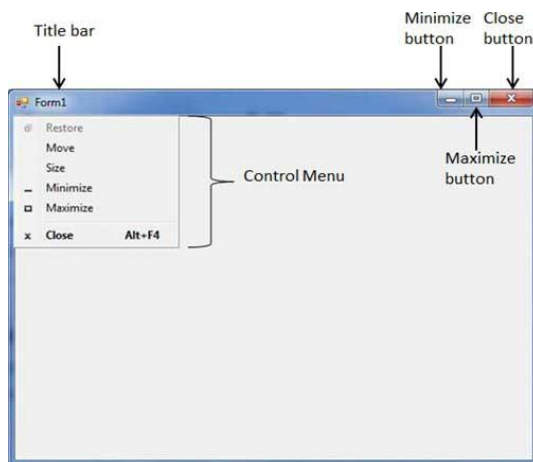
Microsoft Visual Studio - **File** → **New Project** → **Windows Forms Applications**



Visual Basic Form is the container for all the controls that make up the user interface.

Visual Studio creates a default form for you when you create a **Windows Forms Application**.

Every form will have title bar on which the form's caption is displayed and there will be buttons to close, maximize and minimize the form



**Properties:**

Properties	Description
------------	-------------

<b>CancelButton</b>	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.
<b>BackColor</b>	Sets the form background color.
<b>BorderStyle</b>	The BorderStyle property determines the style of the form's border and the appearance of the form – <ul style="list-style-type: none"> <li>• <b>None</b> – Borderless window that can't be resized.</li> <li>• <b>Fixed3D</b> – Window with a visible border, "raised" relative to the main area. In this case, windows can't be resized.</li> <li>• <b>SizableToolWindow</b> – Same as the FixedToolWindow but resizable. In addition, its caption font is smaller than the usual.</li> </ul>
<b>Enabled</b>	If True, allows the form to respond to mouse and keyboard events; if False, disables form.
<b>Font</b>	This property specify font type, style, size
<b>MinimizeBox</b>	By default, this property is True and you can set it to False to hide the Minimize button on the title bar.
<b>MaximizeBox</b>	By default, this property is True and you can set it to False to hide the Maximize button on the title bar.
<b>MinimumSize</b>	This specifies the minimum height and width of the window you can minimize.
<b>MaximumSize</b>	This specifies the maximum height and width of the window you maximize.
<b>Name</b>	This is the actual name of the form.
<b>Text</b>	The text, which will appear at the title bar of the form.
<b>Top, Left</b>	These two properties set or return the coordinates of the form's top-left corner in pixels.
<b>TopMost</b>	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.
<b>Width</b>	This is the width of the form in pixel.

## Methods

Method Name & Description
---------------------------

<b>Activate:</b> Activates the form and gives it focus.
<b>ActivateMdiChild:</b> Activates the MDI child of a form.
<b>AddOwnedForm:</b> Adds an owned form to this form.
<b>BringToFront:</b> Brings the control to the front of the z-order.
<b>CenterToParent:</b> Centers the position of the form within the bounds of the parent form.
<b>CenterToScreen:</b> Centers the form on the current screen.
<b>Close:</b> Closes the form.
<b>Contains:</b> Retrieves a value indicating whether the specified control is a child of the control.
<b>Focus:</b> Sets input focus to the control.
<b>Hide:</b> Conceals the control from the user.
<b>Refresh:</b> Forces the control to invalidate its client area and immediately redraw itself and any child controls.
<b>Show:</b> Displays the control to the user.
<b>ShowDialog:</b> Shows the form as a modal dialog box.

**Events:**

<b>Event</b>	<b>Description</b>
<b>Activated</b>	Occurs when the form is activated in code or by the user.
<b>Click</b>	Occurs when the form is clicked.
<b>Closed</b>	Occurs before the form is closed.
<b>Closing</b>	Occurs when the form is closing.
<b>DoubleClick</b>	Occurs when the form control is double-clicked.
<b>DragDrop</b>	Occurs when a drag-and-drop operation is completed.
<b>Enter</b>	Occurs when the form is entered.
<b>KeyDown</b>	Occurs when a key is pressed while the form has focus.
<b>KeyPress</b>	Occurs when a key is pressed while the form has focus.
<b>KeyUp</b>	Occurs when a key is released while the form has focus.
<b>MouseDown</b>	Occurs when the mouse pointer is over the form and a mouse button is pressed.
<b>MouseEnter</b>	Occurs when the mouse pointer enters the form.
<b>MouseHover</b>	Occurs when the mouse pointer rests on the form.
<b>MouseLeave</b>	Occurs when the mouse pointer leaves the form.
<b>MouseMove</b>	Occurs when the mouse pointer is moved over the form.

<b>MouseUp</b>	Occurs when the mouse pointer is over the form and a mouse button is released.
<b>MouseWheel</b>	Occurs when the mouse wheel moves while the control has focus.

### Example: Write any Program using Windows Application

#### CREATING WINDOWS APPLICATIONS:

*WindowsApp.vbproj*: A Visual Basic project.

*AssemblyInfo.vb*: General Information about an assembly, including version information.

*Form1.vb*: A form's code file.

*Form1.resx.NET*: An XML-based resource template.

*WindowsApp.vbproj.user*: Stores project user options.

*WindowsApp.sln*: The solution file, storing the solution's configuration.

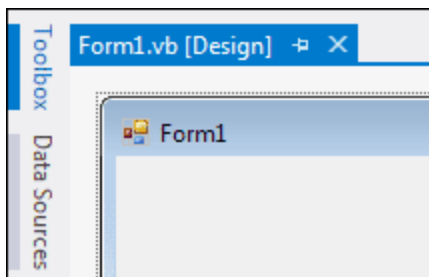
*WindowsApp.suo*: Stores solution user options.

*bin*: Directory for binary executables.

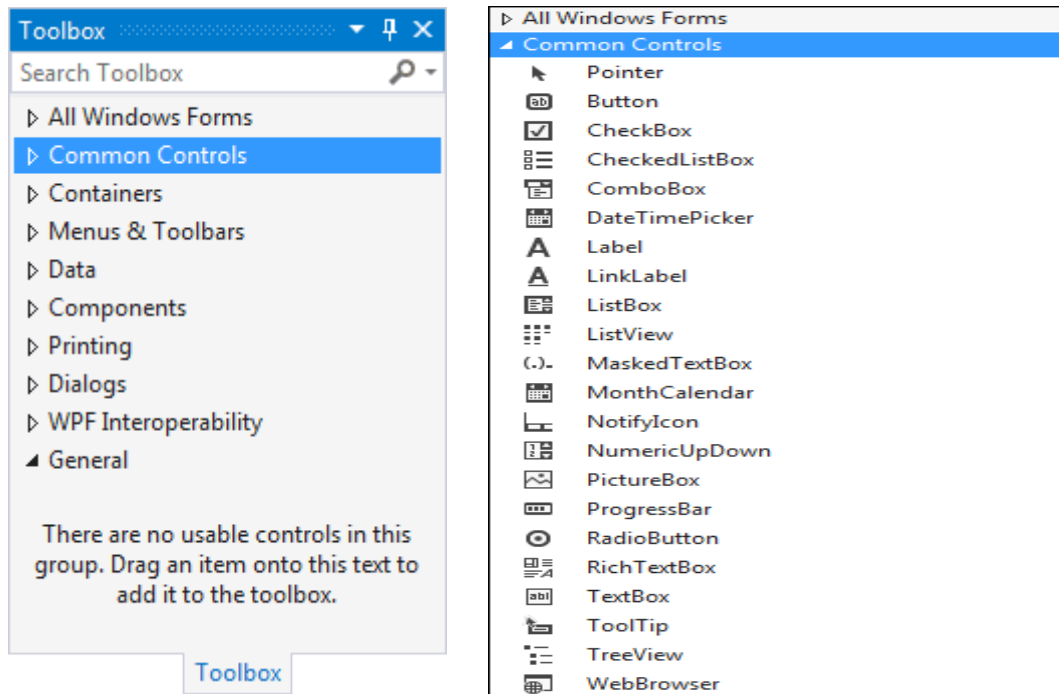
*obj*: Directory for debugging binaries.

#### ADDING CONTROLS TO THE FORM

The Toolbox can be found on the left of the screen. In the picture below, you can see the toolbox icon next to Form1



To display all the tools, click on the Toolbox tab.



## THE MSGBOX () FUNCTION

The objective of MsgBox is to produce a pop-up message box and prompt the user to click on a command button





**Syntax :** *yourMsg=MsgBox(Prompt, Style Value, Title)*

The first argument, Prompt, will display the message in the message box. The Style Value will determine what type of command buttons appear on the message box,

Style Value	Named Constant	Buttons Displayed
0	vbOkOnly	Ok button
1	vbOkCancel	Ok and Cancel buttons
2	vbAbortRetryIgnore	Abort, Retry and Ignore buttons.
3	vbYesNoCancel	Yes, No and Cancel buttons
4	vbYesNo	Yes and No buttons
5	vbRetryCancel	Retry and Cancel buttons

`yourMsg=MsgBox( "Click OK to Proceed", 1, "Startup Menu")` and  
`yourMsg=Msg("Click OK to Proceed". vbOkCancel,"Startup Menu")` are the same.

Value	Named Constant	Button Clicked
1	vbOk	Ok button
2	vbCancel	Cancel button
3	vbAbort	Abort button
4	vbRetry	Retry button
5	vbIgnore	Ignore button
6	vbYes	Yes button
7	vbNo	No button

Value	Named Constant	Icon
16	vbCritical	
3	vbQuestion	
48	vbExclamation	
64	vbInformation	

*Example: Refer Note*

## MESSAGEBOX.SHOW METHOD

The .NET framework's MessageBox class's Show method to display message boxes. This method has many overloaded forms; here's one of them:

### Syntax

*Overloads Public Shared Function Show(ByVal text As String, ByVal caption As String, ByVal buttons As MessageBoxButtons, ByVal icon As MessageBoxIcon, ByVal defaultButton As MessageBoxDefaultButton, ByVal options As MessageBoxOptions )As DialogResult*

Text: The text to display in the message box.

Caption: The text to display in the title bar of the message box.

**Buttons:** One of the `MessageBoxButtons` enumeration values that specifies which buttons to display in the message box.

**Icon:** One of the `MessageBoxIcon` enumeration values that specifies which icon to display in the message box.

**defaultButton:** One of the `MessageBoxDefaultButton` enumeration values that specifies which is the default button for the message box.

**options:** One of the `MessageBoxOptions` enumeration values that specifies which display and association options will be used for the message box.

**Here are the `MessageBoxButtons` enumeration values:**

**AbortRetryIgnore**— The message box will show Abort, Retry, and Ignore buttons.

**OK**— The message box will show an OK button.

**OKCancel**— The message box will show OK and Cancel buttons.

**RetryCancel**— The message box will show Retry and Cancel buttons.

**YesNo**— The message box will show Yes and No buttons.

**YesNoCancel**— The message box will show Yes, No, and Cancel buttons.

**Here are the `MessageBoxIcon` enumeration values:**

**Asterisk**— Shows an icon displaying a lowercase letter i in a circle.

**Error**— Shows an icon displaying a white X in a circle with a red background.

**Exclamation**— Shows an icon displaying an exclamation point in a triangle with a yellow background.

**Hand**— Shows an icon displaying a white X in a circle with a red background.

**Information**— Shows an icon displaying a lowercase letter i in a circle.

**None**— Shows no icons.

**Question**— Shows an icon displaying a question mark in a circle.

**Stop**— Shows an icon displaying white X in a circle with a red background.

**Warning**— Shows an icon displaying an exclamation point in a triangle with a yellow background.

**Here are the `MessageBoxDefaultButton` enumeration values:**

**Button1**— Makes the first button on the message box the default button.

**Button2**— Makes the second button on the message box the default button.

**Button3**— Makes the third button on the message box the default button.

**Here are the `MessageBoxOptions` enumeration values:**

**DefaultDesktopOnly**— Displays the message box on the active desktop.

**RightAlign**— The message box text is right-aligned.

*Example: Refer Note*

### **The InputBox( ) Function**

An InputBox( ) function will display a message box where the user can enter a value or a message in the form of text

```
Syntax : Public Function InputBox(Prompt As String [, Title As String = "" [, DefaultResponse As String = "" [, XPos As Integer = -1 [, YPos As Integer = -1]]) As String
```

Prompt - the message displayed normally as a question asked.

Title - The title of the Input Box.

default-text - The default text that appears in the input field where users can use it as his intended input or he may change to the message he wish to enter.

x-position and y-position - the position or the coordinates of the input box.

*Example: Refer Note*

### **WORKING WITH MULTIPLE FORMS:**

VB.Net allow working with multiple forms.

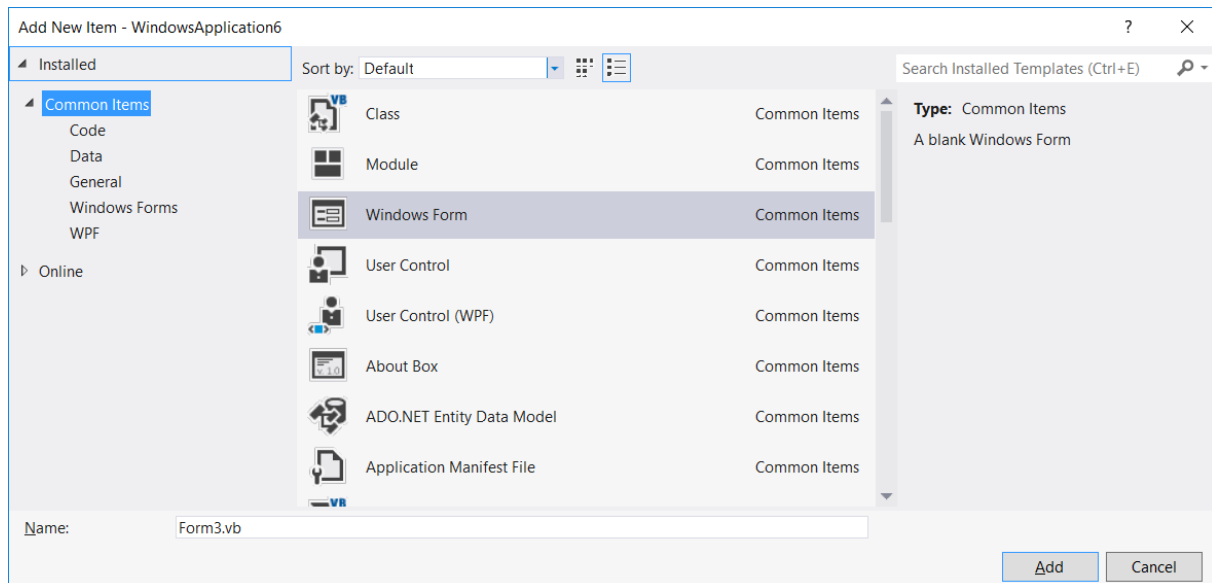
A Windows application can either be a Single Form application or multiple form application.

### **How to Create a New Form in Visual Studio?**

Follow the below steps to create a new form in Visual Studio VB.Net application.

**Project -> Add New Item -> Windows Form**





*Example:*

*Public Class Form1*

*Private Sub Form1\_Load(sender As Object, e As EventArgs) Handles MyBase.Load*

*End Sub*

*Private Sub Button1\_Click(sender As Object, e As EventArgs) Handles Button1.Click*

*Form2.Show()*

*End Sub*

*End Class*

## **EVENT HANDLING**

Events are basically a user action like key press, clicks, mouse movements, etc., or some occurrence like system generated notifications.

Applications need to respond to events when they occur.

An event is an action that calls a function or may cause another event. Event handlers are functions that tell how to respond to an event.

VB.Net is an event-driven language. There are mainly two types of events –

- Mouse events
- Keyboard events

### **Handling Mouse Events**

Mouse events occur with mouse movements in forms and controls. Following are the various mouse events related with a Control class –

- **MouseDown** – it occurs when a mouse button is pressed
- **MouseEnter** – it occurs when the mouse pointer enters the control
- **MouseHover** – it occurs when the mouse pointer hovers over the control
- **MouseLeave** – it occurs when the mouse pointer leaves the control
- **MouseMove** – it occurs when the mouse pointer moves over the control
- **MouseUp** – it occurs when the mouse pointer is over the control and the mouse button is released
- **MouseWheel** – it occurs when the mouse wheel moves and the control has focus

The event handlers of the mouse events get an argument of type **MouseEventArgs**. The **MouseEventArgs** object is used for handling mouse events. It has the following properties –

- **Buttons** – indicates the mouse button pressed
- **Clicks** – indicates the number of clicks
- **Delta** – indicates the number of detents the mouse wheel rotated
- **X** – indicates the x-coordinate of mouse click
- **Y** – indicates the y-coordinate of mouse click

### **Example: Refer Note**

### **Handling Keyboard Events**

- **KeyDown** – occurs when a key is pressed down and the control has focus
- **KeyPress** – occurs when a key is pressed and the control has focus
- **KeyUp** – occurs when a key is released while the control has focus

The event handlers of the **KeyDown** and **KeyUp** events get an argument of type **KeyEventArgs**. This object has the following properties –

- **Alt** – it indicates whether the ALT key is pressed
- **Control** – it indicates whether the CTRL key is pressed
- **Handled** – it indicates whether the event is handled
- **KeyCode** – stores the keyboard code for the event
- **KeyData** – stores the keyboard data for the event

- **KeyValue** – stores the keyboard value for the event
- **Modifiers** – it indicates which modifier keys (Ctrl, Shift, and/or Alt) are pressed
- **Shift** – it indicates if the Shift key is pressed

The event handlers of the KeyDown and KeyUp events get an argument of type **KeyEventArgs**. This object has the following properties –

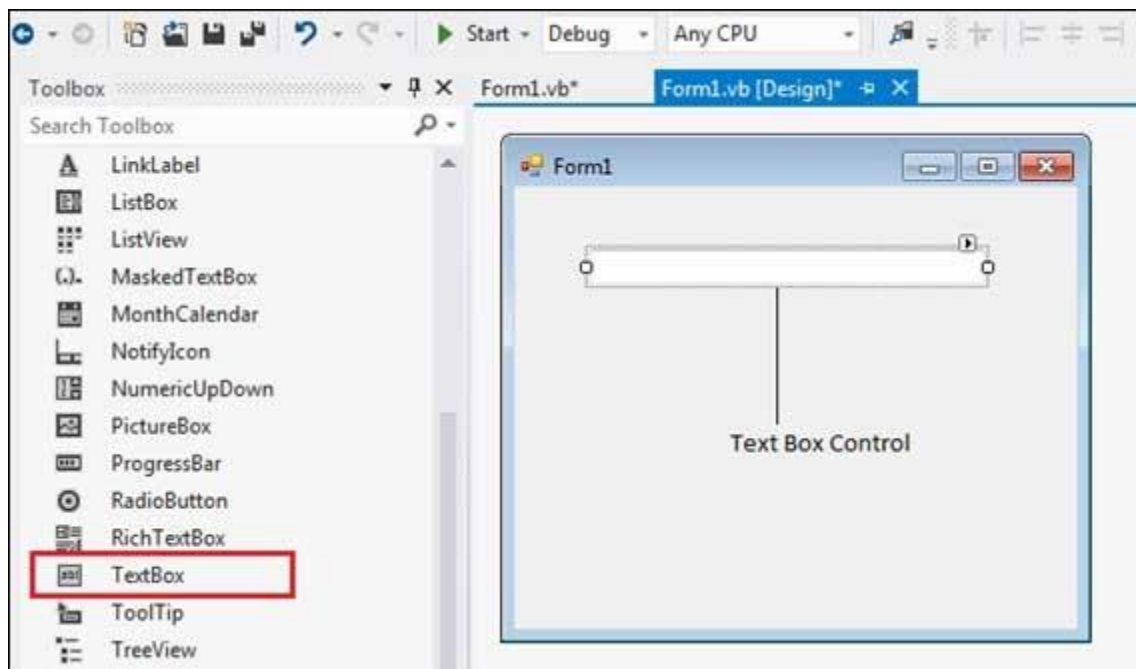
- **Handled** – indicates if the KeyPress event is handled
- **KeyChar** – stores the character corresponding to the key pressed

### Example: Refer Note

### TEXT BOX CONTROL:

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.



### Properties:

Property & Description
<b>Font:</b> Gets or sets the font of the text displayed by the control.
<b>FontHeight:</b> Gets or sets the height of the font of the control.

<b>ForeColor:</b> Gets or sets the foreground color of the control.
<b>Lines:</b> Gets or sets the lines of text in a text box control.
<b>Multiline:</b> Gets or sets a value indicating whether this is a multiline TextBox control.
<b>PasswordChar:</b> Gets or sets the character used to mask characters of a password in a single-line TextBox control.
<b>ReadOnly:</b> Gets or sets a value indicating whether text in the text box is read-only.
<p><b>ScrollBars:</b>Gets or sets which scroll bars should appear in a multiline TextBox control. This property has values –</p> <ul style="list-style-type: none"> <li>• None</li> <li>• Horizontal</li> <li>• Vertical</li> <li>• Both</li> </ul>
<b>TabIndex:</b> Gets or sets the tab order of the control within its container.
<b>Text:</b> Gets or sets the current text in the TextBox.
<p><b>TextAlign:</b>Gets or sets how text is aligned in a TextBox control. This property has values –</p> <ul style="list-style-type: none"> <li>• Left</li> <li>• Right</li> <li>• Center</li> </ul>
<b>TextLength:</b> Gets the length of text in the control.
<b>WordWrap:</b> Indicates whether a multiline text box control automatically wraps words to the beginning of the next line when necessary.

### Methods:

Method Name & Description
<b>AppendText:</b> Appends text to the current text of a text box.
<b>Clear:</b> Clears all text from the text box control.
<b>Copy:</b> Copies the current selection in the text box to the <b>Clipboard</b> .
<b>Cut:</b> Moves the current selection in the text box to the <b>Clipboard</b> .
<b>Paste:</b> Replaces the current selection in the text box with the contents of the <b>Clipboard</b> .
<b>Paste(String):</b> Sets the selected text to the specified text without clearing the undo buffer.
<b>ResetText:</b> Resets the Text property to its default value.

<b>ToString:</b> Returns a string that represents the TextBoxBase control.
<b>Undo:</b> Undoes the last edit operation in the text box.

## Events

Event & Description
<b>Click:</b> Occurs when the control is clicked.
<b>DoubleClick:</b> Occurs when the control is double-clicked.
<b>TextAlignChanged:</b> Occurs when the TextAlign property value changes.

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

```
    TextBox1.Text = "Hello"
```

```
End Sub
```

### i) Creating Multiline,WordWrap Textboxes:

**Multiline:**Gets or sets a value indicating whether this is a multiline TextBox control.

**WordWrap:**Indicates whether a multiline text box control automatically wraps words to the beginning of the next line when necessary.

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

```
    TextBox1.Multiline = True
```

```
    TextBox1.Text = "Line 1"
```

```
    TextBox1.Text = TextBox1.Text & ControlChars.NewLine & "Line 2"
```

```
End Sub
```

### ii) Accessing Text In A Textbox:

```
Private Sub Button1_click_1(ByVal sender as System.Object,ByVal e As System.EventArgs)Handles  
Button1.Click
```

```
    TextBox1.Text="Hello"
```

```
    Dim Str As String
```

```
    Str=TextBox1.Text
```

```
    TextBox2.Text=Str
```

```
End Sub
```

**iii) Adding Scrollbar to a textbox:** Gets or sets which scroll bars should appear in a multiline TextBox control. This property has values “None,Horizontal,Vertical,Both”

*TextBox1.ScrollBars = ScrollBars.Vertical;*

If you want both scroll bars(vertical and horizontal) you should set ..

*TextBox2.ScrollBars = ScrollBars.Both;*

#### **iv) Aligning Text in textbox:**

Gets or sets how text is aligned in a TextBox control. This property has values –

- Left
- Right
- Center
- `textbox1.TextAlign=HorizontalAlignment.Center`

#### **RICH TEXTBOX:**

RichTextBox Control allows user to display, input, edit and format text information. RichTextBox Control supports advance formatting features as compared to TextBox. Using RichTextBox user can format only selected portion of the text. User can also format paragraph using RichTextBox Control. RichTextBox Control also allows user to save as well as load file of RTF format and Standard ASCII format.

Property	Purpose
BackColor	It is used to get or set background color of the RichTextBox.
Font	It is used to set Font Face, Font Style, Font Size and Effects of the text associated with RichTextBox Control.
ForeColor	It is used to get or set Fore color of the text associated with RichTextBox Control.
Multiline	It is used to specify whether RichTextBox can be expanded to enter more than one line of text or not. It has Boolean value. Default value is true.
ScrollBars	It is used to get or set type of scrollbars to be added with RichTextBox control. It has following 4 options: (1) None (2) Horizontal (2) Vertical

	(3) Default value is Both.	Both
Size	It is used to get or set height and width of RichTextBox control in pixel.	
Text	It is used to get or set text associated with RichTextBox Control.	
SelectionAlignment	It is used to get or set horizontal alignment of the text selected in RichTextBox.	
SelectionBackColor	It is used to get or set BackColor of the text selected in RichTextBox.	
SelectionColor	It is used to get or set Fore Color of the text selected in RichTextBox.	
SelectionFont	It is used to get or set font face, font style, and font size of the text selected in RichTextBox.	
SelectionLength	It is used to get or set number of characters selected in the RichTextBox.	
SelectionStart	It is used to get or set starting point of the text selected in the RichTextBox.	
Cut	It is used to move current selection of RichTextBox into clipboard.	
Copy	It is used to copies selected text of RichTextBox in clipboard.	
Paste	It is used to replace current selection of TextBox by contents of clipboard. It is also used to move contents of Clipboard to RichTextBox control where cursor is currently located.	

*Private Sub OpenToolStripMenuItem Click(...) Handles OpenToolStripMenuItem.Click*

*If DiscardChanges() Then*

*OpenFileDialog1.Filter = "RTF Files/\*.RTF/DOC Files/\*.DOC|" &"Text Files/\*.TXT|All Files/\*.\*"*

*If OpenFileDialog1.ShowDialog() = DialogResult.OK Then*

*fName = OpenFileDialog1.FileName*

*Editor.LoadFile(fName)*

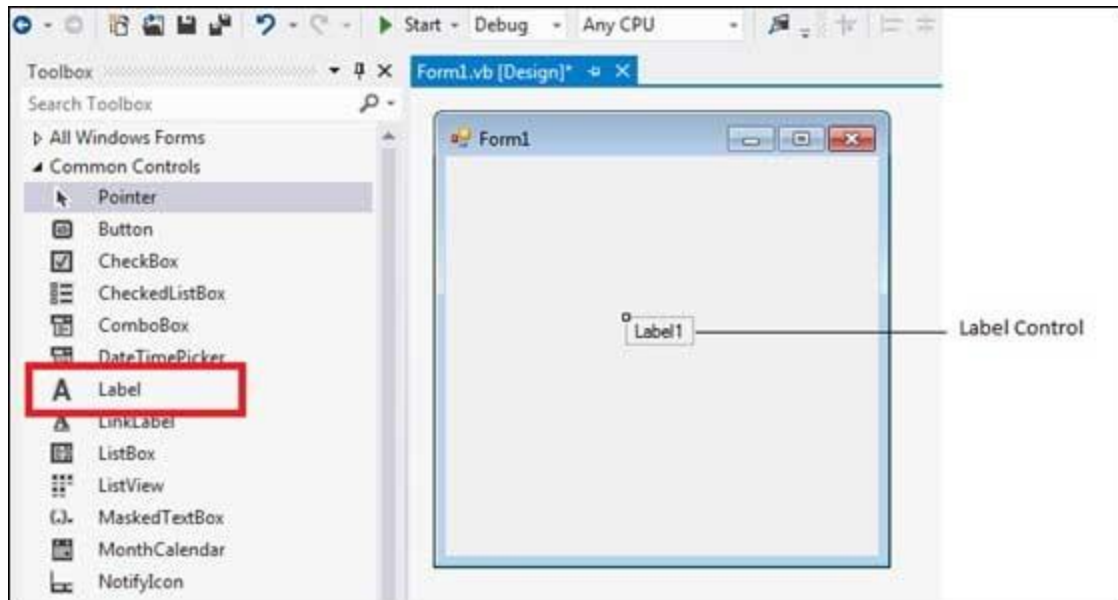
*Editor.Modified = False*

*End If*

*End If*

*End Sub*

**LABELS:** 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.



Property & Description
<b>AutoSize:</b> Gets or sets a value specifying if the control should be automatically resized to display all its contents.
<b>BorderStyle:</b> Gets or sets the border style for the control.
<b>FlatStyle:</b> Gets or sets the flat style appearance of the Label control
<b>Font:</b> Gets or sets the font of the text displayed by the control.
<b>FontHeight:</b> Gets or sets the height of the font of the control.
<b>ForeColor:</b> Gets or sets the foreground color of the control.
<b>Text:</b> Gets or sets the text associated with this control.
<b>TextAlign:</b> Gets or sets the alignment of text in the label.

## Events

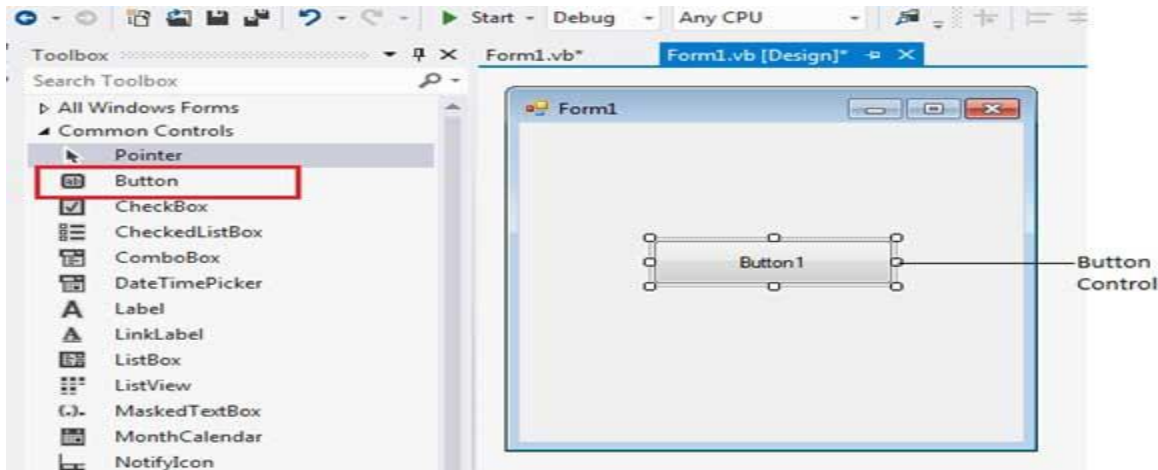
Event & Description
<b>Click:</b> Occurs when the control is clicked.
<b>DoubleClick:</b> Occurs when the control is double-clicked.
<b>Leave:</b> Occurs when the input focus leaves the control.
<b>LostFocus:</b> Occurs when the control loses focus.
<b>TabIndexChanged:</b> Occurs when the TabIndex property value changes.
<b>TabStopChanged:</b> Occurs when the TabStop property changes.
<b>TextChanged:</b> Occurs when the Text property value changes.



## UNIT 3

### 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.



### *Properties*

**AutoSizeMode:**Gets or sets the mode by which the Button automatically resizes itself.

**BackColor:**Gets or sets the background color of the control.

**BackgroundImage:**Gets or sets the background image displayed in the control.

**DialogResult:**Gets or sets a value that is returned to the parent form when the button is clicked.

**ForeColor:**Gets or sets the foreground color of the control.

**Image:**Gets or sets the image that is displayed on a button control.

**Location:**Gets or sets the coordinates of the upper-left corner of the control relative to the upper-left corner of its container.

**TabIndex:**Gets or sets the tab order of the control within its container.

**Text:**Gets or sets the text associated with this control.

### *Methods*

**PreferredSize:**Retrieves the size of a rectangular area into which a control can be fitted.

**NotifyDefault:**Notifies the Button whether it is the default button so that it can adjust.

**Select:**Activates the control.

**ToString:**Returns a String containing the name of the Component, if any.

## Events

**Click:**Occurs when the control is clicked.

**DoubleClick:**Occurs when the user double-clicks the Button control.

**GotFocus:**Occurs when the control receives focus.

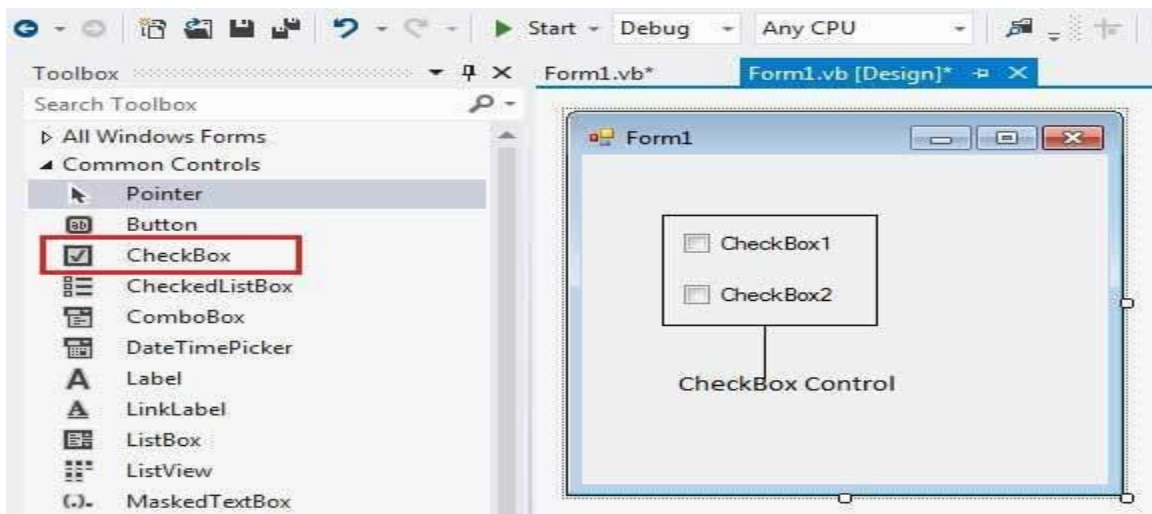
**TabIndexChanged:**Occurs when the TabIndex property value changes.

**TextChanged:**Occurs when the Text property value changes.

**Validated:**Occurs when the control is finished validating.

## CHECKBOX

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.



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

**Appearance:**Gets or sets a value determining the appearance of the check box.

**AutoCheck:**Gets or sets a value indicating whether the Checked or CheckState value .

**CheckAlign:**Gets or sets the horizontal and vertical alignment of the check mark on the check.

**Checked:**Gets or sets a value indicating whether the check box is selected.

**CheckState:**Gets or sets the state of a check box.

**Text:**Gets or sets the caption of a check box.

**ThreeState:**Gets or sets a value indicating whether or not a check box should allow three check states rather than two.

### Methods

**OnCheckedChanged:**Raises the CheckedChanged event.

**OnCheckStateChanged:**Raises the CheckStateChanged event.

**OnClick:**Raises the OnClick event.

### Events

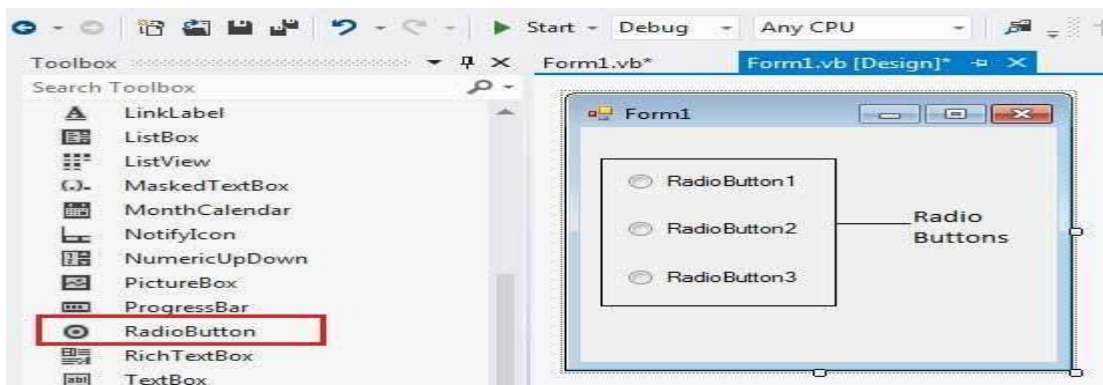
**AppearanceChanged:**Occurs when the value of the Appearance property of the check box is hanged.

**CheckedChanged:**Occurs when the value of the Checked property of the CheckBox control is hanged.

**CheckStateChanged:**Occurs when the value of the CheckState property of the CheckBox control is hanged.

## RADIO BUTTON

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.



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***

**Appearance:**Gets or sets a value determining the appearance of the radio button.

**AutoCheck:**Gets or sets a value indicating whether the Checked value and the appearance of the control automatically change when the control is clicked.

**CheckAlign:**Gets or sets the location of the check box portion of the radio button.

**Checked:**Gets or sets a value indicating whether the control is checked.

**Text:**Gets or sets the caption for a radio button.

**TabStop:**Gets or sets a value indicating whether a user can give focus to the RadioButton control using the TAB key.

### ***Methods***

**PerformClick:**Generates a Click event for the control, simulating a click by a user.

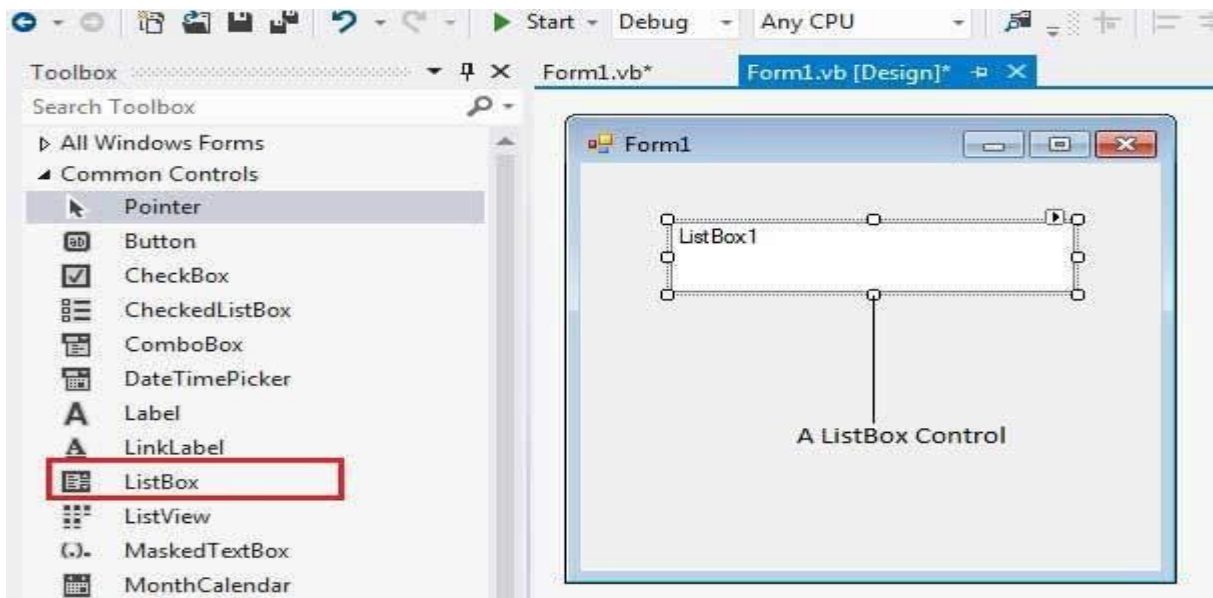
### ***Events***

**AppearanceChanged:**Occurs when the value of the Appearance property of the RadioButton control is changed.

**CheckedChanged:**Occurs when the value of the Checked property of the RadioButton control changed.

## **LIST BOX**

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.



## *Properties*

**Items:** Gets the items of the list box.

**MultiColumn:** Gets or sets a value indicating whether the list box supports multiple columns.

**SelectedIndex:** Gets or sets the zero-based index of the currently selected item in a list box.

**SelectedItem:** Gets or sets the currently selected item in the list box.

**SelectionMode:** Gets or sets the method in which items are selected in the list box. This property has values:

- None
- One
- MultiSimple
- MultiExtended

**Sorted:** Gets or sets a value indicating whether the items in the list box are sorted alphabetically.

**Text:** Gets or searches for the text of the currently selected item in the list box.

**TopIndex:** Gets or sets the index of the first visible item of a list box.

## *Methods of the ListBox Control*

**ClearSelected:** Unselects all items in the ListBox.

**EndUpdate:** Resumes drawing of a list box after it was turned off by the BeginUpdate method.

**FindString:** Finds the first item in the ListBox that starts with the string specified as an argument.

**GetSelected:** Returns a value indicating whether the specified item is selected.

**SetSelected:** Selects or clears the selection for the specified item in a ListBox.

**OnSelectedIndexChanged:** Raises the SelectedIndexChanged event.

**OnSelectedValueChanged:** Raises the SelectedValueChanged event.

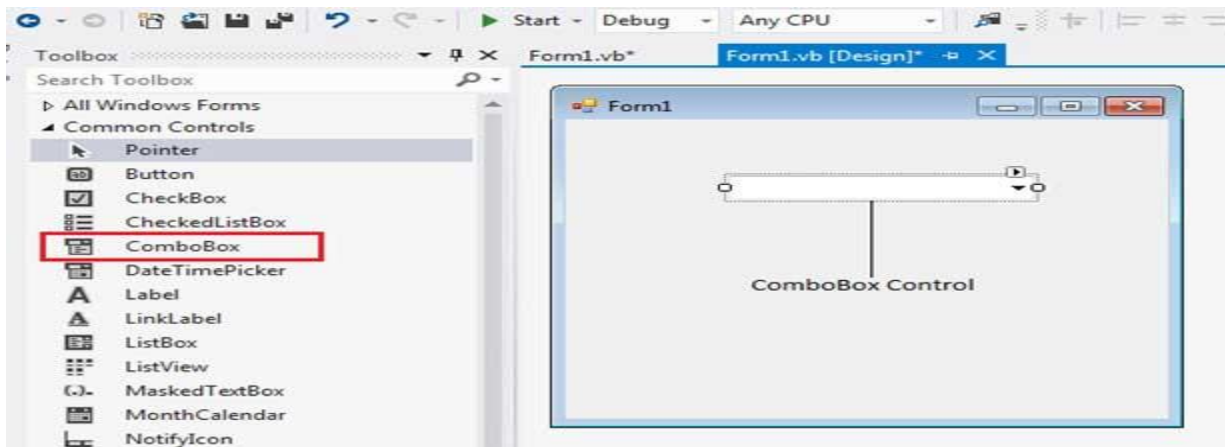
## *Events*

**Click:**Occurs when a list box is selected.

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

## 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.



### *Properties*

**Items:**Gets an object representing the collection of the items contained in this ComboBox.

**SelectedIndex:**Gets or sets the index specifying the currently selected item.

**SelectedItem:**Gets or sets currently selected item in the ComboBox.

**SelectionLength:**Gets or sets the number of characters selected in the editable portion of the combo box.

**Sorted:**Gets or sets a value indicating whether the items in the combo box are sorted.

**Text:**Gets or sets the text associated with this control.

### *Methods*

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

**FindStringExact:**Finds the first item in the combo box that exactly matches the specified string.

**SelectAll:**Selects all the text in the editable area of the combo box.

### *Events*

**DropDown:**Occurs when the drop-down portion of a combo box is displayed.

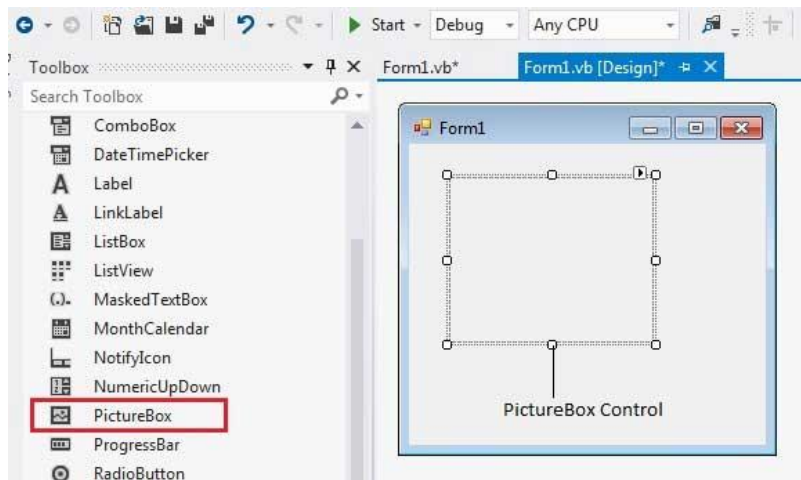
**DropDownClosed:**Occurs when the drop-down portion of a combo box is no longer visible.

**DropDownStyleChanged:**Occurs when the DropDownStyle property of the ComboBox has changed.

**SelectedIndexChanged:**Occurs when the SelectedIndex property of a ComboBox control has changed.

**SelectionChangeCommitted:**Occurs when the selected item has changed and the change appears in the combo box.

**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.



## *Properties*

**Image:**Gets or sets the image that is displayed in the control.

**SizeMode:**Determines the size of the image to be displayed in the control. This property takes its value from the PictureBoxSizeMode enumeration, which has values:

- **Normal** - the upper left corner of the image is placed at upper left part of the picture box
- **StretchImage** - 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.

**TabIndex:**Gets or sets the tab index value.

**TabStop:**Specifies whether the user will be able to focus on the picture box by using the TAB key.

**Text:**Gets or sets the text for the picture box.

## ***Methods***

**CancelAsync:** Cancels an asynchronous image load.

**Load:** Displays an image in the picture box

**LoadAsync:** Loads image asynchronously.

**ToString:** Returns the string that represents the current picture box.

## ***Events***

**Click:** Occurs when the control is clicked.

**Enter:** Overrides the Control.Enter property.

**ForeColorChanged:** Occurs when the value of the ForeColor property changes.

**KeyDown:** Occurs when a key is pressed when the control has focus.

**KeyPress:** Occurs when a key is pressed when the control has focus.

**KeyUp:** Occurs when a key is released when the control has focus.

**Leave:** Occurs when input focus leaves the PictureBox.

## ***Public Class Form1***

*Private Sub Form1\_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load*

*PictureBox1.Image = Image.FromFile("d:\testImage.jpg")*

*PictureBox1.SizeMode = PictureBoxSizeMode.StretchImage*

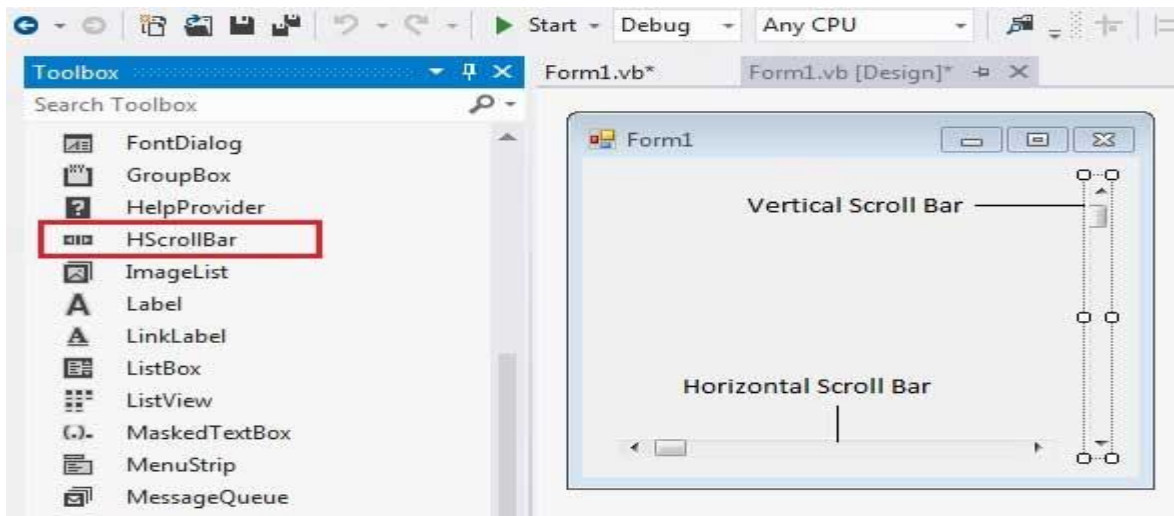
*End Sub*

*End Class*

## **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.





### **Properties**

**AutoSize:**Gets or sets a value indicating whether the ScrollBar is automatically resized to fit its contents.

**BackColor:**Gets or sets the background color for the control.

**ForeColor:**Gets or sets the foreground color of the scroll bar control.

**Maximum:**Gets or sets the upper limit of values of the scrollable range.

**Minimum:**Gets or sets the lower limit of values of the scrollable range.

### **Methods**

**OnClick :**Generates the Click event.

**Select:**Activates the control.

### **Events**

**Click:**Occurs when the control is clicked.

**DoubleClick:**Occurs when the user double-clicks the control.

**Scroll:**Occurs when the control is moved.

**ValueChanged:**Occurs when the Value property changes, either by handling the Scroll event or programmatically.

*Example: Public Class Form1*

*Private Sub Form1\_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load*

*TextBox1.Multiline = True*

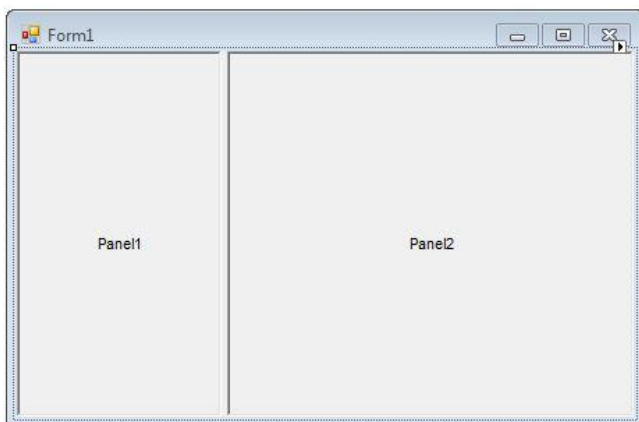
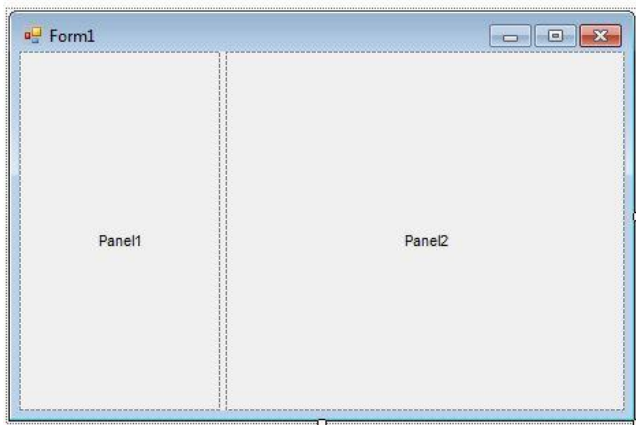
*TextBox1.ScrollBars = ScrollBars.Both*

*End Sub End Class*

## SPLITTERS:

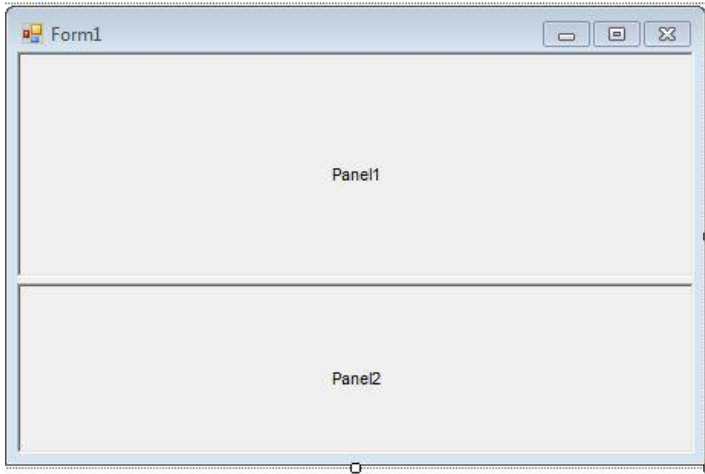
SplitContainer control provides functionality of a splitter to divide and resize two controls.

**Creating a SplitContainer:** A SplitContainer has two built-in panels and a splitter.



## Orientation

Panels on a SplitContainer can be placed horizontally or vertically.



```
splitContainer1.Orientation = Orientation.Horizontal
```

## Panel1 and Panel2

A SplitContainer has two panels. The first panel is represented by Panel1 and second panel is represented by Panel2. These panels are a type of SplitterPanel and can have their own properties and events.

```
Dim leftPanel As SplitterPanel = SplitContainer1.Panel1
```

```
leftPanel.BackColor = Color.Green
```

```
leftPanel.ForeColor = Color.Yellow
```

```
Dim rightPanel As SplitterPanel = SplitContainer1.Panel2
```

```
rightPanel.BackColor = Color.OrangeRed
```

```
rightPanel.ForeColor = Color.White
```

```
splitContainer1.Panel1MinSize = 10
```

```
splitContainer1.Panel2MinSize = 50
```

```
splitContainer1.SplitterDistance = 50
```

```
splitContainer1.SplitterIncrement = 10
```

```
splitContainer1.SplitterWidth = 10
```

## Properties

SplitterDistance property gets or sets the location of the splitter, in pixels, from the left or top edge of the SplitContainer.

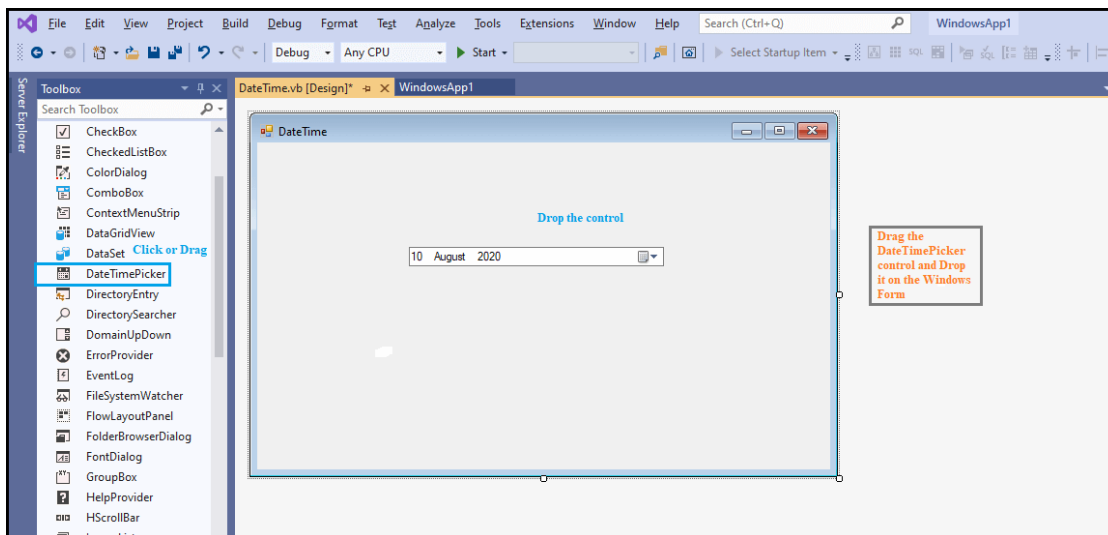
SplitterIncrement property gets or sets a value representing the increment of splitter movement in pixels.

SplitterRectangle property gets the size and location of the splitter relative to the SplitContainer.

SplitterWidth property Gets or sets the width of the splitter in pixels.

**PICKER CONTROL:** Two types i) DateTimePicker Control ii) MonthCalendar Control

i) **DateTimePicker Control:** The DateTimePicker control allows the user to select or display date and time values with a specified format in Windows Forms. Furthermore, we can determine the current date and time using the Value property of the DateTimePicker control. By default, the Value property returns the current date and time in the DateTimePicker.



**Properties:**

Property	Description
<b>BackgroundImage</b>	It is used to set the background image for the DateTimePicker control.
<b>CalendarFont</b>	It is used to set the font style for the calendar in the DateTimePicker control.
<b>CustomFormat</b>	The CustomFormat property is used to set the custom date and time format string in the DateTimePicker control.
<b>Controls</b>	It is used to obtain the collection of controls that are stored within the DateTimePicker control.
<b>Checked</b>	A checked property is used to check whether the value property is checked with a valid date and time in the DateTimePicker control.
<b>Format</b>	The Format property of the DateTimePicker is used to set the format for the Date and time displayed in the Windows Form.
<b>MaxDate</b>	The MaxDate property of the DateTimePicker is used to set the max data and

	time in control selected by the user.
<b>Name</b>	The Name property of the DateTimePicker control allows the user to set the name of the control.
<b>MinimumDateTime</b>	It is used to set the minimum date value that can be allowed by control.

### *Methods*

<b>Method</b>	<b>Description</b>
<b>Contains(Control)</b>	It is used to validate whether the specified control is a child of the DateTimePicker control or not.
<b>CreateControl()</b>	It is used to force the creation of visible control to handle the creation and any visible child controls.
<b>GetAutoSizeMode()</b>	The GetAutoSizeMode() method is used to check the behavior of the DateTimePicker control when the AutoAize property is enabled.
<b>ResetBackColor()</b>	It is used to reset the back color of the DateTimePicker control.
<b>Select()</b>	The Select() method is used to start or activate the DateTimePicker control.
<b>Show()</b>	The Show() method is used to display the control to the user.
<b>ToString()</b>	The ToString() method is used to return a string that represents the current DateTimePicker control.

*Private Sub Button1\_Click(sender As Object, e As EventArgs) Handles Button1.Click*

*Dim dp As Date = DateTimePicker1.Value*

*Dim dp2 As Date = DateTimePicker2.Value*

*Dim result As TimeSpan = dp.Subtract(dp2)*

*Dim ds As Integer = result.TotalDays*

*TextBox1.Text = ds*

*TextBox1.ForeColor = ForeColor.Red*

*MsgBox(" Days = " & ds)*

*End Sub*

*End Class*

**ii) MonthCalendar Control:**

<b>Properties</b>	<b>Description</b>
AnnuallyBodedDate	Property holds an array of DateTime objects specifying which days should be bold.
BoldedDates	Property used to Get or sets an array of DateTime objects specifying which dates should be bold.
CalendarDimensions	Property used to Get or set the number of columns.
FirstDayOfWeek	Property to get or set the first day of the week.
MaxDate	Property to get or set the maximum possible date.
MaxSelectionCount	Property holds the maximum number of days that can be selected.
MinDate	Property to get or set the minimum possible date.
ScrollChange	Property holds the scroll rate.
TodaysDate	Property set or get the todays date.
TodayDateSet	Property specifies whether the Date Time property has been set.

**Methods:**

<b>Method</b>	<b>Description</b>
AddBodedDate	Method used to add a day that will be displayed in Bold.
AddMonthlyBodedDate	Method used to add a day that will be displayed in bold monthly.
RemoveBodedDate	Method used to remove a date from the calendars internal list o monthly boded dates.
SetCalendarDimension	Method used to set the number of columns and rows.
SetDate	Method used to set the selected Date.
SetSelectionRange	Method used to set the selected dates to the given range of dates.

**Events:**

<b>Events</b>	<b>Description</b>
Datechanged	Triggered when the date in the calendar control is changed.
DateSelected	

*Public Class Form11*

*Private Sub MonthCalendar1\_DateChanged(ByVal sender As System.Object, ByVal e As System.Windows.Forms.DateRangeEventArgs) Handles MonthCalendar1.DateChanged*

*'selected date will display in Label*

*Label2.Text = 'Selected date is ' + MonthCalendar1.SelectionRange.Start*

*End Sub*

*End Class*

## **TIMER CONTROL**

Timer Control is used when user wants to perform some task or action continuously at regular interval of time.

### ***Property***

Property Name	Description
<b>Name</b>	It is used to specify name of the Timer Control.
<b>Enabled</b>	It is used to determine whether Timer Control will be enabled or not. It has boolean value true or false. Default value is false.
<b>Interval</b>	It is used to specify interval in millisecond. Tick event of Timer Control generates after the time which is specified in Interval Property.

### ***Methods***

Method Name	Description
<b>Start</b>	This method is used to start the Timer Control.
<b>Stop</b>	This method is used to stop the Timer Control.

### ***Events***

Event Name	Description
<b>Tick</b>	Tick event of the Timer Control fires continuously after the time which is specified in the Interval property of Timer Control.



*lblHour.Text=Now.Hour*

*lblMinute.Text=Now.Minute*

*lblSecond.Text = Now.Second*

Now double click on the Start Button and write following code in the **Click event** of Button.

*Timer1.Start()*

Now double click on the Stop Button and write following code in the **Click event** of Button.

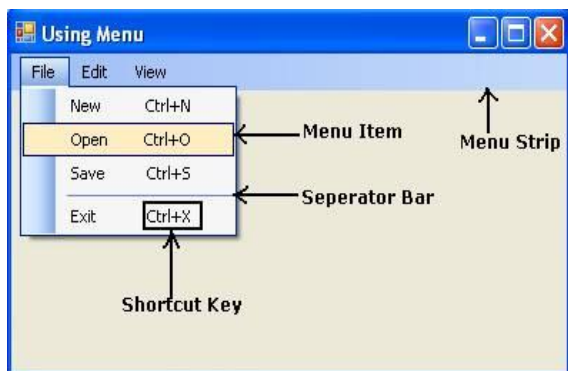
*Timer1.Stop()*

## MENU

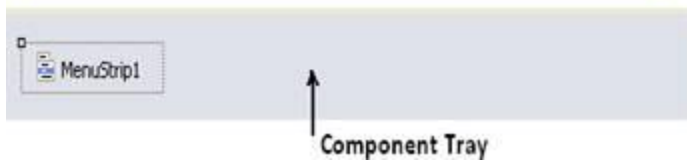
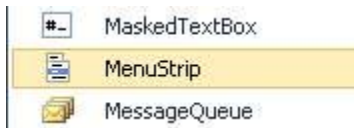
Menu is one of the most common elements of Graphical User Interface.

Menu is a one type of control that represents a group of choices to the user and allows user to select any of them according to their requirement.

It can be attached only with form either SDI or MDI.










You can repeat same procedure in order to create Menu Item and Sub Menu as per your requirement.

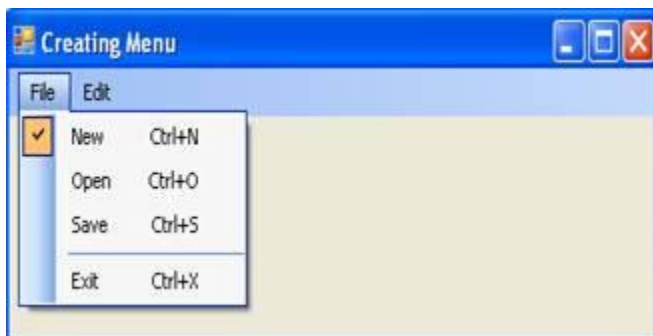
Property	Purpose
BackColor	It is used to get or set back color of the MenuStrip.
Enabled	It is used to specify whether MenuStrip is enabled or not at run time. It has Boolean value. Default value is true.
Font	It is used to set Font Face, Font Style, Font Size and Effects of the text associated with Menu Items of MenuStrip Control.
Items	It represents collection of Menu Items contained in Menu Strip control.
Layout Style	It is used to get or set Layout Style of Menu Strip Control. It has following 5 options: (1) Stack with Overflow (2) Horizontal Stack With Overflow

	<p>(3) Vertical Stack With Overflow</p> <p>(4) Flow</p> <p>(5) Table</p>
Text Direction	<p>It is used to get or set value which determines direction of text in each menu Item. It has following 3 options:</p> <p>(1) Horizontal: </p> <p>(2) Vertical90: </p> <p>(3) Vertical270: </p>
Visible	<p>It is used to specify whether MenuStrip is visible or not at run time. It has Boolean value. Default value is true.</p>



Shortcut Key allows user to perform action associated with particular Menu Item using keyboard.

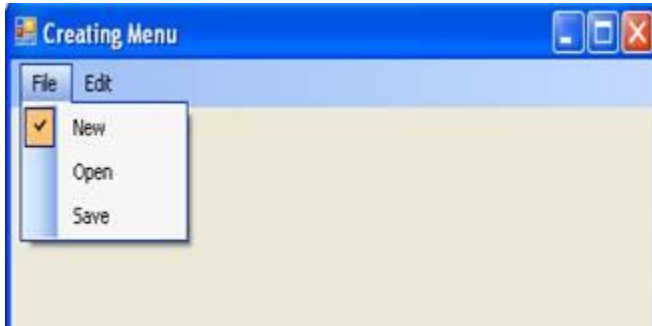
Shortcut Key allows user to perform action with a single keystroke. Shortcut Key is a combination of (Alt, Shift, Ctrl) key and other key as shown in the figure below:



Property	Purpose
ShortcutKeys	It is used to assign Shortcut Key to Menu Item.
showShortcutKeys	is used to specify whether Shortcut Key is displayed beside Menu Item or not. It has

	Boolean value. Default value is False.
ShortcutKey	It is used to get or set string that is display instead of Shortcut Key.
DisplayString	

User can add checkmark to the left side of Menu Item to indicate that the Menu Item is selected or not.



Property	Purpose
Checked	is used to specify whether checkmark will be displayed to the left side of Menu item or Not. It has Boolean value. Default is False.
CheckOnClick	It is used to specify whether Menu Item will toggle (change) its state or not when it is clicked. It has Boolean value. Default value is false.
CheckState	It is used to get or set state of menu item. It can have one of the following 3 values: (1) Checked (2) Unchecked (3) Indeterminate Default value is Unchecked.

## BUILD IN DIALOG BOXES:

### i) OPEN FILE DIALOG:

Property	Purpose
FileName	represents full path of the file that is selected by user in the OpenFileDialog Control.
Filter	It is used to specify which type of files will be display in the OpenFileDialog Control. If user wants to display only executable files than user can set Filter Property to <b>Executable File   *.exe</b> If user wants to display executable files and Image Files than user can set Filter

	Property to <b>Executable File   *.exe   Image Files  *.jpeg</b>
MultiSelect	It is used to specify whether multiple files can be selected from OpenFileDialog control or not. It has Boolean value. Default value is false.
ShowReadOnly	It is used to specify whether “Open as read only” checkbox will be displayed in OpenFileDialog control or not. It has Boolean value. Default value is false.
ReadOnlyChecked	It is used to specify whether “Open as read only” checkbox is selected or not when OpenFileDialog control is open. It has Boolean value. Default value is false.
Title	is used to specify the text to be display in the title bar of the OpenFileDialog Control.

### **Methods**

Method	Purpose
ShowDialog	It is used to Show or run OpenFileDialog Control.
Reset	It is used to reset all the properties of OpenFileDialog to its default values.
OpenFile	It is used to open the file which is selected by user in read only mode.

### **Events:**

Event	Purpose
FileOk	is the default event of OpenFileDialog Control. It fires each time user clicks on Open button of OpenFileDialog Control. It is used to perform specific task when user click on Open button.

### **ii) SAVE FILE DIALOG:**

SaveFileDialog Control allows user to:

- (1) Specify Location where to save the file.
- (2) Specify Name of File by which it is saved.

### **Properties:**

Property	Purpose
DefaultExt	It is used to specify default extension for file name. Default extension is appended at the end of file name if user selects file with no extension.
FileName	It represents full path of the file that is selected by user in the SaveFileDialog Control.

Filter	is used to specify which type of files will be display in the SaveFileDialog Control. If user wants to display only executable files than user can set Filter Property to <b>Executable File   *.exe</b> If user wants to display executable files and Image Files than user can set Filter Property to <b>Executable File   *.exe   Image Files  *.jpeg</b>
Title	It is used to specify the text to be display in the title bar of the SaveFileDialog Control.

**Methods:**

Method	Purpose
ShowDialog	It is used to Show or run SaveFileDialog Control.
Reset	It is used to reset all the properties of SaveFileDialog to its default values.
OpenFile	It is used to open the file which is selected by user in read/write mode.

**Events:**

Event	Purpose
FileOk	It is the default event of SaveFileDialog Control. It fires each time user clicks on Save button of SaveFileDialog Control. It is used to perform specific task when user click on Save button.

**iii) COLOR DIALOG CONTROL:**

Color Dialog Control allows user to select color from the list of available colors.

User can also define custom colors using Color Dialog control.

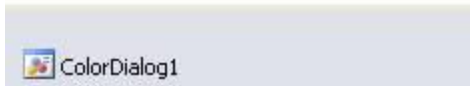
**Properties:**

Property	Purpose
Color	It is used to get or set the color selected by the user in Color Dialog Control. It is also used set specific color in the Color Dialog Control.
FullOpen	It is used to specify whether Custom Color Section of the Color Dialog Control is by default displayed or not. It has Boolean value. Its default value is false.
AllowFullOpen	It is used to enable or disable <b>Define Custom Color</b> button In Color Dialog Control. It

	has Boolean value. Its default value is true.
AnyColor	is used to specify whether Color Dialog will display all the available colors in the set of basic colors or not.
SolidColorOnly	It is used to specify whether Color Dialog will restrict user to select only solid colors or not. It has Boolean value. Default value is False.

**Methods:**

Method	Purpose
ShowDialog	It is used to Show or run Color Dialog Control.
Reset	It is used to reset all the properties of ColorDialog to its default values.



```
If ColorDialog1.ShowDialog = Windows.Forms.DialogResult.OK Then
txtName.ForeColor = ColorDialog1.Color
End If
```

**iv) FONT DIALOG CONTROL**

Font Dialog Control allows user to:

- (1) Set Font Face
- (2) Set Font Style
- (3) Set Font Size
- (4) Set Font Color
- (5) Set Font Effects

**Properties:**

Property	Purpose
MaxSize	It is used to specify Maximum Font Size that user can select from Font Dialog Control.
MinSize	It is used to specify Minimum Font Size that user can select from Font Dialog Control.
Color	It is used to get color selected by user in the Font Dialog Control. User can also set the color in Font Dialog control using this property.
Font	It is used to get the font selected in the Font Dialog Control. User can also set the font style in the Font Dialog Control.
ShowApply	It is used to specify whether Apply button will be shown in the Font Dialog Control or not. It has Boolean value. Default value is false.
ShowColor	It is used to specify whether color selection combo box will be shown in the Font Dialog Control or not. It has Boolean value. Default value is false.
ShowEffects	It is used to specify whether font effect options such as Underline, Strikeout and color selection will be shown in the Font Dialog Control or not. It has Boolean value. Default value is true.

***Methods:***

Method	Purpose
ShowDialog	It is used to Show or run Font Dialog Control.
Reset	It is used to reset all the options of FontDialog to its default values.

***Events:***

Event	Purpose
Apply	is the default event of Font Dialog Control. It fires each time user clicks on Apply button of Font Dialog Control. It is used to apply font settings on selected text without closing Font Dialog Control.

## UNIT 4

### IMAGELIST:

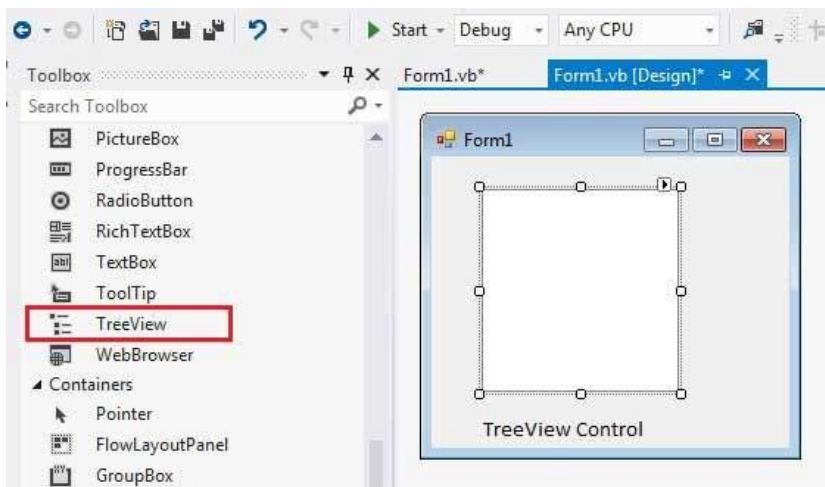
Properties	Description
ColorDepth	Property gets the color depth for this image list.
Handle	Property gets the handle for this image list.
Images	Property gets an ImageCollection object for this image list.
ImageSize	Property used to get or set the image size for the images in the list.
TransparentColor	Property used to get or set the transparent color for this list.

### Methods:

Method	Description
Draw	Method used to draw the given image.

### TREE VIEW

The TreeView control is used to display hierarchical representations of items similar to the ways the files and folders are displayed in the left pane of the Windows Explorer. Each node may contain one or more child nodes.



### Properties

Property	Description
BackColor	Gets or sets the background color for the control.
BackgroundImage	Gets or set the background image for the TreeView control.



<b>BackgroundImageLayout</b>	Gets or sets the layout of the background image for the TreeView control.
<b>BorderStyle</b>	Gets or sets the border style of the tree view control.
<b>Font</b>	Gets or sets the font of the text displayed by the control.
<b>FontHeight</b>	Gets or sets the height of the font of the control.
<b>ForeColor</b>	The current foreground color for this control, which is the color the control uses to draw its text.

### *Methods*

Method Name & Description
<b>CollapseAll:</b> Collapses all the nodes including all child nodes in the tree view control.
<b>ExpandAll:</b> Expands all the nodes.
<b>GetNodeAt:</b> Gets the node at the specified location.
<b>GetNodeCount:</b> Gets the number of tree nodes.
<b>Sort :</b> Sorts all the items in the tree view control.
<b>ToString :</b> Returns a string containing the name of the control.

### *Events*

Event	Description
<b>AfterCheck</b>	Occurs after the tree node check box is checked.
<b>AfterCollapse</b>	Occurs after the tree node is collapsed.
<b>AfterExpand</b>	Occurs after the tree node is expanded.
<b>AfterSelect</b>	Occurs after the tree node is selected.

#### **i) The TreeNode Class**

The *TreeNode* class represents a **node** of a *TreeView*. Each node in a *TreeView* control is an object of the *TreeNode* class.

#### ***Properties:***

Property	Description
<b>BackColor</b>	Gets or sets the background color of the tree node.
<b>Checked</b>	Gets or sets a value indicating whether the tree node is in a checked state.

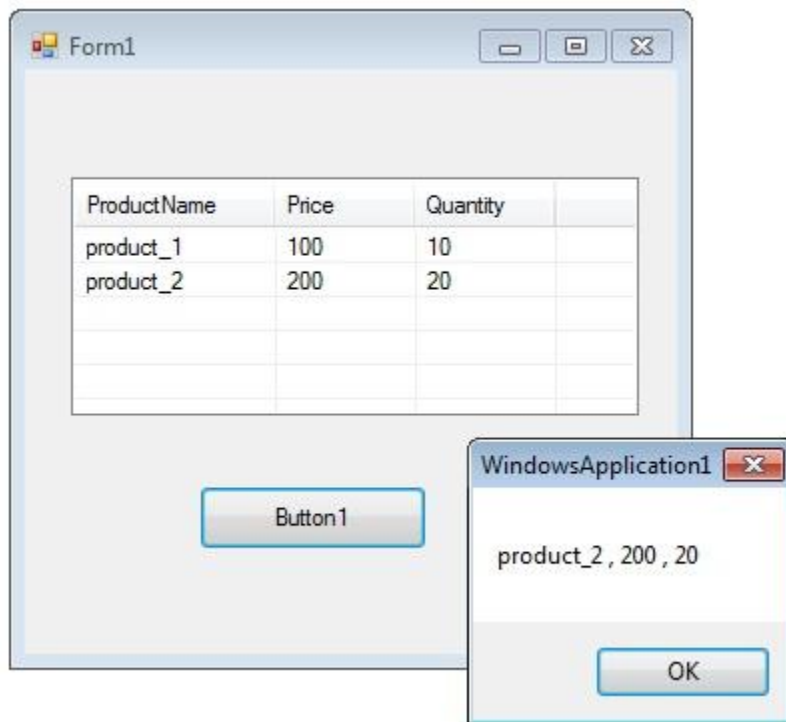
<b>ContextMenu</b>	Gets the shortcut menu that is associated with this tree node.
<b>FirstNode</b>	Gets the first child tree node in the tree node collection.
<b>FullPath</b>	Gets the path from the root tree node to the current tree node.
<b>Index</b>	Gets the position of the tree node in the tree node collection.
<b>IsEditing</b>	Gets a value indicating whether the tree node is in an editable state.

### *Methods*

Method Name & Description
<b>Collapse</b> :Collapses the tree node.
<b>Expand</b> :Expands the tree node.
<b>ExpandAll</b> :Expands all the child tree nodes.
<b>GetNodeCount</b> :Returns the number of child tree nodes.
<b>Remove</b> :Removes the current tree node from the tree view control.

## LISTVIEW CONTROL

List views displays a collection of items that can be displayed using one of five different views, such as LargeIcon, Details , SmallIcon, List and Tile.



**i) Add Columns in VB.Net ListView :** You can add columns in Listview by using Columns.Add() method.

```
listView 1.Columns.Add("Produc
```

**ii) Add Item in VB.Net Listview :** You can add items in listbox using ListViewItem which represents an item in a ListView control.

```
Dim arr As String() =  
Dim itm As ListViewItem  
'add items to ListView
```

**iii) Get selected item from VB.Net ListView:**

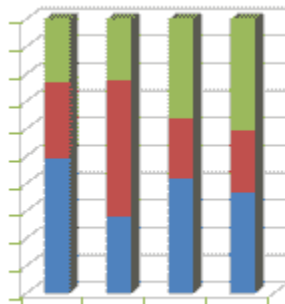
```
productName = listView 1.Selec
```

**iv) Sorting VB.Net Listview Items:** If the Sorted property of Listview is set to true, then the ListView items are sorted. The following code sorts the ListView items:

```
ListView 1.Sorted = True
```

**v) Add Checkbox in Listview:** You can add checkbox in VB.Net Listview columns.

```
myListView .CheckBoxes = True  
myListView .Columns.Add(text,
```



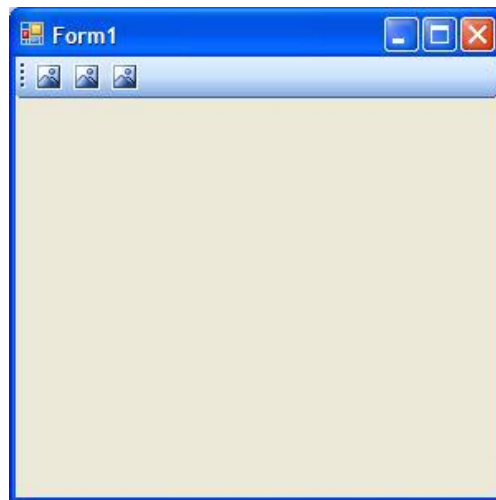
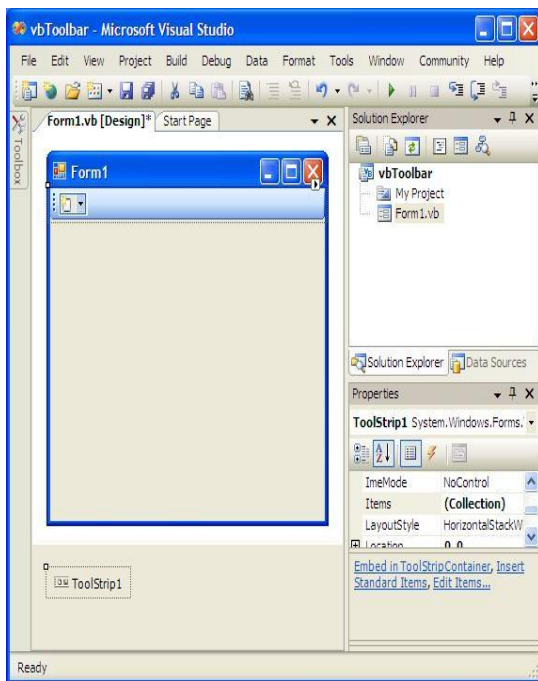
```
ListView 1.View = View.Details
ListView 1.GridLines = True
ListView 1.FullRowSelect = True
```

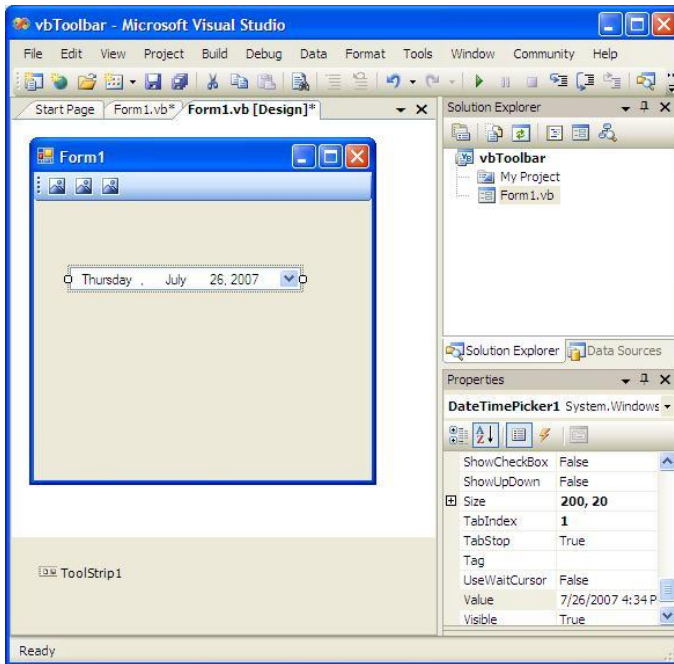
## CREATING A TOOLBAR

Toolbars are referred to in Visual Basic as *ToolStrips*.

The first step in creating a toolbar is to add a *ToolStrip* control to the form.

Begin by starting Visual Studio and creating a new Windows Application project named *vbToolbar*





```
Private Sub ToolStripButton1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
```

```
    Handles ToolStripButton1.Click
```

```
    MyDateTime.Show()
```

```
End Sub
```

```
Private Sub ToolStripButton2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
```

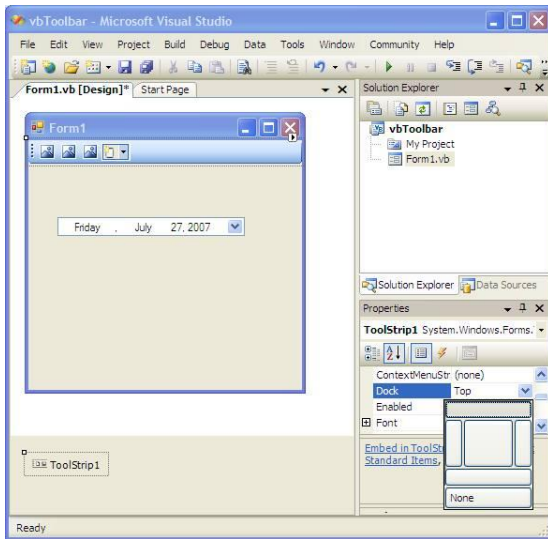
```
    Handles ToolStripButton2.Click
```

```
    MyDateTime.Hide()
```

```
End Sub
```

### i) Changing the Toolbar Position

By default, the Visual Basic ToolStrip object will be positioned across the top edge of the form. Whilst this is the most common location for a toolbar, it can be positioned along the top, bottom, left or right edges of a form, or even in the center of the form using the *Dock* property. To modify this property, select the *ToolStrip* object in the form and look for the *Dock* property in the Properties panel. Click on the down arrow in the value field to display the location map:



**STATUS BAR:** The Windows Forms StatusBar control is used on forms as an area, usually displayed at the bottom of a window, in which an application can display various kinds of status information. StatusBar controls can have status-bar panels on them that display icons to indicate state, or a series of icons in an animation that indicate a process is working.

You can display a single message on the status bar by setting the ShowPanels property to false (the default) and setting the Text property of the status bar to the text you want to appear in the status bar. You can divide the status bar into panels to display more than one type of information by setting the ShowPanels property to true and using the Add method of StatusBar.StatusBarPanelCollection.

*Imports System.Drawing  
Imports System.Windows.Forms*

*Public Class Exercise  
Inherits System.Windows.Forms.Form*

*Dim statusbar As ToolStrip*

*Public Sub New()  
statusbar = new ToolStrip*

*Controls.Add(statusbar)  
End Sub*

*Public Shared Function Main() As Integer  
Application.Run(New Exercise)*

*Return 0  
End Function*

*End Class*

**PROGRESS BAR :** It used to graphically display the progress of particular task. Thus using ProgressBar control you can display how much task has been completed and how much task is remaining.



Property Name	Description
<b>Minimum</b>	It Get or Set Lower Bound of the range within which ProgressBar Control works.
<b>Maximum</b>	It Get or Set Upper Bound of the range within which ProgressBar Control works.
<b>Value</b>	It Get or Set current value of the ProgressBar within range specified using Minimum and Maximum property.
<b>Step</b>	It Get or Set Step value by which the current value of ProgressBar control is Increment.
<b>Style</b>	It is used to set the Style of progressBar Control. It can have one of the following value: Blocks,Continuous,Marquee
<b>Visible</b>	It is used to set whether ProgressBar control is visible on the form or not. It has boolean value true or false. Default value is true.
<b>Enabled</b>	It is used to set whether ProgressBar control is enabled or not. It has boolean value true or false. Default value is true.
<b>MarqueeAnimationSpeed</b>	It Get or Set speed of marquee animation when Style property of ProgressBar Control is set to marquee. The speed is in milisecond. Default value is 100 ms.

**Methods:**

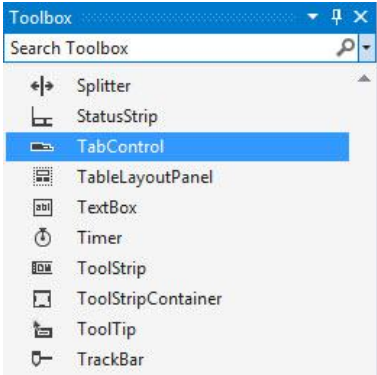
Method Name	Description
<b>Increment</b>	It is used to increment the current value of ProgressBar Control by specific value. <b>Syntax:</b> ProgressBar1.Increment(value)
<b>PerformStep</b>	It is used to increment the current value of ProgressBar Control by the value specified in the Step property of ProgressBar. <b>Syntax:</b>

```
progressBar1.PerformStep()
```



```
Dim i As Integer  
For i = ProgressBar1.Minimum To ProgressBar1.Maximum - 1  
    ProgressBar1.PerformStep()  
Next
```

**TAB CONTROL**



These controls include **FlowLayoutPanel**, **TableLayoutPanel**, **GroupBox**, **Panel**, **TabControl**, and **SplitContainer**.

The **TabControl** displays data **grouped by pages**, while the **tabs** enable the user to quickly **jump from page to page**.

**CREATING CLASSES:**

```
Syntax  
[ <attrlist> ] [ Public / Private / Protected / Friend / Protected [ Shadows ] [ MustInherit / NotInheritable ] Class name [ Implements interfacename ]
```



### ***[ statements ] End Class***

Here are the various parts of this statement:

attrlist—Optional. This is the list of attributes for this class. Separate multiple attributes by commas.

Public—Optional. Classes declared Public have public access; there are no restrictions on the use of public classes.

Private—Optional. Classes declared Private have private access, which is accessible only within its declaration context.

Protected—Optional. Classes declared Protected have protected access, which means they are accessible only from within their own class or from a derived class.

Friend—Optional. Classes declared Friend have friend access, which means they are accessible only within the program that contains their declaration.

Protected Friend—Optional. Classes declared Protected Friend have both protected and friend accessibility.

Shadows—Optional. Indicates that this class shadows a programming element in a base class.

MustInherit—Optional. Indicates that the class contains methods that must be implemented by a deriving class.

NotInheritable—Optional. Indicates that the class is a class from which no further inheritance is allowed.

name—Required. Name of the class.

interfacename—Optional. The name of the interface implemented by this class.

statements—Optional. The statements that make up the variables, properties, events, and methods of the class.

Each attribute in the attrlist part has the following syntax:

***<attrname [( { attrargs | attrinit } )]> Attrlist***

***Here are the parts of the attrlist part:***

attrname—Required. Name of the attribute.

attrargs—Optional. List of arguments for this attribute. Separate multiple arguments by commas.

attrinit—Optional. List of field or property initializers. Separate multiple initializers by commas.

*Public Class DataClass*

*Private value As Integer*

*Public Sub New(ByVal newValue As Integer)*

*value = newValue*

*End Sub*

*Public Function GetData() As Integer*

*Return value*

*End Function*

*End Class*

## **CREATING OBJECTS**

You can create objects of a class using the Dim statement; this statement is used at module, class, structure, procedure, or block level:

```
[ <attrlist> ] [ { Public / Protected / Friend / Protected Friend / Private / Static } ] [ Shared ] [ Shadows ] [ ReadOnly ] Dim [ WithEvents ] name [ (boundlist) ] [ As [ New ] type ] [ = initexp
```

Here are the parts of this statement:

## **SAME AS CLASS**

**CREATING MODULES** : used modules to divide your code up into smaller units, because modules were designed primarily to hold code.

*Module Module1*

*Sub Main()*

*System.Console.WriteLine("Hello from Visual Basic")*

*System.Console.WriteLine("Press Enter to continue...")*

*System.Console.ReadLine()*

*End Sub*

*End Module*

**CREATING CONSTRUCTOR:** A class constructor is a special member Sub of a class that is executed whenever we create new objects of that class. A constructor has the name New and it does not have any return type.

*Class Line*

*Private length As Double ' Length of a line*

*Public Sub New() 'constructor*

*Console.WriteLine("Object is being created")*

*End Sub*

*Public Sub setLength(ByVal len As Double)*

*length = len*

*End Sub*

*Public Function getLength() As Double*

*Return length*

*End Function*

*Shared Sub Main()*

*Dim line As Line = New Line()*

*'set line length*

*line.setLength(6.0)*

*Console.WriteLine("Length of line : {0}", line.getLength())*

*Console.ReadKey()*

*End Sub*

*End Class*

A default constructor does not have any parameter, but if you need, a constructor can have parameters. Such constructors are called **parameterized constructors**.

*Class Line*

*Private length As Double ' Length of a line*

*Public Sub New(ByVal len As Double) 'parameterised constructor*

*Console.WriteLine("Object is being created, length = {0}", len)*

*length = len*

*End Sub*

*Public Sub setLength(ByVal len As Double)*

*length = len*

*End Sub*

*Public Function getLength() As Double*

*Return length*

*End Function*

*Shared Sub Main()*

*Dim line As Line = New Line(10.0)*

*Console.WriteLine("Length of line set by constructor : {0}", line.getLength())*

*'set line length*

*line.setLength(6.0)*

*Console.WriteLine("Length of line set by setLength : {0}", line.getLength())*

*Console.ReadKey()*

*End Sub*

*End Class*

**CREATING METHODS:** Methods are functions/procedures defined inside the body of a class. They are used to perform operations with the attributes of our objects. Methods are essential in *encapsulation* concept of the OOP paradigm.

*Option Strict On*

*Module Example*

*Class Circle*

*Public Radius As Integer*

*Public Sub SetRadius(ByVal Radius As Integer)*

*Me.Radius = Radius*

*End Sub*

*Public Function Area() As Double*

*Return Me.Radius \* Me.Radius \* Math.PI*

*End Function*

*End Class*

*Sub Main()*

*Dim c As New Circle*

*c.SetRadius(5)*

*Console.WriteLine(c.Area())*

*End Sub*

*End Module*

**USING INHERITANCE:** It is one of the primary concepts of object-oriented programming (OOP) and it is useful to inherit the properties from one class (base) to another (child) class. The inheritance will enable us to create a new class by inheriting the properties from other classes to reuse, extend and modify the behavior of other class members based on our requirements. In visual basic inheritance, the class whose members are inherited is called a **base (parent)** class and the class that inherits the members of **base (parent)** class is called a **derived (child)** class.

Syntax

*<access\_modifier> Class <base\_class\_name>*

*// Base class Implementation*

*End Class*

*<access\_modifier> Class <derived\_class\_name>*

*Inherits base\_class\_name*

*// Derived class implementation*

*End Class*

### **USING PUBLIC INHERITANCE:**

*Public class form1 Inherits System.Windows.Forms.Form*

*Dim spot As Dog*

*Private sub Button1\_Click(ByVal sender As System.Object,ByVal e As System.EventArgs) Handles*

*Button1.Click*

*Spot=New Dog(Me)*

*Spot.Breathing()*

*End Sub*

*End Class*

*Public class Animal*

*Protected MainFrame As Form1*

*Public Sub New(ByVal form1 As Form1)*

*MainForm=form1*

*End Sub*

*Public Sub Breathing()*

*MainForm.TextBox1.Text="Breathing...."*

*End Sub*

*End Class*

*Public Class Dog*

*Inherit Animal*

*Public Sub New(ByVal form1 As Form1)*

*MyBase.New(form1)*

*End Sub*

*Public Sub Barking*

*MainForm.TextBox1.Text="Barking....."*

*End Sub*

*End Class*

## **USING PROTECTED INHERITANCE**

*Public class form1 Inherits System.Windows.Forms.Form*

*Dim spot As Dog*

*Private sub Button1\_Click(ByVal sender As System.Object,ByVal e As System.EventArgs) Handles  
Button1.Click*

*Spot=New Dog(Me)*

*Spot.Breathing()*

*End Sub*

*End Class*

*Public class Animal*

*Protected MainFrame As Form1*

*Public Sub New(ByVal form1 As Form1)*

*MainForm=form1*

*End Sub*

*Public Sub Breathing()*

*MainForm.TextBox1.Text="Breathing...."*

*End Sub*

*End Class*

*Public Class Dog*

*Inherit Animal*

*Public Sub New(ByVal form1 As Form1)*

*MyBase.New(form1)*

*End Sub*

*Public Sub Barking()*

*MainForm.TextBox1.Text="Barking....."*

*End Sub*

*End Class*

## **USING PRIVATE INHERITANCE**

*Public class form1 Inherits System.Windows.Forms.Form*

*Dim spot As Dog*

*Dim jaws As Fish*

*Private sub Button1\_Click(ByVal sender As System.Object,ByVal e As System.EventArgs) Handles  
Button1.Click*

*Spot=New Dog(Me)*

*Spot.Breathing()*

*End Sub*

*Private sub Button1\_Click(ByVal sender As System.Object,ByVal e As System.EventArgs) Handles  
Button1.Click*

*Jaws=New Fish(Me)*

*Jaws.Breathing()*

*End Sub*

*End Class*

*Public class Animal*

*Private MainForm As Form1*

*Public Sub New(ByVal form1 As Form1)*

*MainForm=form1*

*End Sub*

*Public Overridable Sub Breathing()*

*MainForm.TextBox1.Text="Breathing...."*

*End Sub*

*End Class*

*Public Class Dog*

*Inherit Animal*

*Public Sub New(ByVal form1 As Form1)*

*MyBase.New(form1)*

*End Sub*

*Public Sub Barking()*

*MainForm.TextBox1.Text="Barking....." 'Will Not Work*

*End Sub*

*End Class*

*Public Class Fish Inherit Animal*

*Public Sub New(ByVal form1 As Form1)*

*MyBase.New(form1)*

*End Sub*

*Public Overrides Sub Breathing()*

*MainForm.TextBox1.Text="Bubbling...."*

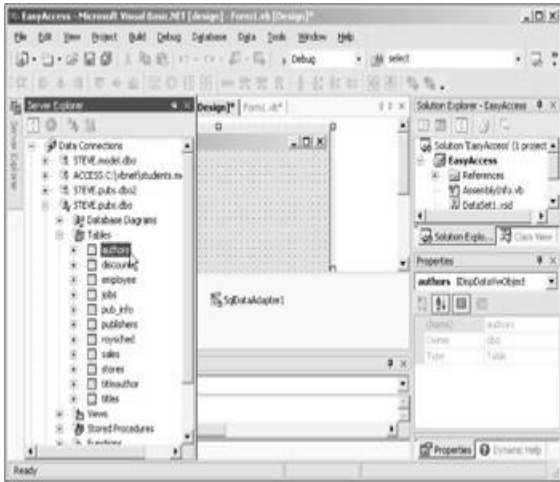
*End Sub*

*End Class*

## UNIT 5

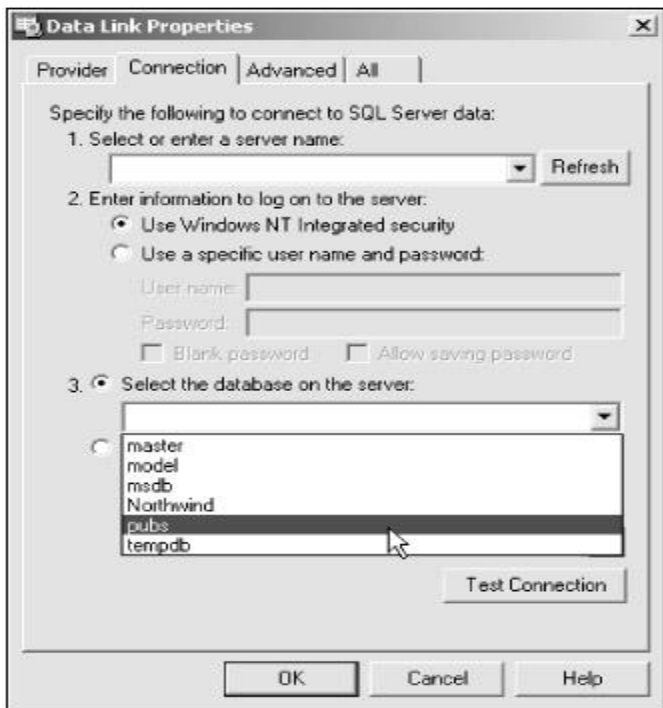
### Accessing data with Server Explorer

To work with a database, you need a connection to that database. In Visual Basic, the Server Explorer lets you work with connections to various data sources. To display the Server Explorer if it's not already visible, use the View|Server Explorer menu item, or press Ctrl+Alt+S.

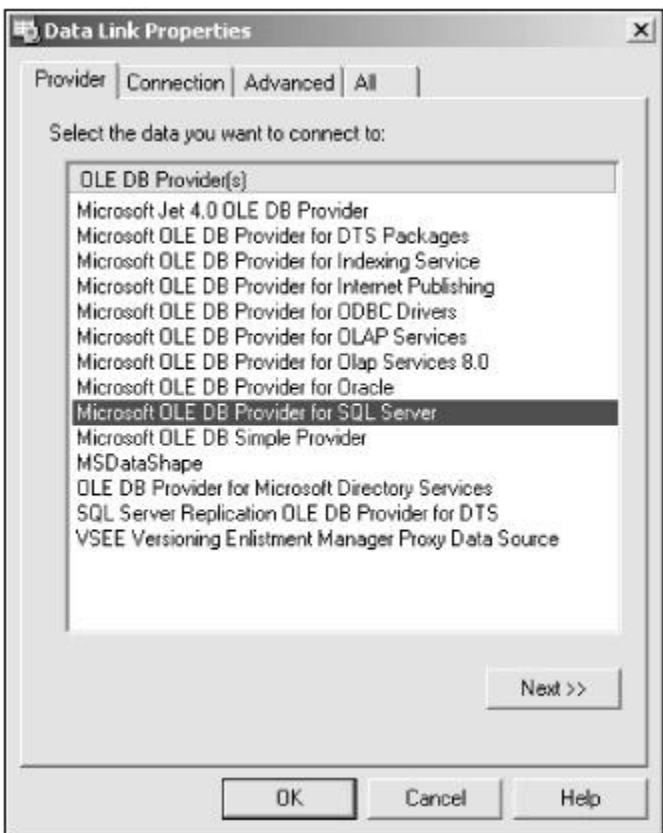


To create that connection, right-click the Data Connections icon in the Server Explorer and select the Add Connection item, or use the Tools|Connect to Database menu item. Doing so opens the Data Link Properties dialog





What if you're not using SQL Server, but, say, Oracle to connect to a database? In that case, you click the Provider tab in the Data Link Properties dialog, as you see in Figure 20.6, and select the type of provider you're working with—Oracle, MS Jet, and so on (the default is SQL Server). Then you go back to the Connection tab and choose the specific database file you want to work with



When the connection is set, click the OK button to close the Data Link Properties dialog. Doing so adds a new connection to the pubs database to the Server Explorer, as you see in Figure 20.3. You can open that

connection (assuming, in this case, that SQL Server is running) and take a look what tables are in the database.

## ACCESSING DATA WITH DATA ADAPTER AND DATA SET

**DataSet** to hold all of your information from the database

Each imaginary row of the DataSet represents a Row of information in your Access database. And each imaginary column represents a Column of information in your Access database (called a Field in database terminology).

The Connection Object and the DataSet can't see each other. They need a go-between so that they can communicate. This go-between is called a Data Adapter.

The Data Adapter contacts your Connection Object, and then executes a query that you set up. The results of that query are then stored in the DataSet.

```
Dim ds As DataSet
Dim da As OleDb.OleDbDataAdapter
da = New OleDb.OleDbDataAdapter( sql, con )
```

### The Data Adapter

The Data Adapter is a property of the OLEDB object, hence the full stop between the two:

### OleDb.OleDbDataAdapter

We're passing this object to the variable called **da**. This variable will then hold a reference to the Data Adapter.

```
da = New OleDb.OleDbDataAdapter(sql, con )
```

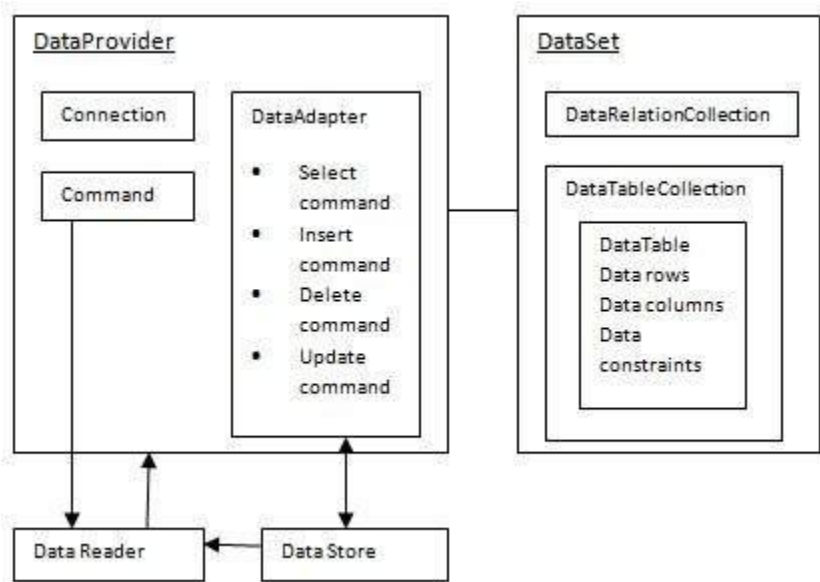
We need something else, though. The **sql** in between the round brackets is the name of a variable. We haven't yet set this up. We'll have a look at SQL in a moment. But bear in mind what the Data Adaptor is doing: *Acting as a go-between for the Connection Object and the Data Set*

## WORKING WITH ADO.NET

Applications communicate with a database, firstly, to retrieve the data stored there and present it in a user-friendly way, and secondly, to update the database by inserting, modifying and deleting data.

Microsoft ActiveX Data Objects.Net (ADO.Net) is a model, a part of the .Net framework that is used by the .Net applications for retrieving, accessing and updating data.

ADO.Net object model is nothing but the structured process flow through various components. The object model can be pictorially described as –



- **Datasets** store data in a disconnected cache and the application retrieves data from it.
- **Data readers** provide data to the application in a read-only and forward-only mode.

### Data Provider:

A data provider is used for connecting to a database, executing commands and retrieving data, storing it in a dataset, reading the retrieved data and updating the database.

The data provider in ADO.Net consists of the following four objects –

No.	Objects & Description
	<b>Connection:</b> This component is used to set up a connection with a data source.
	<b>Command:</b> A command is a SQL statement or a stored procedure used to retrieve, insert, delete modify data in a data source.

	<b>DataReader:</b> Data reader is used to retrieve data from a data source in a read-only and forward-only mode.
	<b>DataAdapter:</b> This is integral to the working of ADO.Net since data is transferred to and from database through a data adapter. It retrieves data from a database into a dataset and updates database. When changes are made to the dataset, the changes in the database are actually done by the data adapter.

There are following different types of data providers included in ADO.Net

- The .Net Framework data provider for SQL Server - provides access to Microsoft SQL Server.
- The .Net Framework data provider for OLE DB - provides access to data sources exposed by using OLE DB.
- The .Net Framework data provider for ODBC - provides access to data sources exposed by ODBC.
- The .Net Framework data provider for Oracle - provides access to Oracle data source.
- The EntityClient provider - enables accessing data through Entity Data Model (EDM) applications.

## **DataSet**

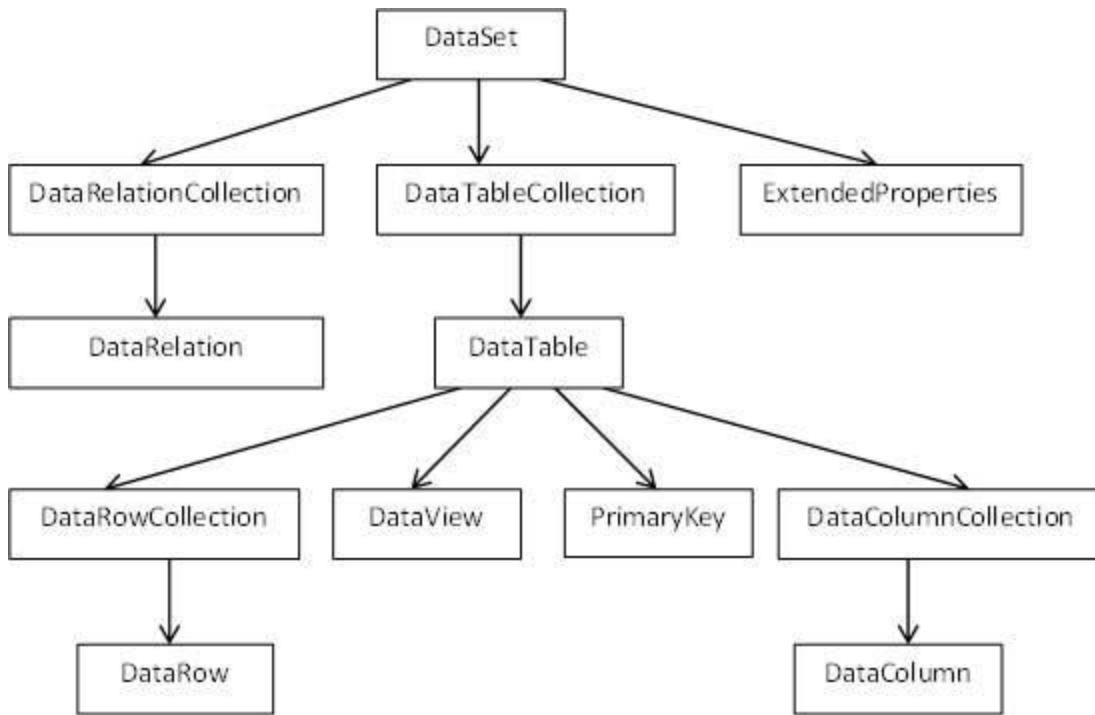
**DataSet** is an in-memory representation of data.

It is a disconnected, cached set of records that are retrieved from a database.

When a connection is established with the database, the data adapter creates a dataset and stores data in it.

After the data is retrieved and stored in a dataset, the connection with the database is closed.

This is called the 'disconnected architecture'. The dataset works as a virtual database containing tables, rows, and columns.



The DataSet class is present in the **System.Data** namespace. The following table describes all the components of DataSet –

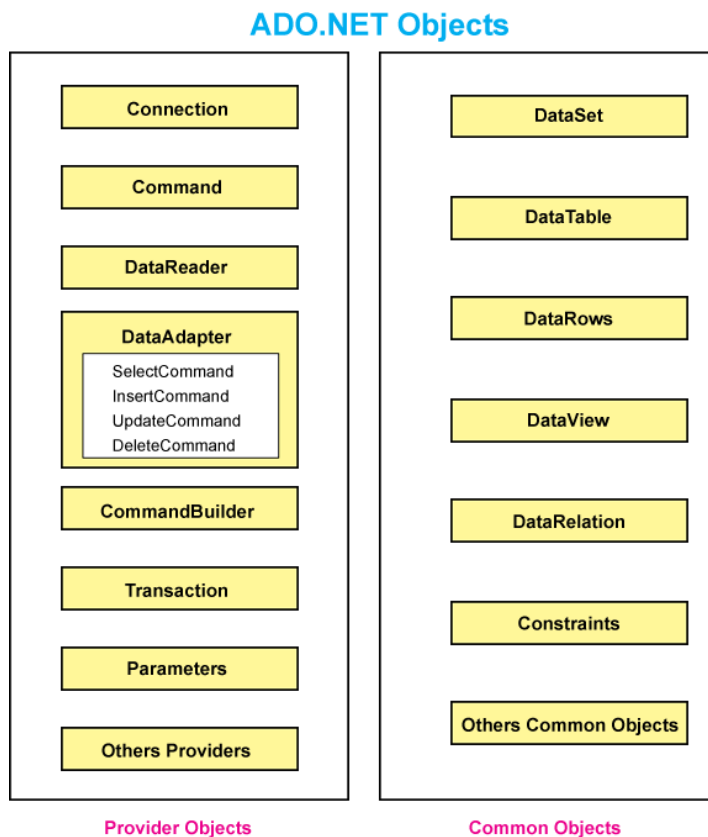
Components & Description
<b>DataTableCollection:</b> It contains all the tables retrieved from the data source.
<b>DataRelationCollection:</b> It contains relationships and the links between tables in a data set.
<b>ExtendedProperties:</b> It contains additional information, like the SQL statement for retrieving data, time of retrieval, etc.
<b>DataTable:</b> It represents a table in the DataTableCollection of a dataset. It consists of the DataRow and DataColumn objects. The DataTable objects are case-sensitive.
<b>DataRelation:</b> It represents a relationship in the DataRelationshipCollection of the dataset. It is used to relate two DataTable objects to each other through the DataColumn objects.
<b>DataRowCollection:</b> It contains all the rows in a DataTable.
<b>DataView:</b> It represents a fixed customized view of a DataTable for sorting, filtering, searching, editing and navigation.
<b>PrimaryKey:</b> It represents the column that uniquely identifies a row in a DataTable.
<b>DataRow:</b> It represents a row in the DataTable. The DataRow object and its properties and methods are used to retrieve, evaluate, insert, delete, and update values in the DataTable. The NewRow method is used to create a new row and the Add method adds a row to the table.
<b>DataColumnCollection:</b> It represents all the columns in a DataTable.
<b>DataColumn:</b> It consists of the number of columns that comprise a DataTable.

## OVERVIEW OF ADO.NET OBJECTS:

ADO.NET is designed to help developers work efficiently with multi-tier databases, across intranet or Internet scenarios.

The ADO.NET object model consists of two key components as follows:

- Connected model (.NET Data Provider - a set of components including the Connection, Command, DataReader, and DataAdapter objects)
- Disconnected model (DataSet).



**Connection:** The Connection object is the first component of ADO.NET. The connection object opens a connection to your data source. All of the configurable aspects of a database connection are represented in the Connection object, which includes ConnectionString and ConnectionTimeout. Connection object helps in accessing and manipulating a database. Database transactions are also dependent upon the Connection object.

In ADO.NET the type of the Connection is depended on what Database system you are working with. The following are the commonly used connections in the ADO.NET

- SqlConnection

- OleDbConnection
- OdbcConnection

**Command:** The Command object is used to perform an action on the data source. Command object can execute stored procedures and T-SQL commands. You can execute SQL queries to return data in a DataSet or a DataReader object. Command object performs the standard Select, Insert, Delete, and Update T-SQL operations.

**DataReader:** The DataReader is built as a way to retrieve and examine the rows returned in response to your query as quickly as possible. No DataSet is created; in fact, no more than one row of information from the data source is in memory at a time.

**DataAdapter:** The DataAdapter takes the results of a database query from a Command object and pushes them into a DataSet using the DataAdapter.Fill() method.

DataAdapter object works in a connected model. DataAdapter performs the five following steps:

1. Create/open the connection
2. Fetch the data as per command specified
3. Generate XML file of data
4. Fill data into DataSet.
5. Close connection.

**Command Builder:** It is used to save changes made in an in-memory cache of data on the backend. The work of Command Builder is to generate Command as per changes in DataRows. Command Builder generates command on basis of row state. There is a five-row state:

1. Unchanged
2. Added
3. Deleted
4. Modified
5. Detached

Command Builder works on add, delete, and modified row state only.

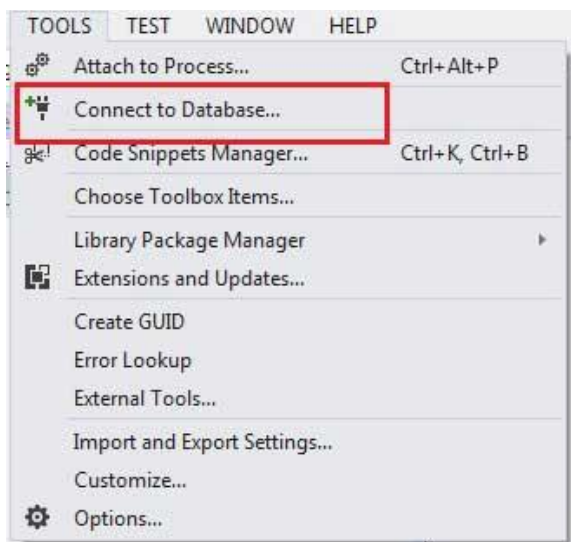
**Transaction:** The Transaction object is used to execute the backend transaction. Transactions are used to ensure that multiple changes to database rows occur as a single unit of work.

**Parameters:** Parameter object is used to solve the SQL Injection attack problem while dealing with the user input parameters. Parameter object allows passing parameters into a Command object the Parameter class allows you to quickly put parameters into a query without string concatenation.

## CREATING A NEW DATA CONNECTION

The .Net Framework provides two types of Connection classes –

- **SqlConnection** – designed for connecting to Microsoft SQL Server.
- **OleDbConnection** – designed for connecting to a wide range of databases, like Microsoft Access and Oracle.
- Select TOOLS → Connect to Database



Select a server name and the database name in the Add Connection dialog box



Add Connection

Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.

Data source:  
Microsoft SQL Server (SqlClient) Change...

Server name:  
KABIR-DESKTOP Refresh

Log on to the server

Use Windows Authentication  
 Use SQL Server Authentication

User name:   
Password:   
 Save my password

Connect to a database

Select or enter a database name:  
testDB

Attach a database file:  
 Browse...  
Logical name:

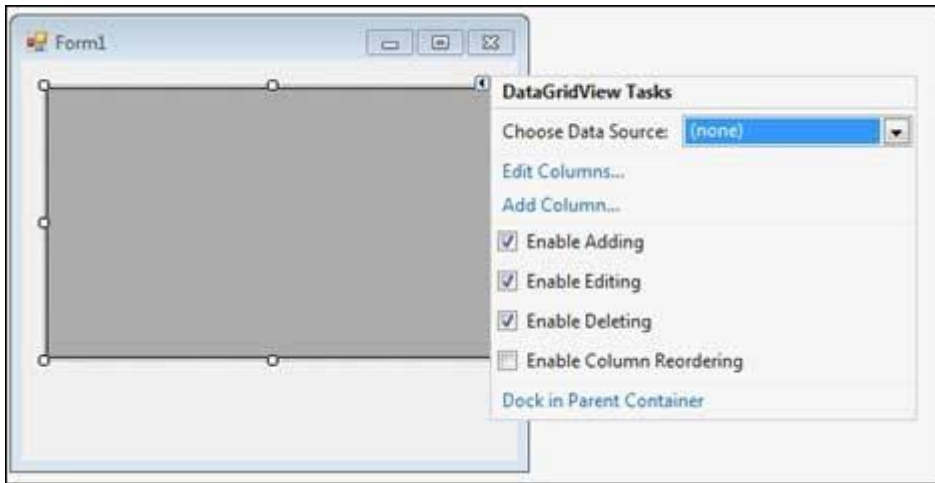
Advanced...

Test Connection OK Cancel

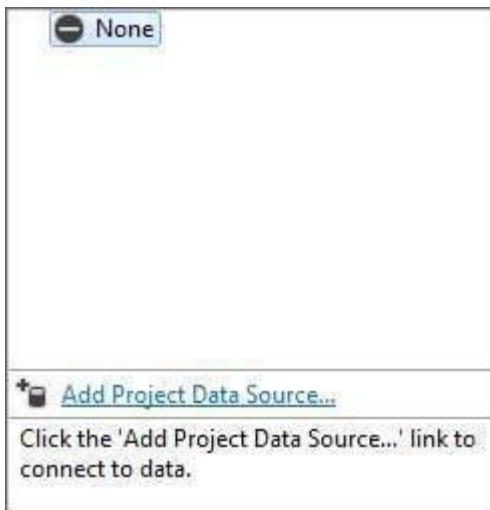
- Click on the Test Connection button to check if the connection succeeded.



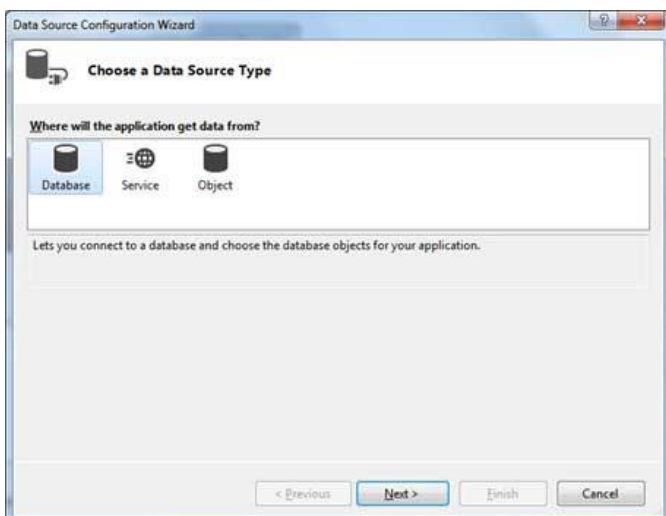
- Add a DataGridView on the form.



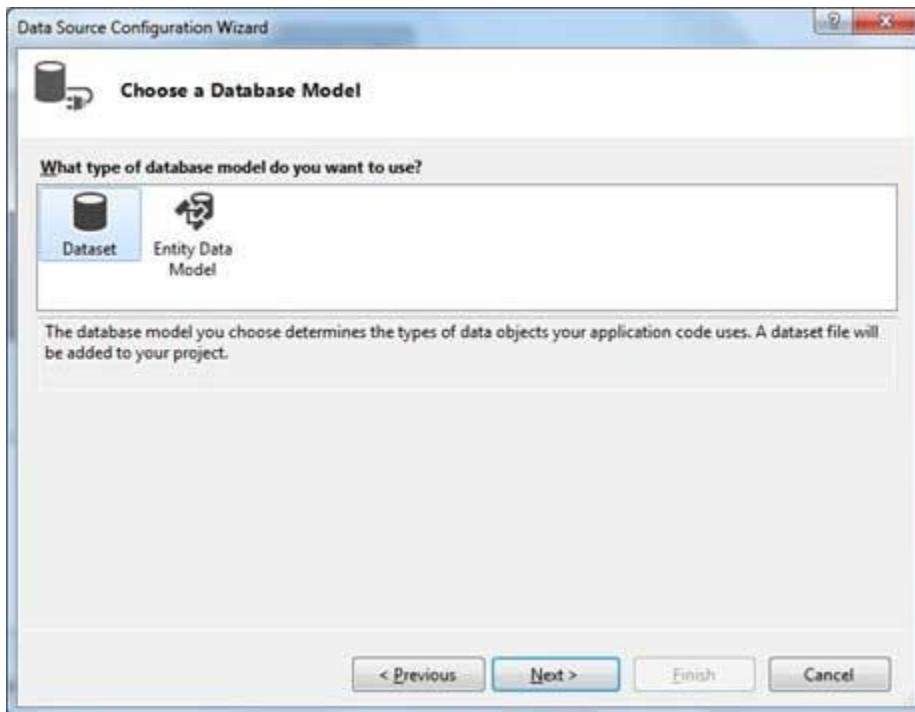
- Click on the Choose Data Source combo box.
- Click on the Add Project Data Source link.



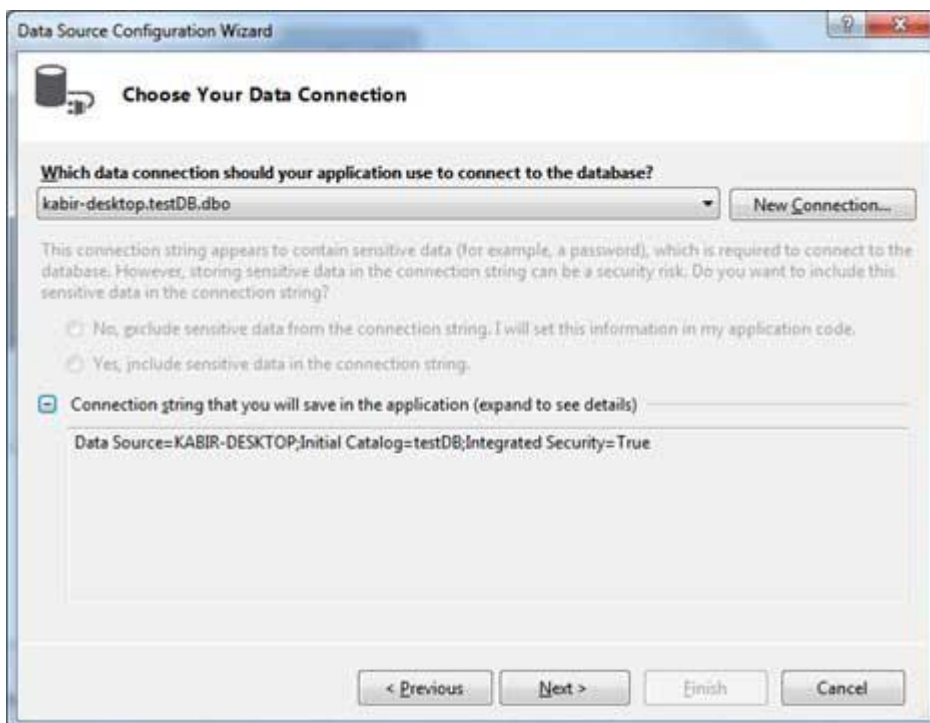
- This opens the Data Source Configuration Wizard.
- Select Database as the data source type



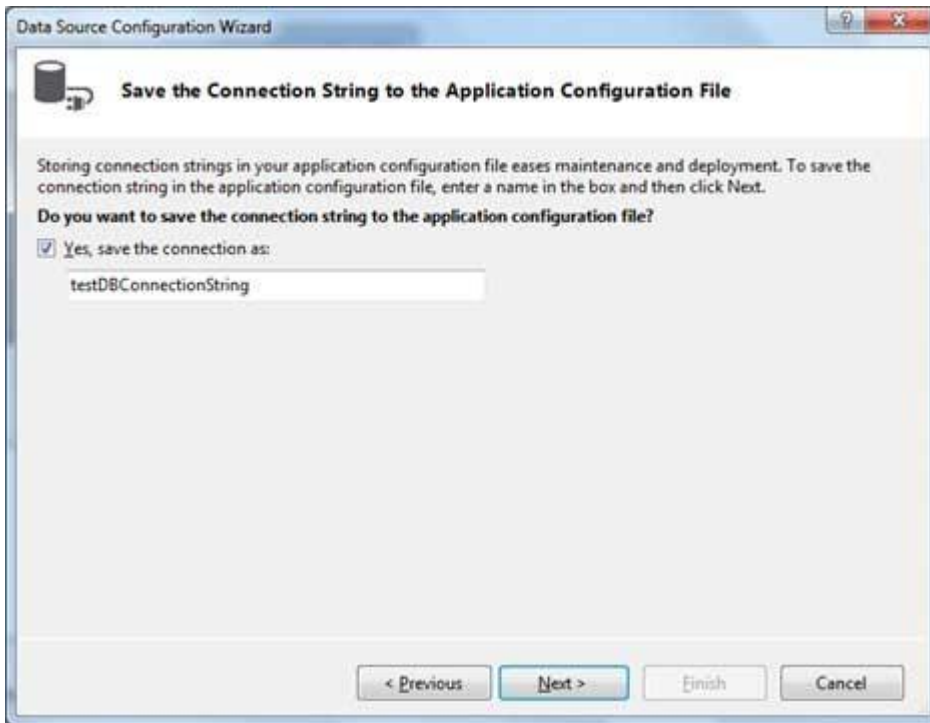
- Choose DataSet as the database model.



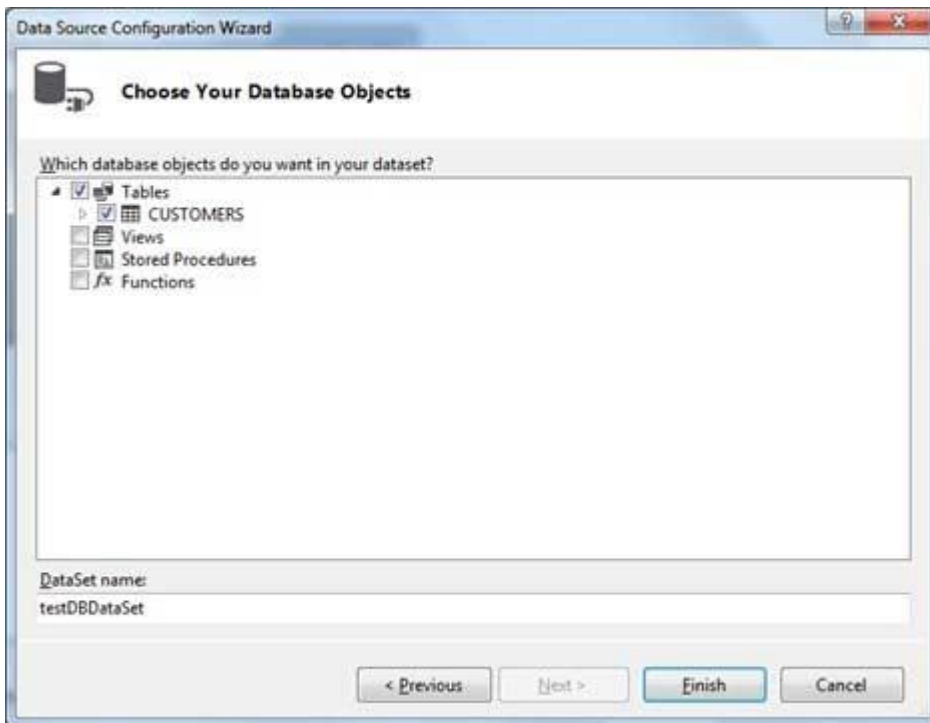
- Choose the connection already set up.



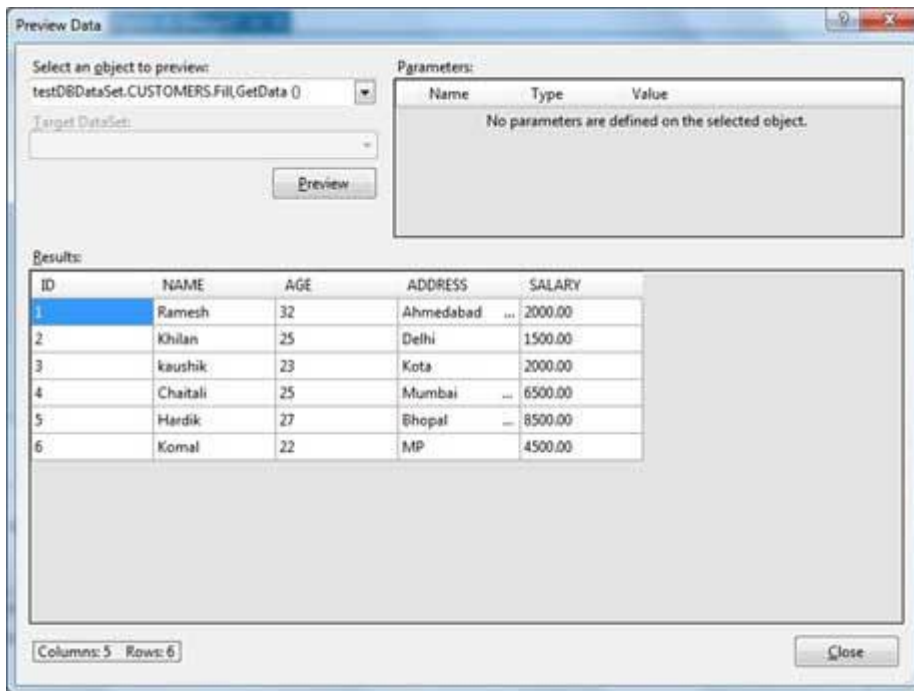
- Save the connection string.



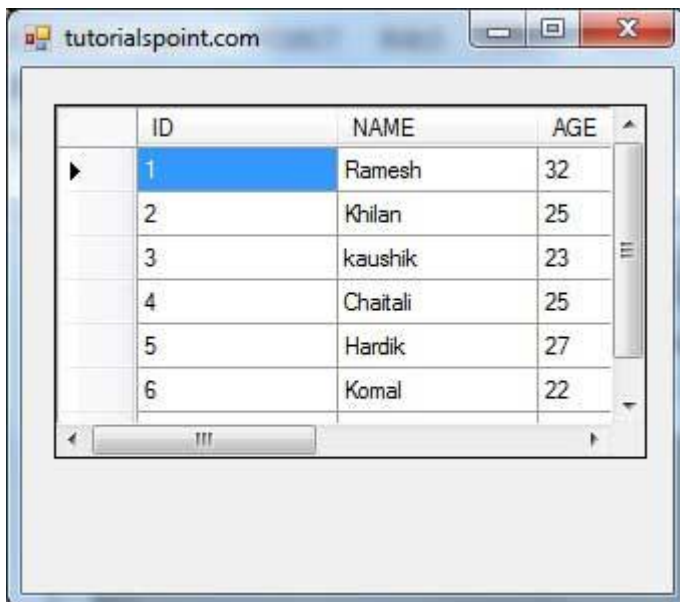
- Choose the database object, Customers table in our example, and click the Finish button.



- Select the Preview Data link to see the data in the Results grid –



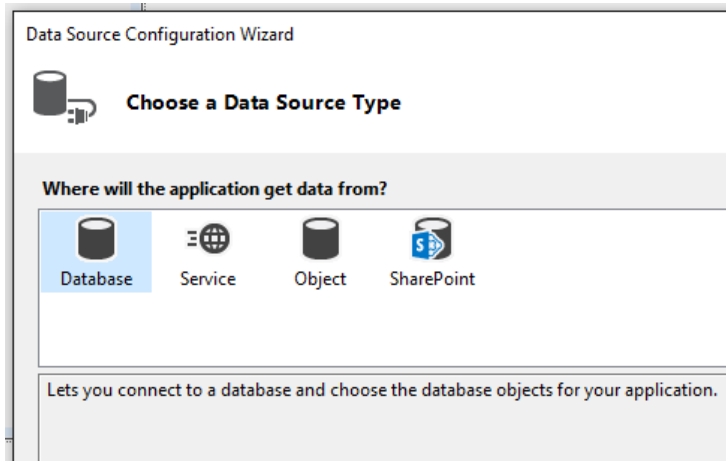
When the application is run using **Start** button available at the Microsoft Visual Studio tool bar, it will show the following window –



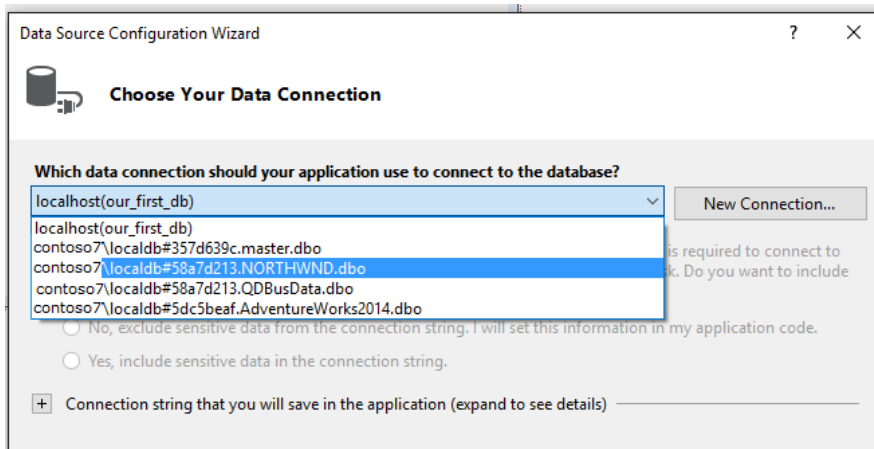
## CREATING A DATASET:

A dataset is a set of objects that store data from a database in memory and support change tracking to enable create, read, update, and delete (CRUD) operations on that data without the need to be always connected to the database.

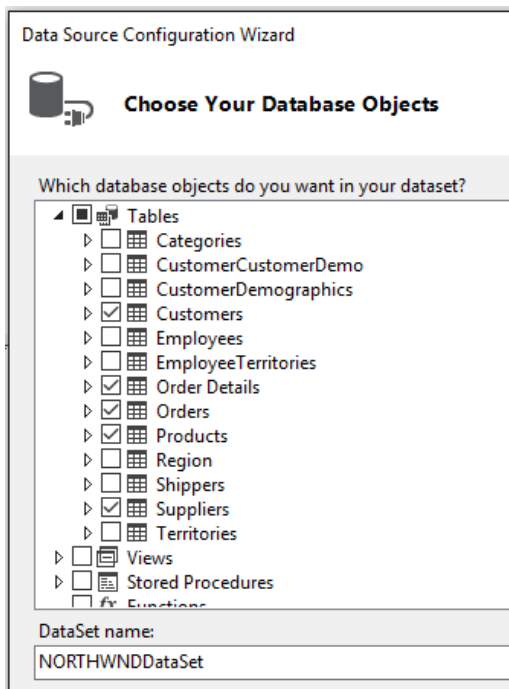
1. Open your project in Visual Studio, and then choose **Project > Add New Data Source** to start the **Data Source Configuration Wizard**.
2. Choose the type of data source to which you'll be connecting.



3. Choose the database or databases that will be the data source for your dataset.

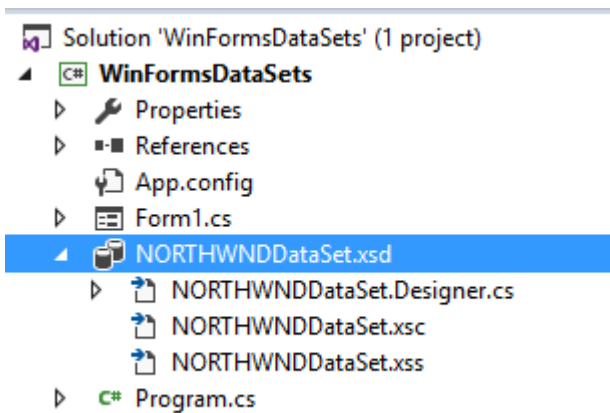


4. Choose the tables (or individual columns), stored procedures, functions, and views from the database that you want to be represented in the dataset.

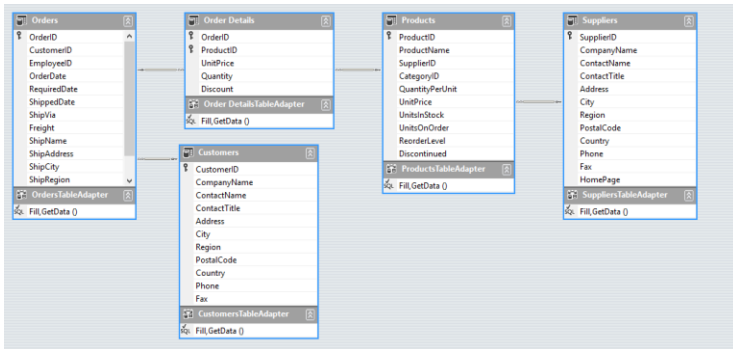


5. Click **Finish**.

The dataset appears as a node in **Solution Explorer**.



6. Click the dataset node in **Solution Explorer** to open the dataset in the **DataSet Designer**. Each table in the dataset has an associated TableAdapter object, which is represented at the bottom. The table adapter is used to populate the dataset and optionally to send commands to the database.



Relation

Name:

Specify the keys that relate tables in your dataset.

Parent Table:  Child Table:

Key Columns	Foreign Key Columns
OrderID	CustomerID

Choose what to create

Both Relation and Foreign Key Constraint  
 Foreign Key Constraint Only  
 Relation Only

Update Rule:

Delete Rule:

Accept/Reject Rule:

Nested Relation

- Click a table, table adapter, or column name in a table to see its properties in the **Properties** window. You can modify some of the values here. Just remember that you are modifying the dataset, not the source database.

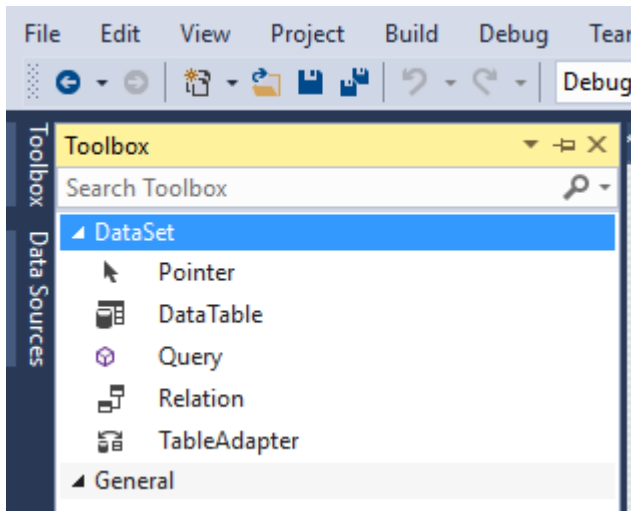
Properties

**OrderID** DataColumn

Property	Value
AllowDBNull	False
AutoIncrement	True
AutoIncrementSeed	-1
AutoIncrementStep	-1
Caption	OrderID
DataType	System.Int32
DateTimeMode	UnspecifiedLocal
DefaultValue	<DBNull>
Expression	
MaxLength	-1
NullValue	(Throw exception)
ReadOnly	True
Source	OrderID
Unique	True
Misc	
Name	OrderID



8. You can add new tables or table adapters to the dataset, or add new queries for existing table adapters, or specify new relations between tables by dragging those items from the **Toolbox** tab. This tab appears when the **DataSet Designer** is in focus.



## ADDING A TABLE TO A DATASET

To define the structure, or *schema*, of your new DataSet. It's time to add a table to it.

Inside that table, define three columns: one for the last name of each person in your address book, the second for the first name, and the third for an auto incrementing primary key.

1. Right-click the DataSet1 icon from the previous example in the tray, and choose Properties.

The Properties window appears, showing the properties of DataSet1.

2. In the Properties window, change the Name property (not the DataSetName property) of DataSet1 to dsAddresses.

The DataSet icon in the tray changes to display its new name. (Behind the scenes, VB.NET also changes the name in the source code that it writes automatically to define the contents of your form.)

3. In the Properties window, click the Tables property and then click the ellipsis (...).

The Tables Collection Editor appears.

4. In the Tables Collection Editor, click the Add button.

The table's properties appears.

5. Change the Name property (not the TableName property) to Friends.

VB.NET again changes the source code behind the scenes. You don't have to worry about these details — just relax and know that VB.NET knows how to write the code that defines your new DataSet's schema.

6. In the Properties list of the Tables Collection Editor, click Columns and then click the ellipsis.

The Columns Collection Editor dialog box appears.

7. In the Columns Collection Editor, click the Add button.

You can now define a new column and its properties. Note that the DataType property for all columns defaults to the string (text) type, although you can change it. This is the data type that you want for the LastName and FirstName columns.

8. Change the Name property (not the ColumnName property) to LastName. (The name defaults to DataColumn1.)

9. Click the Add button.

Column2 is now created.

10. Change this column's Name property to FirstName.

11. Click the Add button.

Column3 is now created.

12. Change this column's Name property to Key, and its ReadOnly property to True.

With the ReadOnly property True, nobody can write (change) any of the data in this column. That's what you want; it's supposed to be looked at (read) only.

13. Double-click the Unique property.

The property changes from False to True. Now the DataSet refuses to permit two rows to contain identical data in the Key column. In addition, as long as this property is True, you can't use the Remove button in the Columns Collection Editor dialog box to delete the column.

14. Double-click the AutoIncrement property.

The property changes from False to True. Now the DataSet automatically increments (increases) the number in this column by one for each row. Notice that when you double-clicked this property, VB.NET was wise enough to change this column's DataType property from String to Integer. After all, you want ordinary numbers (1, 2, 3, 4, and so on) in this column, not text. Text can't be incremented.

15. Click the Close button twice.

The Columns Collection Editor and the Tables Collection Editor close.

## **POPULATING A DATASET FROM A DATAADAPTER**

The ADO.NET DataSet is a memory-resident representation of data that provides a consistent relational programming model independent of the data source.

The DataSet represents a complete set of data that includes tables, constraints, and relationships among the tables.

The SelectCommand property of the DataAdapter is a Command object that retrieves data from the data source.

The InsertCommand, UpdateCommand, and DeleteCommand properties of the DataAdapter are Command objects that manage updates to the data in the data source according to modifications made to the data in the DataSet.

These properties are covered in more detail in Updating Data Sources with DataAdapters.

The Fill method of the DataAdapter is used to populate a DataSet with the results of the SelectCommand of the DataAdapter.

Fill takes as its arguments a DataSet to be populated, and a DataTable object, or the name of the DataTable to be filled with the rows returned from the SelectCommand.

vb

```
// Assumes that connection is a valid SqlConnection object.
```

```
string queryString =
```

```
"SELECT CustomerID, CompanyName FROM dbo.Customers";
```

```
SqlDataAdapter adapter = new SqlDataAdapter(queryString, connection);
```

```
DataSet customers = new DataSet();  
adapter.Fill(customers, "Customers");
```

### **Multiple Result Sets**

If the DataAdapter encounters multiple result sets, it creates multiple tables in the DataSet.

The tables are given an incremental default name of TableN, starting with "Table" for Table0.

If a table name is passed as an argument to the Fill method, the tables are given an incremental default name of TableNameN, starting with "TableName" for TableName0.

### **POPULATING A DATASET FROM MULTIPLE DATAADAPTERS**

Any number of DataAdapter objects can be used with a DataSet. Each DataAdapter can be used to fill one or more DataTable objects and resolve updates back to the relevant data source.

Data Relation and Constraint objects can be added to the DataSet locally, which enables you to relate data from dissimilar data sources.

```
SqlDataAdapter custAdapter = new SqlDataAdapter( "SELECT * FROM dbo.Customers",  
customerConnection);  
OleDbDataAdapter ordAdapter = new OleDbDataAdapter(  
"SELECT * FROM Orders", orderConnection);  
DataSet customerOrders = new DataSet();  
custAdapter.Fill(customerOrders, "Customers");  
ordAdapter.Fill(customerOrders, "Orders");
```

```
DataRelation relation = customerOrders.Relations.Add("CustOrders",  
customerOrders.Tables["Customers"].Columns["CustomerID"],  
customerOrders.Tables["Orders"].Columns["CustomerID"]);
```

### **DISPLAY DATA IN A DATAGRID**

Refer lab exercise program.....

## SELECTING A DATA PROVIDERS

A .NET Framework data provider is used for connecting to a database, executing commands, and retrieving results.

.NET Framework data providers are lightweight, creating a minimal layer between the data source and code, increasing performance without sacrificing functionality.

The following table lists the data providers that are included in the .NET Framework.

<b>.NET Framework data provider</b>	<b>Description</b>
.NET Framework Data Provider for SQL Server	Provides data access for Microsoft SQL Server. Uses the <u>System.Data.SqlClient</u> namespace.
.NET Framework Data Provider for OLE DB	For data sources exposed by using OLE DB. Uses the <u>System.Data.OleDb</u> namespace.
.NET Framework Data Provider for ODBC	For data sources exposed by using ODBC. Uses the <u>System.Data.Odbc</u> namespace.
.NET Framework Data Provider for Oracle	For Oracle data sources. The .NET Framework Data Provider for Oracle supports Oracle client software version 8.1.7 and later, and uses the <u>System.Data.OracleClient</u> namespace.
EntityClient Provider	Provides data access for Entity Data Model (EDM) applications. Uses the <u>System.Data.EntityClient</u> namespace.
.NET Framework Data Provider for SQL Server Compact 4.0.	Provides data access for Microsoft SQL Server Compact 4.0. Uses the <u>System.Data.SqlServerCe</u> namespace.

## DATA VIEW

A DataView represents a view a DataSet object.

You can set filters on the data or sort on data in the DataSet through different DataViews and produce different views of the data

```
private void DemonstrateDataView()
{
    // Create one DataTable with one column.
```

```

DataTable table = new DataTable("table");
DataColumn colItem = new DataColumn("item",
    Type.GetType("System.String"));
table.Columns.Add(colItem);

// Add five items.
DataRow NewRow;
for(int i = 0; i <5; i++)
{
    NewRow = table.NewRow();
    NewRow["item"] = "Item " + i;
    table.Rows.Add(NewRow);
}
// Change the values in the table.
table.AcceptChanges();
table.Rows[0]["item"]="cat";
table.Rows[1]["item"] = "dog";
// Print current table values.
PrintTableOrView(table,"Current Values in Table");
firstView.RowStateFilter=DataRowState.ModifiedOriginal;
PrintTableOrView(firstView,"First DataView: ModifiedOriginal");

}

```

## **DATA BINDING**

Data binding, in the context of .NET, is the method by which controls on a user interface (UI) of a client application are configured to fetch from, or update data into, a data source, such as a database or XML document.

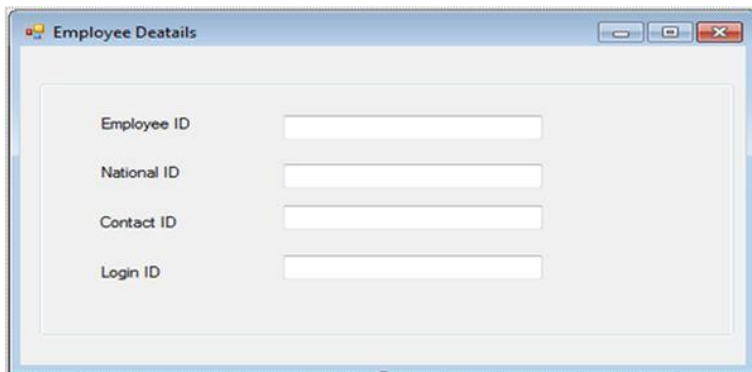
Many database management systems (DBM) could indirectly access the data source through their application programming interface (API) without any flexibility in controlling the data binding process. The development of Web applications is simplified by providing data binding capability to Web pages using .NET server side Web controls.

Simple data binding is the process of binding a control, such as a TextBox or a Label control, to a value in a dataset.

The dataset value can be bound to the control by using the properties of the control. A Textbox control is used for simple binding. While simple data binding involves binding a control to a dataset value, complex data binding is the process of binding a component to display multiple values for a column from the dataset rows. The DataGrid, Listbox, and ComboBox controls are used for complex binding.

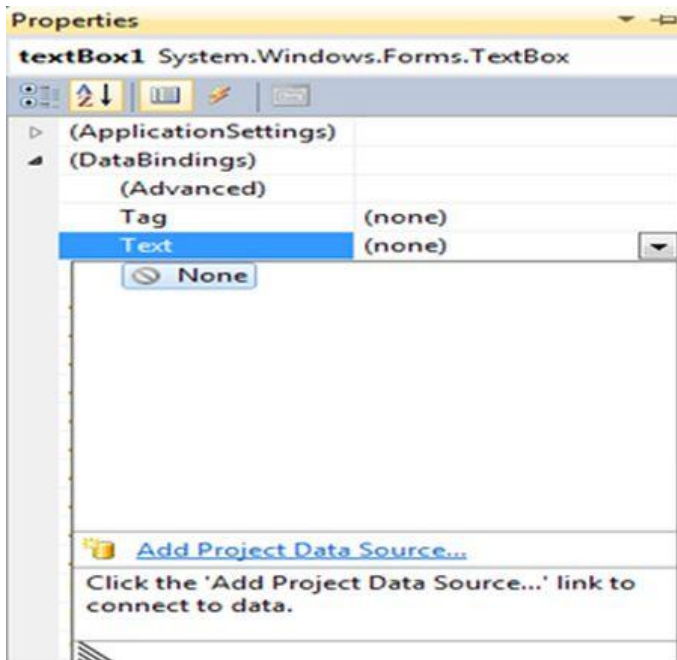
### **i) Simple data binding**

Simple data binding allows you to bind a control to a single data element. The most common use of simple data binding involves binding a single data element, such as the value of a column in a table, to a control on a form. You use this type of data binding for controls that show only one value. Uses of simple data binding include binding data to text boxes and labels.

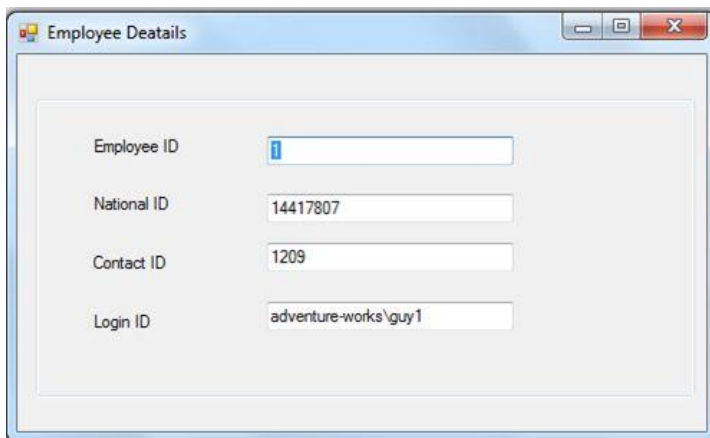


### **Steps**

1. Press F4 to open the properties window.
2. Select the first TextBox to display its properties window.
3. Expand the (DataBindings) property.
4. Select the text property to enable the drop-down list. Click the drop-down list.
5. Add a project data source in the drop-down list.



6. Create a connection with the AdventureWorks database and select the HumanResources.employee table.
7. Select the first TextBox. Expand "Other data source" ---> "Project data source" --> "AdventureWorksdataset" --> "Employee" --> "EmployeeId".
8. Select the second TextBox. Expand the DataBinding property then select "Text" ---> "Employee Binding source" then select column(National ID) from the list.
9. Similarly, bind TextBox3 and TextBox4 with the column contactid and Login ID.
10. Press F5. If everything goes well, you will see the following output:



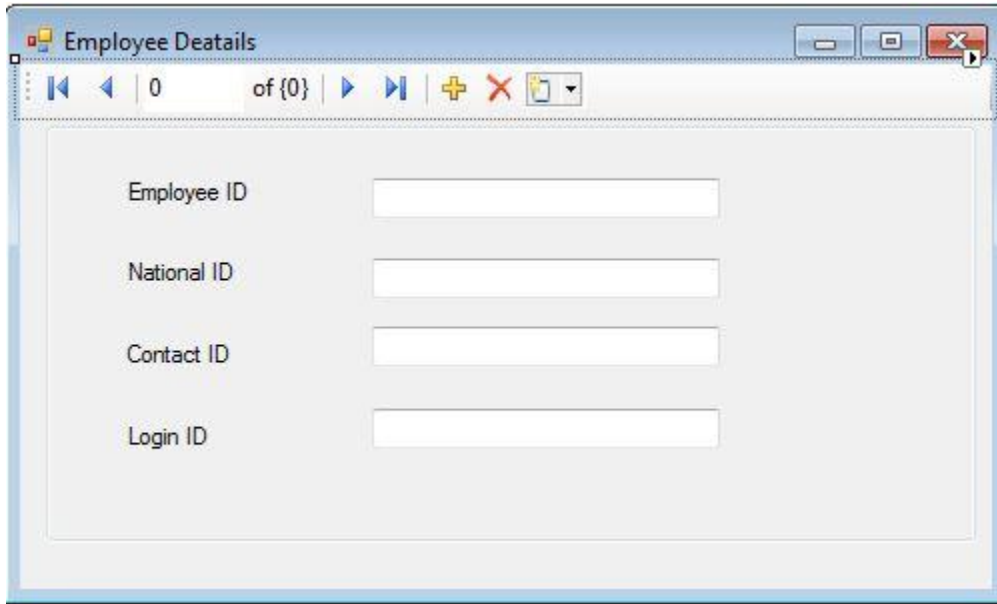
Here is one problem with the preceding. Every time you run your project, you are able to see only one record. So how to solve this problem?



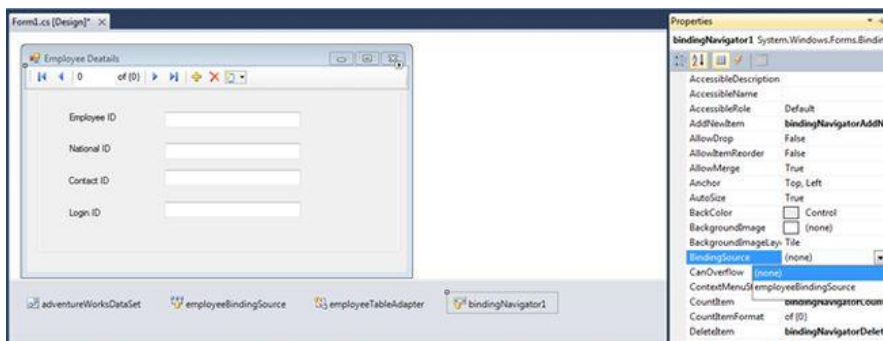
We can solve that problem using the BindingNavigator Control. For every data source that is bound to a Windows Forms control, there exists a BindingNavigator control. The BindingNavigator control handles the binding to the data by keeping the pointer to the item in the record list current.

## Implementing BindingNavigator

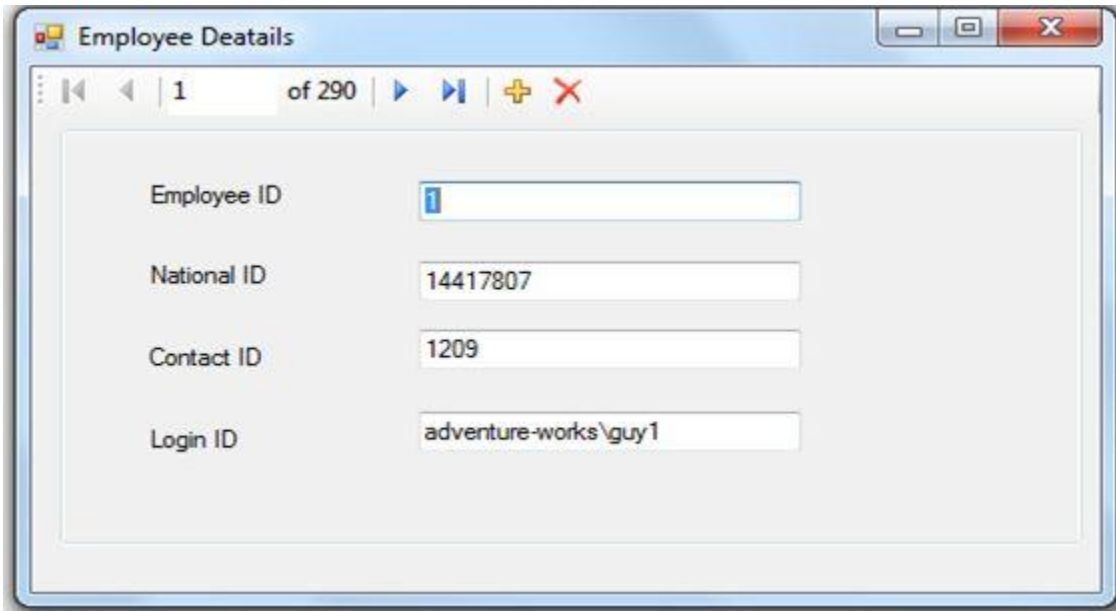
1. Drag and drop a BindingNavigator control from the Toolbox.



2. Select BindingNavigator1; this will display the properties window.
3. Select the bindingSource property from the properties Window to enable the corresponding drop-down list.
4. Select employeeBindingSource from the Drop-Down list as shown in the following figure.



5. Execute the Windows Forms form and verify the output. The Employee Details will be displayed as shown in the following figure.



The following table describes the various symbols and their function in the BindingNavigator control:

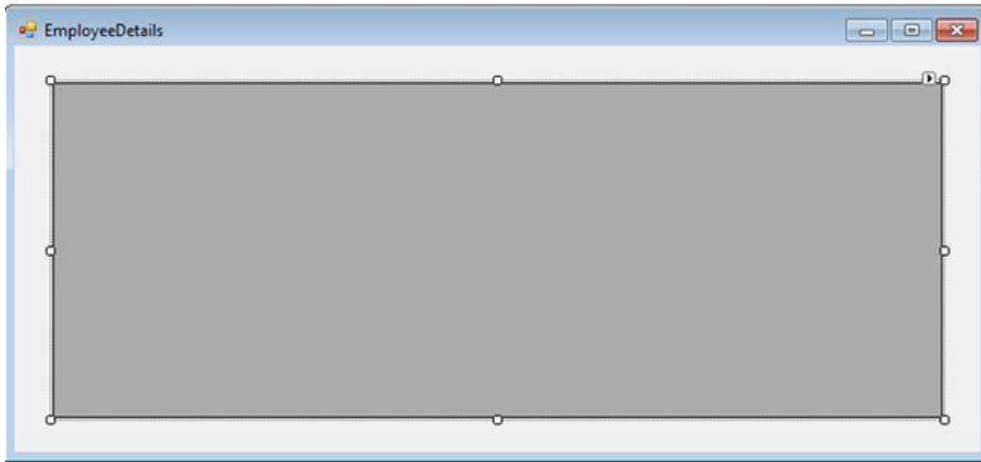
Symbol	Function
+	Insert a new row to data source
X	Delete current row from data source
⏪	Move to first record in data source
⏩	Move to last record in data source
⏴	Move to previous record in data source
⏵	Move to next record in data source
0	Return the current position of record in data source.
of {0}	Return total number of records present in data source

## ii) Complex Data Binding

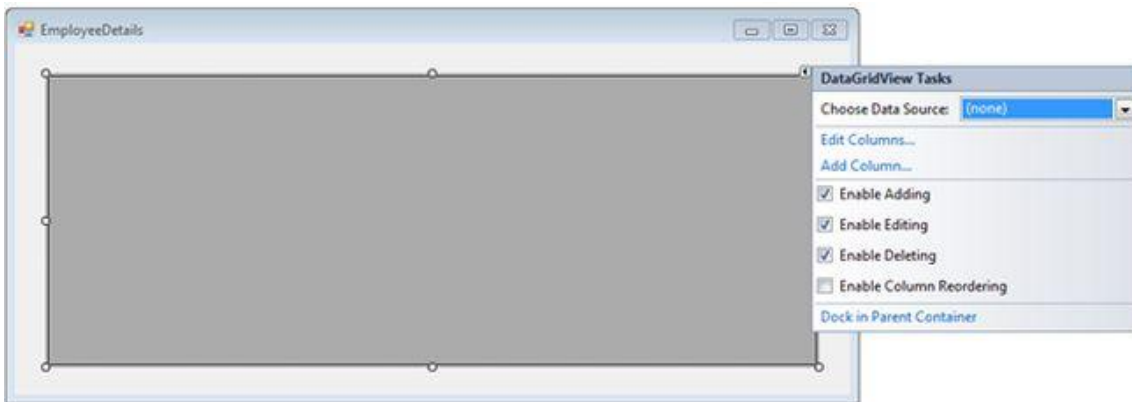
Complex data binding allows you to bind more than one data element to control. Using the column example, complex data binding involves binding more than one column or row from the underlying record source. Controls that support complex data binding include data grid controls, combo boxes, and list boxes.

Let's see complex data binding with a DataGridView:

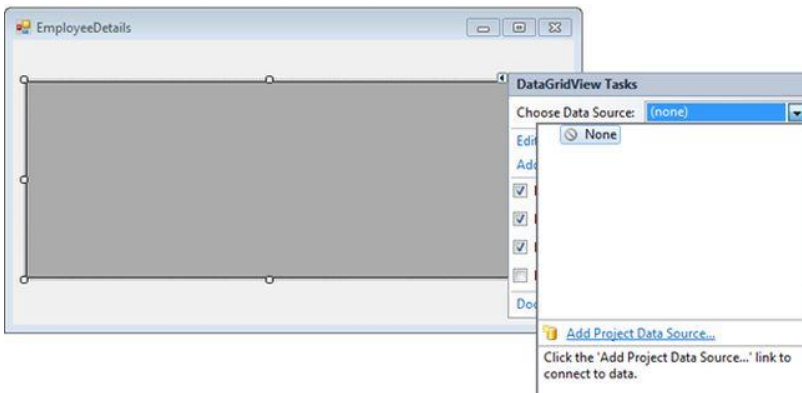
1. Drag and drop a DataGridView from the Toolbox under the Data tab.



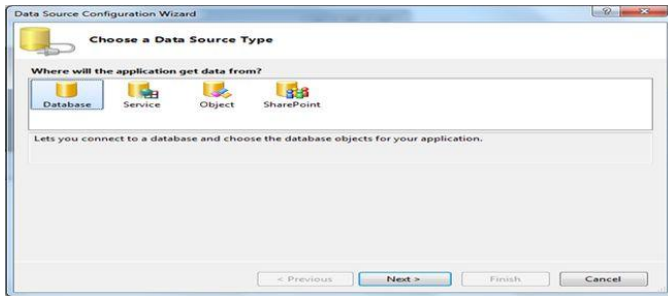
2. Click on the DataGridView task pop-up menu as shown in the following figure.



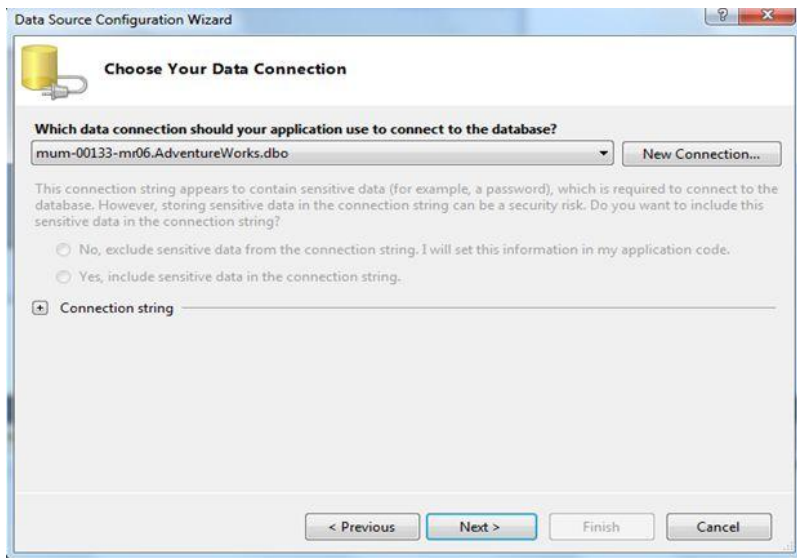
3. Select the choose Data Source drop-down list and then select the Add Project Data Source from the DataGridView task pop-up menu as shown in the following figure.



4. In the database configuration wizard select database and click on "Next" as shown in the following figure.



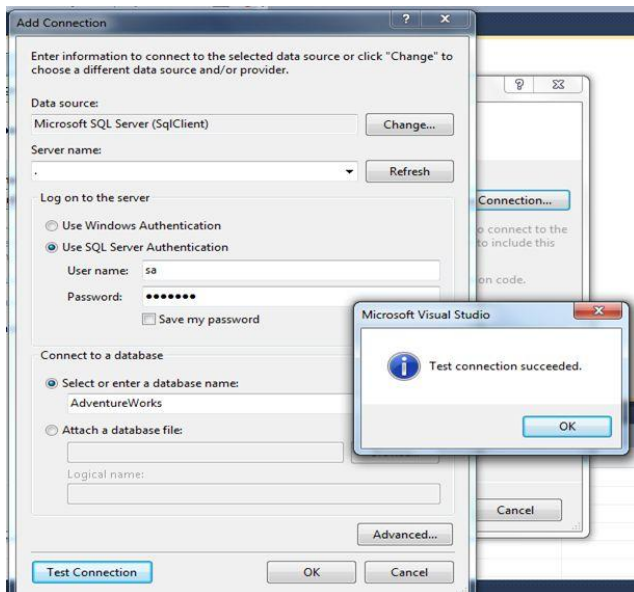
5. After clicking on the Next button you will get the following output.



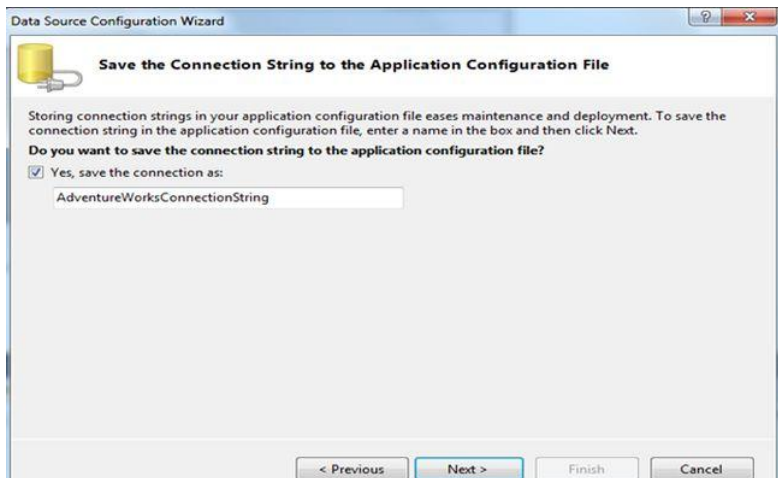
6. Click on "New Connection" to create a new connection to your data source.

To add a connection:

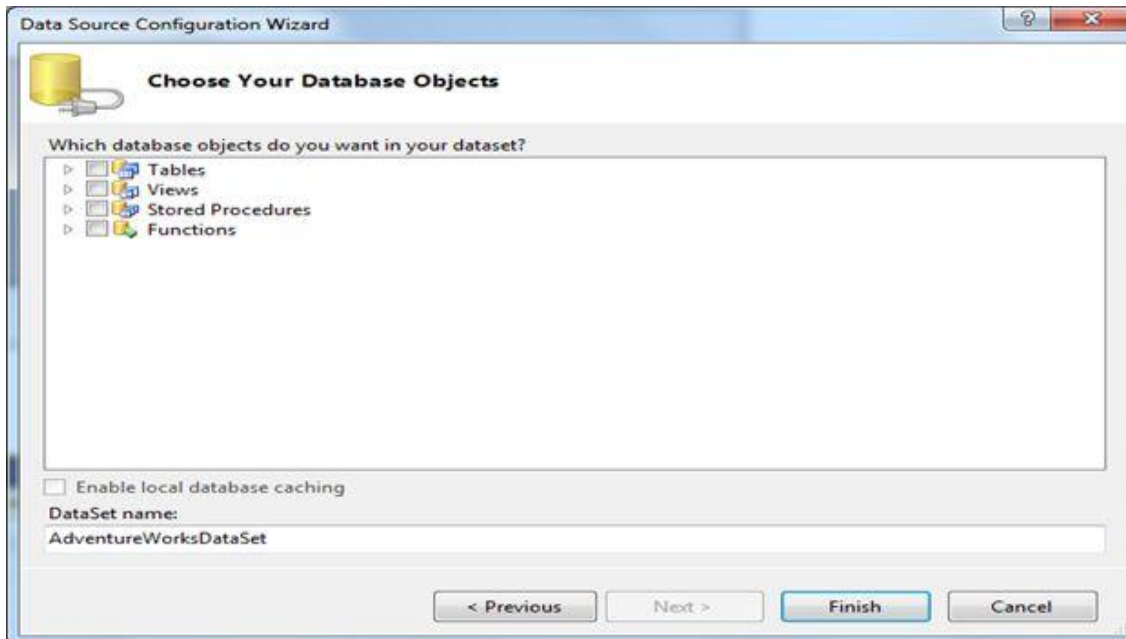
- Provide the server name (in my case it is (.))
- If your server is not using Windows authentication then select "Use SQL Server authentication".
- Provide the username and password.
- Provide the database name AdventureWorks from the Select or enter a database name drop-down list.
- Click on the Test Connection Button. If everything goes well you will see a message box saying that Test the connection succeeded as shown in the following figure.



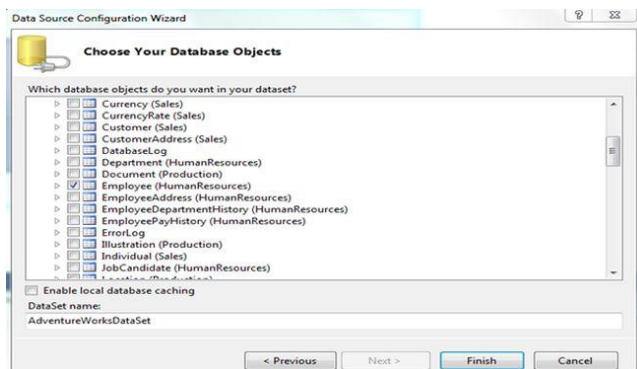
2. Click the OK button.
3. Click the OK button on the Add Connection dialog box.
4. Select Yes, Include sensitive data in the connection string, and click the "Next" button in the DataSource configuration wizard. You will get a page as displayed in the following figure.



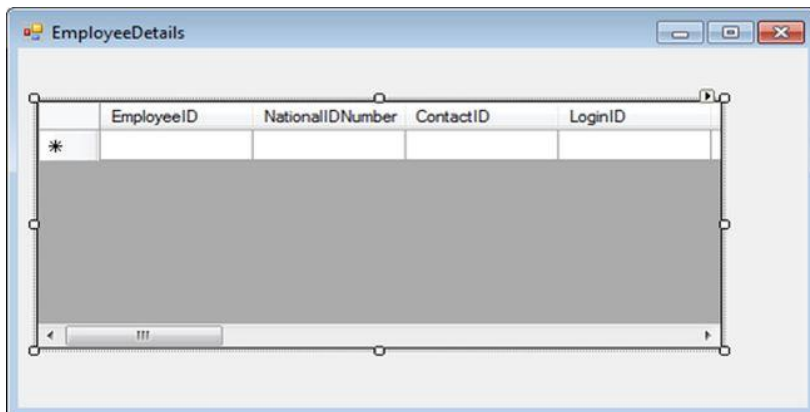
5. Ensure that the Yes, save the connection as the check box is selected and the AdventureWorksConnectionString string appears in the TextBox.
6. Click the "Next" button. The Choose Your Database Objects page is displayed as shown in the following figure.



7. Expand the table node and select the HumanResources.Employee table is shown in the following figure.



8. . Click the "Finish" button. The form is displayed, as shown in the following figure.



9. Press F5 to execute the application. You will get the following output.

EmployeeDetails

	EmployeeID	NationalIDNumber	ContactID	LoginID
▶	1	14417807	1209	adventure-works..
	2	253022876	1030	adventure-works..
	3	509647174	1002	adventure-works..
	4	112457891	1290	adventure-works..
	5	480168528	1009	adventure-works..
	6	24756624	1028	adventure-works..
	7	200700750	1070	adventure-works..