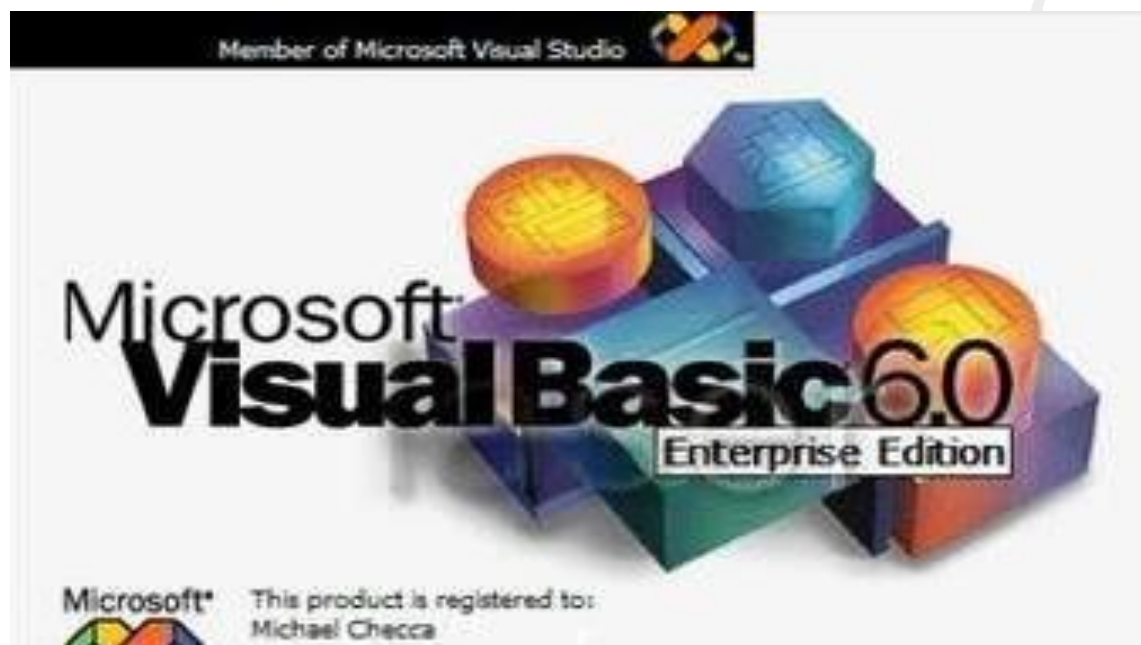


VISUAL PROGRAMMING
(COURSE MATERIAL)
II-B.COM(COMPUTER APPLICATION)
SUB CODE:16CCCCA8



CORE COURSE –VIII
VISUAL PROGRAMMING

Objective

To enable the students to know about the visual programming and its applications

UNIT-I:

Introduction to Visual Basic, Integrated development environment features – Forums – Controls – Events – Methods – Properties - Uses of Property Window – Code Window (Code Behind File) – Variable declaration.

UNIT-II:

Scope of Variables – Constant – Array – Loops in Visual Basic: For ... Next, While, Do...While - Select statements: if...end if - if...else if...end if - Select...Case End Case -

UNIT-III:

Standard Controls: Form - Text Box – Command Button – Label Box – Check Box – Frame Control – Combo Box – List Box – Radio Button - Image Control - Picture Box – Timer.

UNIT-IV:

File System – Drive, DirList, File List Box – Introduction to Built-in-Active X control tool bar – Tree view – Menu Editor – Command dialog control – Rich Text Box.

UNIT-V:

Introduction to Database – MS Access – Data Grid (Accessing Data Base data) – Open data base connectivity – Introduction to Dot Net: IDE – Execution Procedures – CLR – CTS.

Text and Reference Books (Latest revised edition only)

1. Mastering Visual Basic 6 – BPB Publications, New Delhi.
2. Mohammed Azam, Programming with Visual basic 6.0 – Vikas Publishing House.
3. Test Your Vb.Net Skills: Language Elements Part 1 Paperback – 1 Dec 2000
by Yashavant P. Kanetkar (Author), Asang Dani, BPB Publications, New Delhi.

VERIFIED BY

VERIFIED BY

PRINCIPAL

HEAD OF THE DEPARTMENT

P.INDHU MCA.,M.Phil., ASST. PROFESSOR,ANNAI WOMEN'S COLLEGE,KARUR.

UNIT – I

Introduction to Visual Basic :

In this chapter we begin learning about the fundamentals of programming and Visual Basic .NET. First we examine the two elements that are required by every practical Visual Basic program: the screens and instructions seen by the user, and the –behind the scenes processing that is done by the program. We then present the basic design windows that you must be familiar with to produce such programs. Finally, we show you how to use these design windows to create the visual user interface, or GUI, and then add processing instructions.

What is Visual Basic?

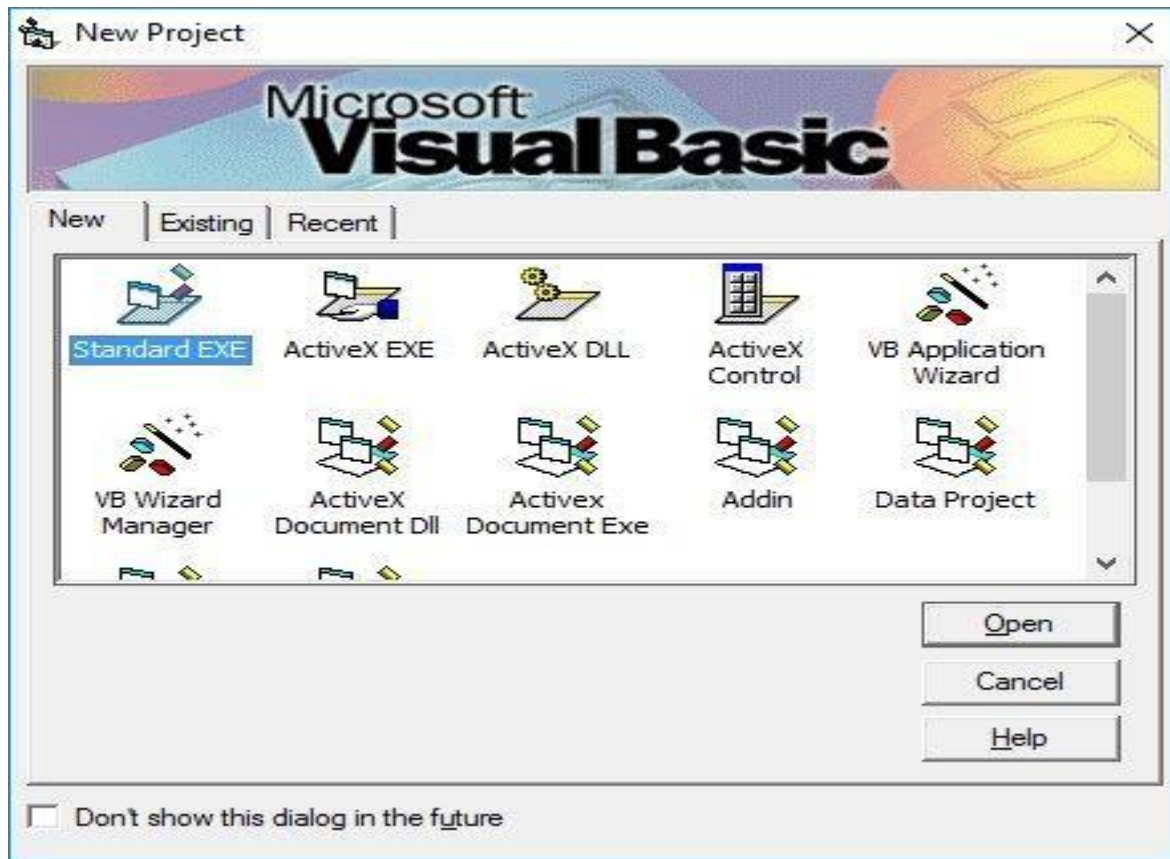
Visual Basic is a third-generation event-driven programming language first released by Microsoft in 1991. It evolved from the earlier DOS version called BASIC. BASIC means Beginners' All-purpose Symbolic Instruction Code. Since then Microsoft has released many versions of Visual Basic, from Visual Basic 1.0 to the final version Visual Basic 6.0. Visual Basic is a user-friendly programming language designed for beginners, and it enables anyone to develop GUI window applications easily. What is Visual Basic?

In 2002, Microsoft released Visual Basic.NET(VB.NET) to replace Visual Basic 6. Thereafter, Microsoft declared VB6 a legacy programming language in 2008. Fortunately, Microsoft still provides some form of support for VB6. VB.NET is a fully object-oriented programming language implemented in the .NET Framework. It was created to cater for the development of the web as well as mobile applications. However, many developers still favor Visual Basic 6.0 over its successor Visual Basic.NET.

The Visual Basic 6 Integrated Development Environment

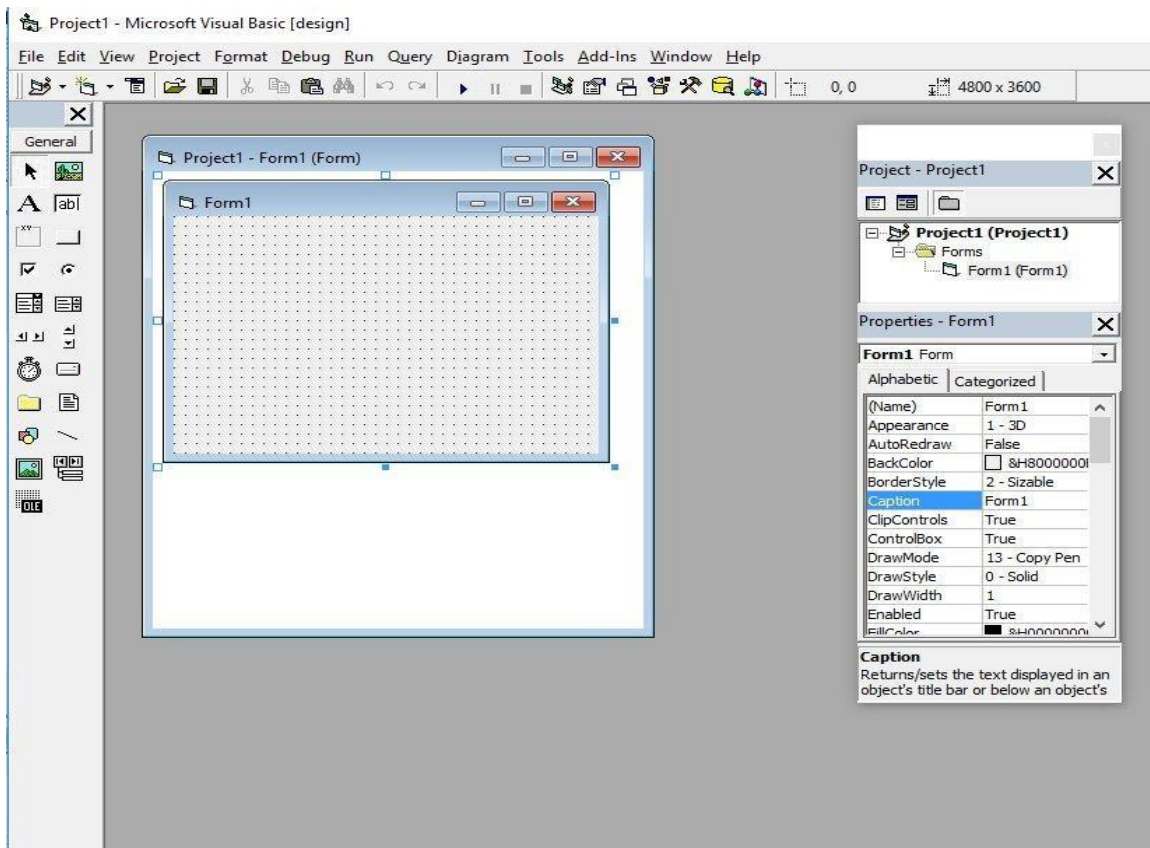
Before you can write programs in VB 6, you need to install Visual Basic 6 compiler on your computer. You can purchase a copy of Microsoft Visual Basic 6.0 Learning Edition or Microsoft Visual Basic Professional Edition from Amazon.com, both are vb6 compilers. Besides, you can also buy it from eBay at Microsoft Visual Basic 6.0 6 Professional PRO MSDN Library Manual Service Pack. If you have already installed Microsoft Office in your PC or laptop, you can also use the built-in Visual Basic Application in Excel to start creating Visual Basic programs without having to spend extra cash to buy the VB6 compiler.

You can also install VB6 on Windows 10 but you need to follow certain steps otherwise the installation will fail. First, you need to run setup as administrator. Next, you need to use custom installation. Clear the checkbox for Data Access. If you don't, set up will hang at the end of the installation. Finally, click next and wait for the installation to complete. For complete instructions, please follow this link [Install VB6 on Windows 10](#)



You can choose to either start a new project, open an existing project or select a list of recently opened programs. A project is a collection of files that make up your application. There are various types of applications that we could create, however, we shall concentrate on creating Standard EXE programs (EXE means executable). Before you begin, you must think of an application that preferably have commercial ,educational or recreational value. Next, click on the Standard EXE icon to go into the actual Visual Basic 6 programming environment.

When you start a new Visual Basic 6 Standard EXE project, you will be presented with the Visual Basic 6 Integrated Development Environment (IDE). The Visual Basic 6 Integrated Programming Environment is shown in Figure 1.2. It consists of the toolbox, the form, the project explorer and the properties window.



VBProgramming Environment

The Form is the primary building block of a Visual Basic 6 application. A Visual Basic 6 application can actually comprise many forms, but we shall focus on developing an application with one form first. We will learn how to develop applications with multiple forms later. Before you proceed to build the application, it is a good practice to save the project first. You can save the project by selecting **Save** Project from the File menu, assign a name to your project and save it in a certain folder. You shall now proceed to learn Visual Basic programming from the next lesson onwards.

Integrated Development Environment Features:

The Visual Basic integrated development environment (IDE) consists of the following elements.

Menu Bar:

Displays the commands you use to work with Visual Basic. Besides the standard File, Edit, View, Window, and Help menus, menus are provided to access functions specific to programming such as Project, Format, or Debug.

Context Menus:

Contain shortcuts to frequently performed actions. To open a context menu, click the right mouse button on the object you're using. The specific list of shortcuts available from context menus depends on the part of the environment where you click the right mouse button. For example, the context menu displayed when you right click on the Toolbox lets you display

the Components dialog box, hide the Toolbox, dock or undock the Toolbox, or add a custom tab to the Toolbox.

Toolbars:

Provide quick access to commonly used commands in the programming environment. You click a button on the toolbar once to carry out the action represented by that button. By default, the Standard toolbar is displayed when you start Visual Basic. Additional toolbars for editing, form design, and debugging can be toggled on or off from the Toolbars command on the View menu.

Toolbars can be docked beneath the menu bar or can "float" if you select the vertical bar on the left edge and drag it away from the menu bar.

Toolbox:

Provides a set of tools that you use at design time to place controls on a form. In addition to the default toolbox layout, you can create your own custom layouts by selecting Add Tab from the context menu and adding controls to the resulting tab.

Project Explorer Window:

Lists the forms and modules in your current project. A *project* is the collection of files you use to build an application.

Lists the property settings for the selected form or control. A *property* is a characteristic of an object, such as size, caption, or color.

Object Browser:

Lists objects available for use in your project and gives you a quick way to navigate through your code. You can use the Object Browser to explore objects in Visual Basic and other applications, see what methods and properties are available for those objects, and paste code procedures into your application.

Form Designer:

Serves as a window that you customize to design the interface of your application. You add controls, graphics, and pictures to a form to create the look you want. Each form in your application has its own form designer window.

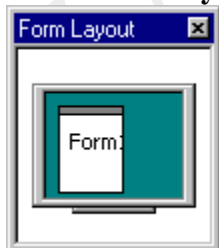
Code Editor Window:

Serves as an editor for entering application code. A separate code editor window is created for each form or code module in your application.

Form Layout Window:

The Form Layout window (Figure 2.2) allows you to position the forms in your application using a small graphical representation of the screen.

The Form Layout window



Immediate, Locals, and Watch Windows

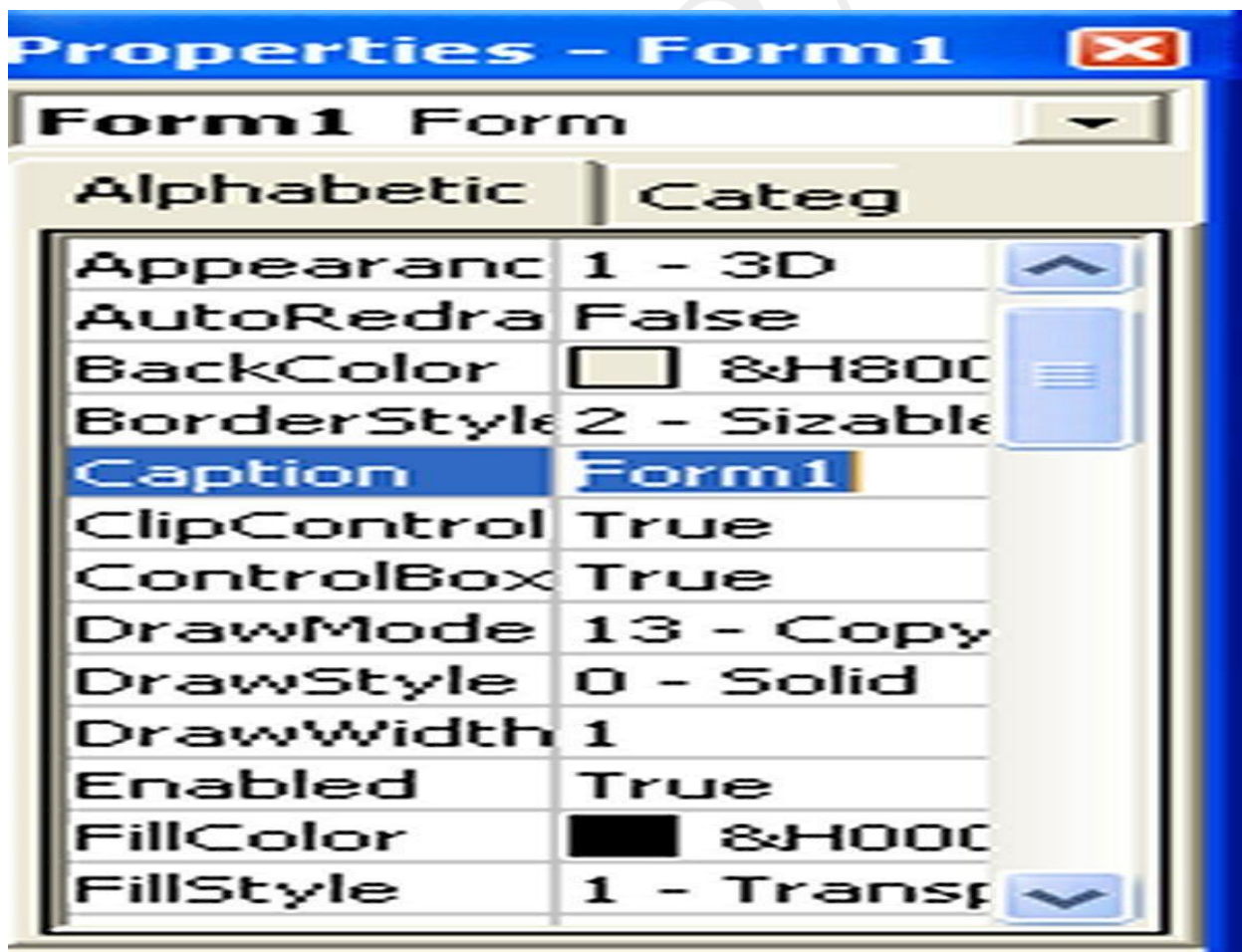
These additional windows are provided for use in debugging your application. They are only available when you are running your application within the IDE.

The Control Properties:

Before writing an event procedure for the control to respond to an event, you have to set certain properties for the control to determine its appearance and how will it work with the event procedure. You can set the properties of the controls in the properties window or at runtime.

It is a typical properties window for a form. In the properties window, the item appears at the top part is the object currently selected. At the bottom part, the items listed in the left column represent the names of various properties associated with the selected object while the items listed in the right column represent the states of the properties. Properties can be set by highlighting the items in the right column then change them by typing or selecting the options available.

For example, in order to change the caption, just highlight Form1 under the name Caption and change it to other names. You may also alter the appearance of the form by setting it to 3D or flat, change its foreground and background color, change the font type and font size, enable or disable, minimize and maximize buttons and more.



Properties Window

You can also change the properties at runtime to give special effects such as change of color, shape, animation effect and so on. Example 3.1 show the code that will change the form color to red every time the form is loaded. VB uses the hexadecimal system to represent the color. You can check the color codes in the properties windows which are showing up under ForeColor and BackColor .

Example : Program to change background color

This example changes the background colour of the form using the **BackColor** property.

```
Private Sub Form_Load()  
Form1.Show  
Form1.BackColor = &H000000FF&  
End Sub
```

Example 3.2: Program to change shape

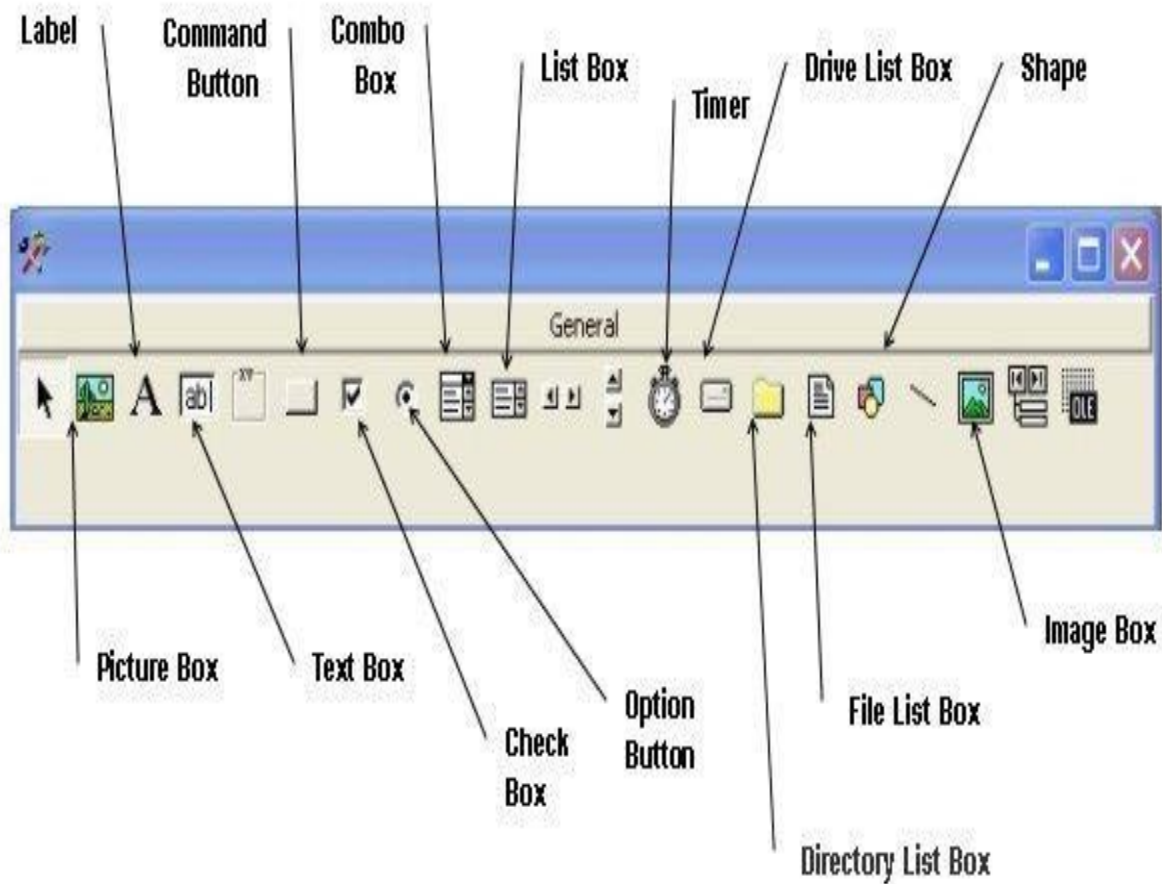
This example is to change the control's Shape using the **Shape** property. This code will change the shape to a circle at runtime.

```
Private Sub Form_Load()  
Shape1.Shape = 3  
End Sub
```

We are not going into the details on how to set the properties yet. However, we would like to stress a few important points about setting up the properties.

- You should set the Caption Property of a control clearly so that a user knows what to do with that command.
- Use a meaningful name for the Name Property because it is easier to write and read the event procedure and easier to debug or modify the programs later.
- One more important property is whether to make the control enabled or not.
- Finally, you must also considering making the control visible or invisible at runtime, or when should it become visible or invisible.

Controls



Toolbox

The TextBox:

The text box is the standard control for accepting input from the user as well as to display the output. It can handle string (text) and numeric data but not images or pictures. A string entered into a text box can be converted to a numeric data by using the function `Val(text)`. The following example illustrates a simple program that processes the input from the user.

Example :

In this program, two text boxes are inserted into the form together with a few labels. The two text boxes are used to accept inputs from the user and one of the labels will be used to display the sum of two numbers that are entered into the two text boxes. Besides, a command button is also programmed to calculate the sum of the two numbers using the plus operator. The

P.INDHU MCA.,M.Phil., ASST. PROFESSOR,ANNAI WOMEN'S COLLEGE,KARUR.

program use creates a variable sum to accept the summation of values from text box 1 and text box 2. The procedure to calculate and to display the output on the label is shown below.

```
Private Sub Command1_Click()  
    'To add the values in TextBox1 and TextBox2  
    Sum = Val(Text1.Text) + Val(Text2.Text)  
    'To display the answer on label 1  
    Label1.Caption = Sum  
End Sub
```

The output is shown

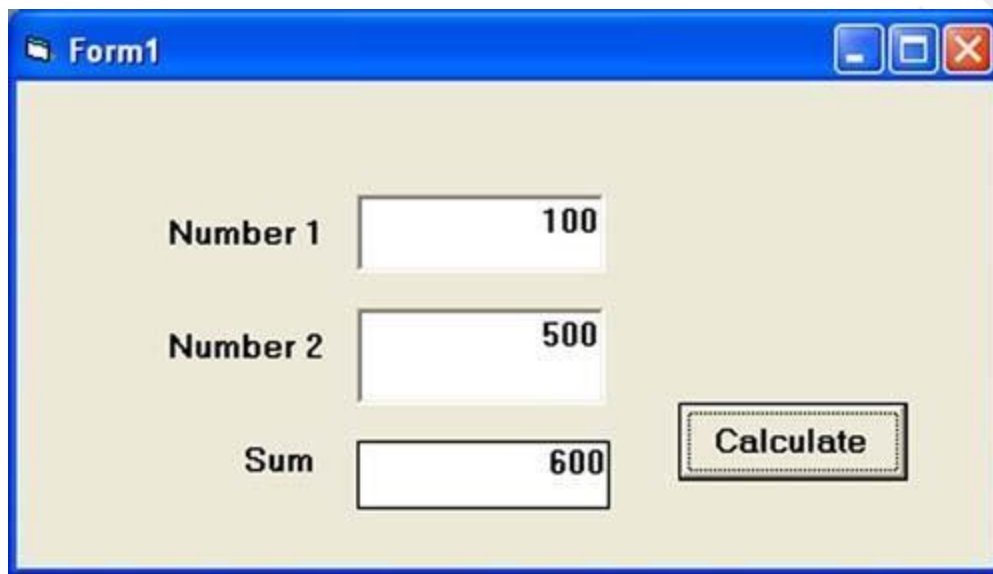


Figure 3.3

The Label:

The label is a very useful control for Visual Basic, as it is not only used to provide instructions and guides to the users, it can also be used to display outputs. One of its most important properties is **Caption**. Using the syntax **Label.Caption**, it can display text and numeric data. You can change its caption in the properties window and also at runtime.

The Command Button:

The command button is one of the most important controls as it is used to execute commands. It displays an illusion that the button is pressed when the user click on it. The most common event associated with the command button is the Click event, and the syntax for the procedure is

```
Private Sub Command1_Click ()
```

Statements

P.INDHU MCA.,M.Phil., ASST. PROFESSOR,ANNAI WOMEN'S COLLEGE,KARUR.

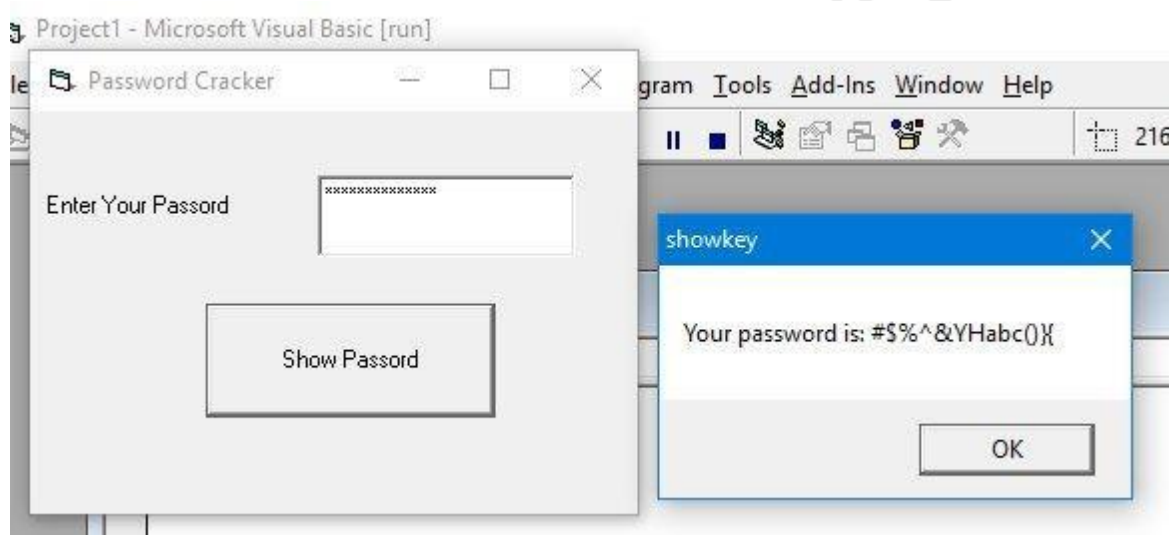
End Sub

A Simple Password Cracker

In this program, we want to crack a secret password entered by the user. In the design phase, insert a command button and change its name to cmd_ShowPass. Next, insert a TextBox and rename it as Txt_Password and delete Text1 from the Text property. Besides that, set its PasswordChr to *. Now, enter the following code in the code window.

```
Private Sub cmd_ShowPass_Click()  
    Dim yourpassword As String  
    yourpassword = Txt_Password.Text  
    MsgBox ("Your password is: " & yourpassword)  
End Sub
```

Run the program and enter a password, then click on the Show Password button to reveal the password.



The Password Cracker

You can also reveal the password by setting the PasswordChr property back to normal mode, as follows:

```
Private Sub cmd_ShowPass_Click()  
    Dim yourpassword As String  
    Txt_Password.PasswordChar = ""  
End Sub
```

The PictureBox

The Picture Box is one of the controls that is used to handle graphics. You can load a picture at design phase by clicking on the picture item in the properties window and select the picture from the selected folder. You can also load the picture at runtime using the **LoadPicture** method. For example, the statement will load the picture grape.gif into the picture box.

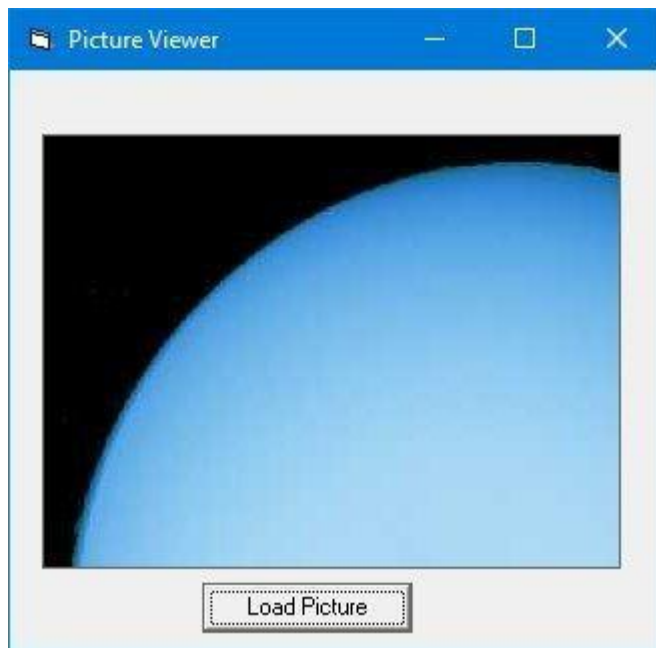
```
Picture1.Picture=LoadPicture ("C:\VBprogram\Images\grape.gif")
```

Example : Loading Picture

In this program, insert a command button and a picture box. Enter the following code:

```
Private Sub cmd_LoadPic_Click()  
MyPicture.Picture = LoadPicture("C:\Users\admin.DESKTOP-G1G4HEK\Documents\My  
Websites\vbtutor\vb6\images\uranus.jpg")  
End Sub
```

* You must ensure the path to access the picture is correct. Besides that, the image in the picture box is not resizable.



The Picture Viewer

The Image Control

The Image Control is another control that handles images and pictures. It functions almost identically to the pictureBox. However, there is one major difference, the image in an

P.INDHU MCA.,M.Phil., ASST. PROFESSOR,ANNAI WOMEN'S COLLEGE,KARUR.

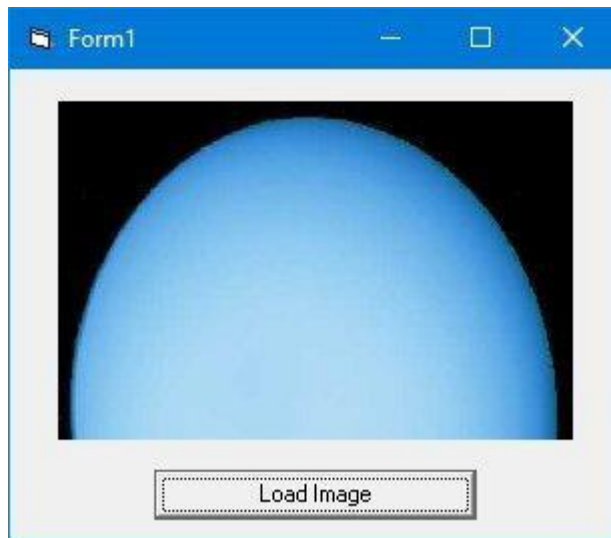
Image Box is stretchable, which means it can be resized. This feature is not available in the PictureBox. Similar to the Picture Box, it can also use the LoadPicture method to load the picture. For example, the statement loads the picture grape.gif into the image box.

```
Image1.Picture=LoadPicture ("C:\VBprogram\Images\grape.gif")
```

Example: Loading Image

In this program, we insert a command button and an image control into the form. Besides that, we set the image Stretch property to true. Next, enter the following code:

```
Private Sub cmd_LoadImg_Click()  
    MyImage.Picture = LoadPicture("C:\Users\admin.DESKTOP-G1G4HEK\Documents\My  
    Websites\vb6\tutor\vb6\images\uranus.jpg")  
End Sub
```



The Image Viewer

The ListBox

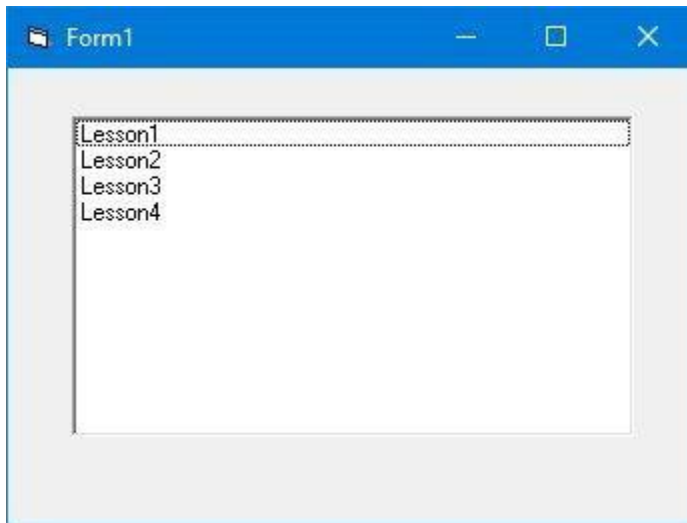
The function of the ListBox is to present a list of items where the user can click and select the items from the list. In order to add items to the list, we can use the **AddItem method**. For example, if you wish to add a number of items to list box 1, you can key in the following statements

Example :

```
Private Sub Form_Load ( )  
    List1.AddItem -Lesson1|  
    List1.AddItem -Lesson2|  
    List1.AddItem -Lesson3|
```

```
List1.AddItem -Lesson4!  
End Sub
```

The Output



The ListBox

The items in the list box can be identified by the **ListIndex** property, the value of the ListIndex for the first item is 0, the second item has a ListIndex 1, and the third item has a ListIndex 2 and so on

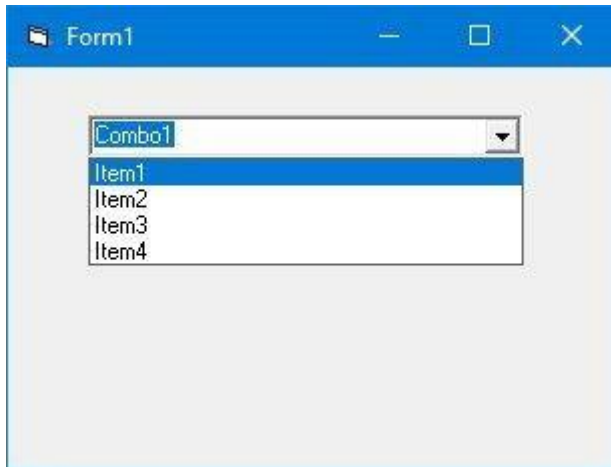
The ComboBox

The function of the Combo Box is also to present a list of items where the user can click and select the items from the list. However, the user needs to click on the small arrowhead on the right of the combo box to see the items which are presented in a drop-down list. In order to add items to the list, you can also use the **AddItem method**. For example, if you wish to add a number of items to Combo box 1, you can key in the following statements

Example :

```
Private Sub Form_Load ( )  
Combo1.AddItem "Item1"  
Combo1.AddItem "Item2"  
Combo1.AddItem "Item3"  
Combo1.AddItem "Item4"  
End Sub
```


The Output



The ListBox

The CheckBox

The Check Box control lets the user select or unselect an option. When the Check Box is checked, its value is set to 1 and when it is unchecked, the value is set to 0. You can include the statements `Check1.Value=1` to mark the Check Box and `Check1.Value=0` to unmark the Check Box, as well as use them to initiate certain actions. For example, the program in Example 3.4 will show which items are selected in a message box.

Example :

```
Private Sub Cmd_OK_Click()  
If Check1.Value = 1 And Check2.Value = 0 And Check3.Value = 0 Then  
    MsgBox "Apple is selected"  
ElseIf Check2.Value = 1 And Check1.Value = 0 And Check3.Value = 0 Then  
    MsgBox "Orange is selected"  
ElseIf Check3.Value = 1 And Check1.Value = 0 And Check2.Value = 0 Then  
    MsgBox "Orange is selected"  
ElseIf Check2.Value = 1 And Check1.Value = 1 And Check3.Value = 0 Then  
    MsgBox "Apple and Orange are selected"  
ElseIf Check3.Value = 1 And Check1.Value = 1 And Check2.Value = 0 Then  
    MsgBox "Apple and Pear are selected"  
ElseIf Check2.Value = 1 And Check3.Value = 1 And Check1.Value = 0 Then  
    MsgBox "Orange and Pear are selected"  
Else  
    MsgBox "All are selected"  
End If  
End Sub
```

The Output



The ListBox

The OptionButton

The OptionButton control also lets the user select one of the choices. However, two or more Option buttons must work together because as one of the option buttons is selected, the other Option button will be unselected. In fact, only one Option Box can be selected at one time. When an option box is selected, its value is set to `True` and when it is unselected; its value is set to `False`.

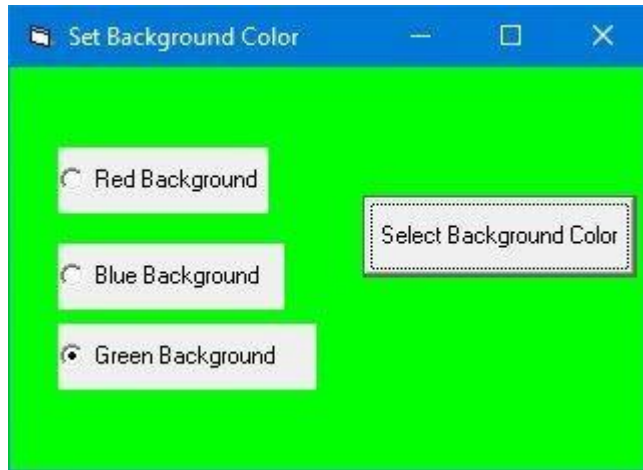
Example:

In this example, we want to change the background color of the form according to the selected option. We insert three option buttons and change their captions to "Red Background", "Blue Background" and "Green Background" respectively. Next, insert a command button and change its name to `cmd_SetColor` and its caption to "Set Background Color". Now, click on the command button and enter the following code in the code window:

```
Private Sub cmd_SetColor_Click()  
    If Option1.Value = True Then  
        Form1.BackColor = vbRed  
    ElseIf Option2.Value = True Then  
        Form1.BackColor = vbBlue  
    Else
```

```
Form1.BackColor = vbGreen
End If
End Sub
```

Run the program, select an option and click the "Set Background Color" produces the output, as shown ,

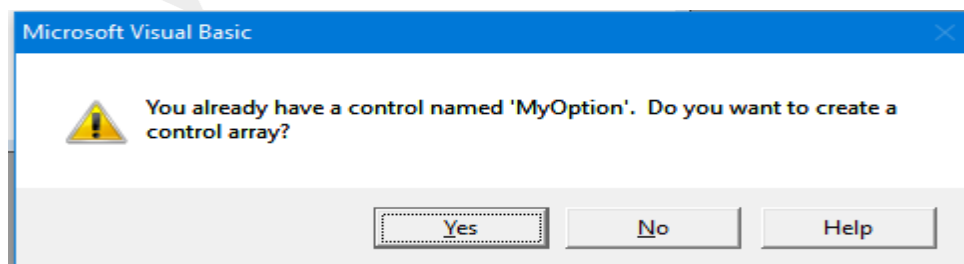


The Shape Control

In the following example, the shape control is placed in the form together with six OptionButtons. To determine the shape of the shape control, we use the shape property. The property values of the shape control are 0, 1, and 2,3,4,5 which will make it appear as a rectangle, a square, an oval shape, a circle, a rounded rectangle and a rounded square respectively.

Example :

In this example, we insert six option buttons. It is better to make the option buttons into a control array as they perform similar action, i.e to change shape. In order to create a control array, click on the first option button, rename it as MyOption. Next, click on the option button and select copy then paste. After clicking the paste button, a popup dialog (Figure 3.11) will ask you whether you wish to create a control array, select yes. The control array can be accessed via its index value, MyOption(Index). In addition, we also insert a shape control.



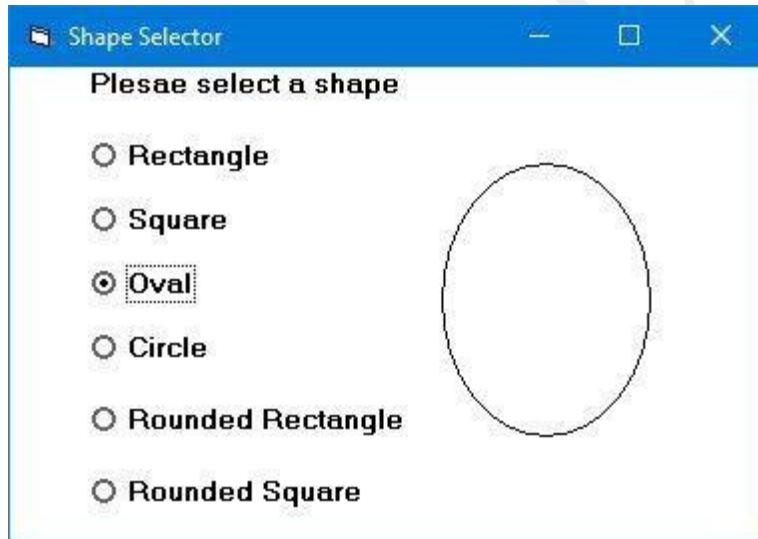
Now, enter the code in the code window. We use the If..Then..Else program structure to determine which option button is selected by the user.

```
Private Sub MyOption_Click(Index As Integer)
If Index = 0 Then
MyShape.Shape = 0
ElseIf Index = 1 Then
MyShape.Shape = 1
ElseIf Index = 2 Then
MyShape.Shape = 2
ElseIf Index = 3 Then
MyShape.Shape = 3
ElseIf Index = 4 Then
MyShape.Shape = 4
ElseIf Index = 5 Then
MyShape.Shape = 5

End If

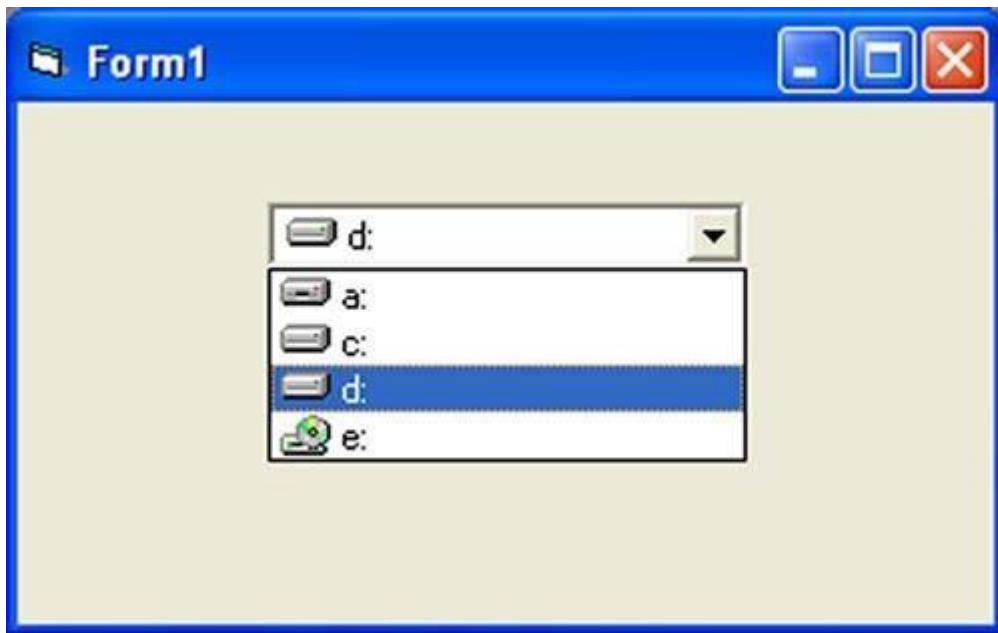
End Sub
```

Run the program and you can change the shape of the shape control by clicking one of the option buttons. The output



The DriveListBox

The DriveListBox is for displaying a list of drives available in your computer. When you place this control into the form and run the program, you will be able to select different drives from your computer as shown in Figure 3.13

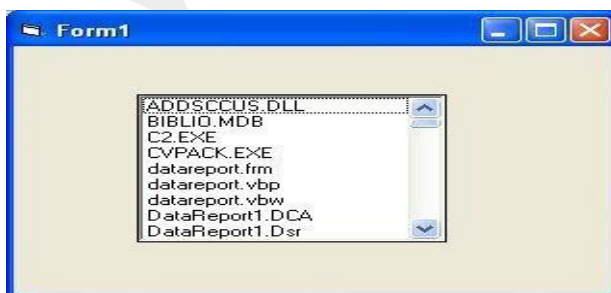
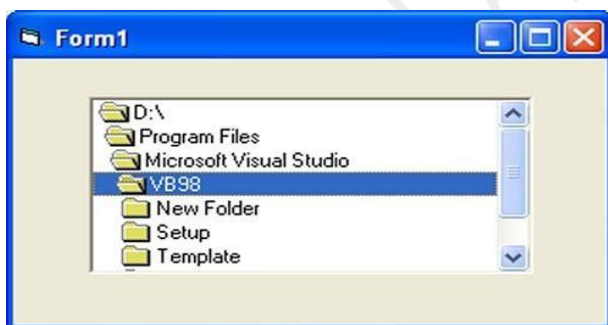


The Drive List Box

The DirListBox

The DirListBox means the Directory List Box. It is for displaying a list of directories or folders in a selected drive. When you place this control into the form and run the program, you will be able to select different directories from a selected drive in your computer as shown .

The DirListBox



The FileListBox

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.

Clicking on a button, or entering some text in a text box, or clicking on a menu item, all are examples of events. 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

Following is an example, which shows how to handle mouse events. Take the following steps:

- Add three labels, three text boxes and a button control in the form.

- Change the text properties of the labels to - Customer ID, Name and Address, respectively.
- Change the name properties of the text boxes to txtID, txtName and txtAddress, respectively.
- Change the text property of the button to 'Submit'.
- Add the following code in the code editor window:

```
Public Class Form1
```

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

```
' Set the caption bar text of the form.
```

```
Me.Text = "tutorialspont.com"
```

```
End Sub
```

```
Private Sub txtID_MouseEnter(sender As Object, e As EventArgs)_
```

```
Handles txtID.MouseEnter
```

```
'code for handling mouse enter on ID textbox
```

```
txtID.BackColor = Color.CornflowerBlue
```

```
txtID.ForeColor = Color.White
```

```
End Sub
```

```
Private Sub txtID_MouseLeave(sender As Object, e As EventArgs) _
```

```
Handles txtID.MouseLeave
```

```
'code for handling mouse leave on ID textbox
```

```
txtID.BackColor = Color.White
```

```
txtID.ForeColor = Color.Blue
```

```
End Sub
```

```
Private Sub txtName_MouseEnter(sender As Object, e As EventArgs) _
```

```
Handles txtName.MouseEnter
```

```
'code for handling mouse enter on Name textbox
```

```
txtName.BackColor = Color.CornflowerBlue
```

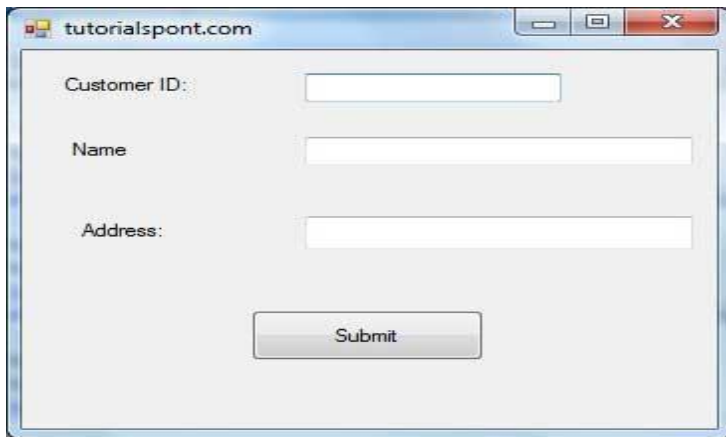
```

txtName.ForeColor = Color.White
End Sub
Private Sub txtName_MouseLeave(sender As Object, e As EventArgs) _
    Handles txtName.MouseLeave
    'code for handling mouse leave on Name textbox
    txtName.BackColor = Color.White
    txtName.ForeColor = Color.Blue
End Sub
Private Sub txtAddress_MouseEnter(sender As Object, e As EventArgs) _
    Handles txtAddress.MouseEnter
    'code for handling mouse enter on Address textbox
    txtAddress.BackColor = Color.CornflowerBlue
    txtAddress.ForeColor = Color.White
End Sub
Private Sub txtAddress_MouseLeave(sender As Object, e As EventArgs) _
    Handles txtAddress.MouseLeave
    'code for handling mouse leave on Address textbox
    txtAddress.BackColor = Color.White
    txtAddress.ForeColor = Color.Blue
End Sub

Private Sub Button1_Click(sender As Object, e As EventArgs) _
    Handles Button1.Click
    MsgBox("Thank you " & txtName.Text & ", for your kind cooperation")
End Sub
End Class

```

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



Try to enter text in the text boxes and check the mouse events:



Events:

Following are the various keyboard events related with a Control class:

- **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/p>
- **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

Let us continue with the previous example to show how to handle keyboard events. The code will verify that the user enters some numbers for his customer ID and age.

- Add a label with text Property as 'Age' and add a corresponding text box named txtAge.
- Add the following codes for handling the **KeyUp** events of the text box txtID.

```

• Private Sub txtID_KeyUP(sender As Object, e As KeyEventArgs) _
•     Handles txtID.KeyUp
•     If (Not Char.IsNumber(ChrW(e.KeyCode))) Then
•         MessageBox.Show("Enter numbers for your Customer ID")
•         txtID.Text = " "
•     End If
• End Sub

```

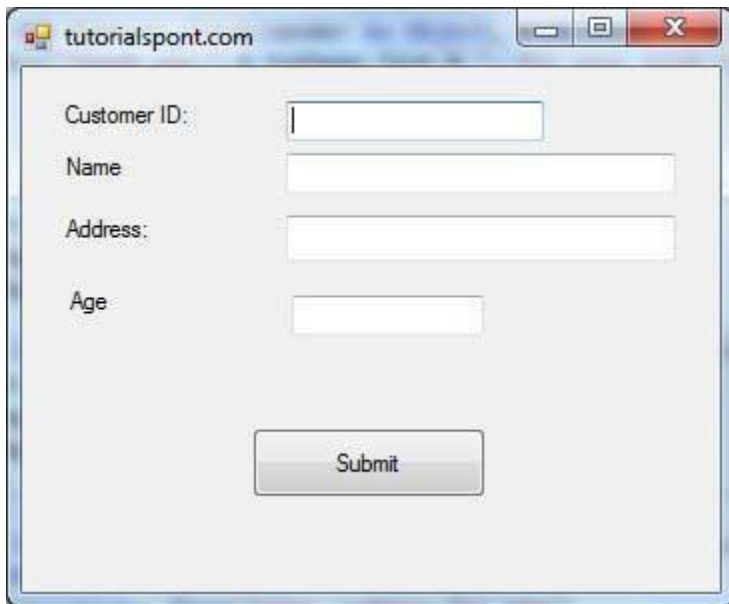
- Add the following codes for handling the **KeyUp** events of the text box txtID.

```

• Private Sub txtAge_KeyUP(sender As Object, e As KeyEventArgs) _
•     Handles txtAge.KeyUp
•     If (Not Char.IsNumber(ChrW(e.keyCode))) Then
•         MessageBox.Show("Enter numbers for age")
•         txtAge.Text = " "
•     End If
• End Sub

```

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



If you leave the text for age or ID as blank or enter some non-numeric data, it gives a warning message box and clears the respective text:



Methods:

A method is a built-in function that works with an object. A method has no meaning without its object. But this is very important to note that a method is not a special thing in VB6 but this is just a function, a block of code, which is associated with an object. The method causes the object to perform a specific task. See the following examples:

Example:

```
Form1.Hide
```

The Hide method hides the form Form1 without unloading from memory. So a method performs an action for a particular object.

Example:

```
'In Form1  
Cls  
Text1.SetFocus
```

The Cls method clears the object, generally the form and PictureBox object. In the above code, the method Cls is written without any object. But I said before that it has no meaning without an object. This is because, in this example, Cls means Form1.Cls.

When you're coding in the form module, you can omit the form name while invoking a form's method.

In the above code, SetFocus is a method that moves the focus to TextBox. Here also the method causes the object to do something. There are so many methods in Visual Basic.

Print

The Print method prints something on the object, generally on form and PictureBox.

Example:

```
Form1.Print "Your text here"  
'Or,  
Print "Your text here"
```

```
Picture1.print "Your text here"
```

Here Picture1 is the PictureBox control.

Move

The Move method moves an object.

Syntax:

```
Object.Move Left, [Top], [Width], [Height]
```

The parameters enclosed in square brackets "[]" are always optional. Do not type the brackets in your code.

Example:

```
Form1.Move 20, 50, 2000, 2000
```

ZOrder

VB controls are overlapped, the ZOrder method sets the position of the control. You can position the control in front of or behind other controls. To position the control in front of other controls, simply write the method without any argument. To position it behind other controls, pass 1 as an argument.

Example:

```
Text1.ZOrder 1 'moves it behind other controls  
Text1.ZOrder 'moves it to the front
```

Cls

The Cls method clears the form or PictureBox. But it does not mean that it will clear anything on the form or PictureBox. The Cls method clears the contents that are created or drawn by the graphic methods such as Print, Circle etc.

Example:

```
Print "hello"  
Picture1.Print "welcome"  
Picture1.Cls  
Form1.Cls 'or Cls
```

Here Picture1 is the PictureBox. This program will not print anything as Cls method is executed after printing a text.

Show

The Show method is generally used with multiple forms. It shows an object, a form.

Example:

```
Form2.Show
```

Refresh

The Refresh method repaints or redraws an object completely. You can immediately update the content of a control using the Refresh method.

Common Properties in VB6

Back Color and Fore Color

The Back Color property sets the background color of an object while the ForeColor property changes the foreground color used to display text.

You can set these properties either from Properties Window or you may wish to set in run-time.

Example:

When you click on command1 button, the code in the Click event procedure is executed.

```
Private Sub cmdChangeColor_Click()  
    Label1.BackColor = vbRed  
    Label1.ForeColor = vbBlue  
End Sub
```

On execution, the background color of the label will be red and label's text color will be blue.

'vbRed' and 'vbBlue' are the color constants.

Another example:

```
Private Sub cmdChangeColor_Click()  
    Label1.BackColor = vbRed  
    Label1.ForeColor = vbBlue  
    Form1.BackColor = vbGreen  
    Text1.BackColor = vbBlack  
    Text1.ForeColor = vbYellow  
    Frame1.BackColor = vbWhite  
End Sub
```

The color can also be expressed in hexadecimal code.

Example:

```
Private Sub cmdChangeColor_Click()  
    Label1.BackColor = &HFF&  
    Label1.ForeColor = &HC0000  
End Sub
```

In this case, you have to copy the hexadecimal code from Properties Window.

Font

You can set the font property from the Properties Window. See the below example to set property in run time.

Example:

```
Private Sub cmdChangeFont_Click()  
    Text1.FontSize = 16  
    Text1.FontBold = True  
    Text1.FontItalic = True  
    Text1.Font = "Tahoma"  
End Sub
```

The above block of code can be written in the following way too.

```
Private Sub cmdChangeFont_Click()  
    Text1.Font.Size = 16  
    Text1.Font.Bold = True  
    Text1.Font.Italic = True  
    Text1.Font.Name = "Tahoma"  
End Sub
```

Caption:

It sets the text displayed in the object's title bar or on the object.

Example:

```
Private Sub cmdSetTitle_Click()  
    Form1.Caption = "New Program"  
    Label1.Caption = "Hello"  
    Frame1.Caption = "New frame"  
End Sub
```

'Caption' property of form sets the form's title text. The text to be displayed on label is set.

Text:

It sets the text in a TextBox.

Example:

```
Text1.Text = "New program"
```

Here the text string is assigned to Text1.Text.

The Left, Top, Width and Height properties:

1. The Left Property sets the distance between the internal left edge of an object and the left edge of its container.
2. The Top Property sets the distance between the internal top edge of an object and the top edge of its container.
3. The Width Property sets the width of an object.
4. The Height Property sets the height of an object.

Example:

```
Private Sub cmdChange_Click()  
    Form1.Left = 12000  
    Form1.Top = 7000  
    Form1.Height = 10000  
    Form1.Width = 12000  
End Sub
```

Container:

Moves an object into another container. You don't see it in the Properties Window, it is a run-time only property.

In this case, you need to start with the 'Set' keyword.

Example:

```
Set Command1.Container = Frame1
```

The command1 control will be moved into frame1.

Visible

Determines whether an object is visible or hidden.

Example:

```
Label1.Visible = False
```


Enabled

Determines whether an object can respond to user generated events.

Example:

```
Text1.enabled=False
```

MousePointer and MouseIcon:

The MousePointer property sets the type of mouse pointer over an object. The values of MousePointer property are

0- Default

1- Arrow

2- Cross

3- I-Beam

4- Icon

5- Size etc.

TabIndex and TabStop

The TabStop property indicates whether the user can use the TAB key to give the focus to an object. You can give focus to the objects pressing the TAB key in the daily use software applications.

The TabIndex property sets the tab order of an object.

Example:

```
Command3.TabIndex = 0
```

```
Command2.TabIndex = 1
```

```
Command1.TabIndex = 2
```

When you press the TAB key, the focus will be given on Command3 first, then on Command2 and at last on Command1.

Movable

Determines whether an object is movable.

Locked

Determines whether an object e.g TextBox can be edited.

Tag

Stores any extra data for your program that is used in the code.

Control Box

Indicates whether a control-menu box is displayed on the form at run time. If this property is set to False, the Close Button, Minimize Button and Maximize Button are not shown on the form.

ShowInTaskBar

Determines whether the form appears in the windows taskbar.

StartPosition

Specifies the position of the form when it first appears.

Icon

Sets the icon displayed when the form is minimized at run time.

Label and TextBox Controls

Some basic properties of Label and TextBox control in Visual Basic 6 are explained in this lesson.

Properties:

Properties of label

- **Appearance:** Sets whether or not a label is painted with 3D effects.
- **BackStyle:** Sets whether the background of the label is transparent or opaque.
- **Caption:** Sets the text to be displayed.

Properties of TextBox

- **Enabled:** Determines whether the TextBox can respond to user-generated events.
- **Locked:** Determines whether the TextBox can be edited. You can prevent the user from changing the content of the TextBox.
- **MultiLine:** Indicates whether it can accept multiple lines of text.
- **Alignment:** Aligns the text to left justify, right justify or center.

- **MaxLength:** Sets the maximum number of characters that can be entered. If you want to set a limit on the number of characters that the user can enter, you can do it very easily with the MaxLength property.
- **PasswordChar:** This property is useful for a password program. For example, if u enter '*' as a value to this property, whatever character you type in, it will be shown as '*'.
- **ScrollBars:** Says whether the TextBox has a scroll bar.
- **ToolTipText:** Sets the text displayed when the mouse is paused over the control.

Run-Time properties of TextBox

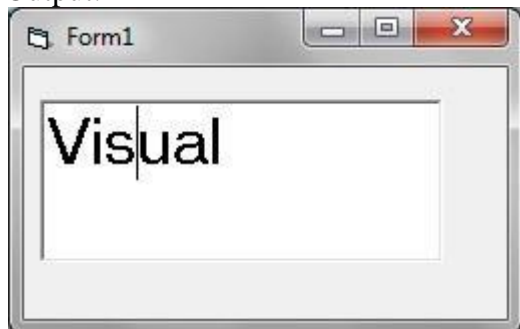
Run-time properties are not visible in the Properties Window. You need to write code to set run-time properties.

SelStart: Sets the position of the blinking cursor that appears in the TextBox Control. When the cursor is at the beginning of the text, the value of SelStart is 0. When its at the end of the text, it returns Len(text1.text). 'Len' is a string function that returns the length of the text or the number of characters in the text. Set SelStart property to move the cursor in the TextBox.

Example:

```
Private Sub Form_Load()
    Text1.Text = "Visual"
    Text1.SelStart = 3
End Sub
```

Output:



SelLength: Sets the number of characters that has been selected or highlighted.

Example:

```
Private Sub Form_Load()
    Text1.Text = "Visual"
    Text1.SelLength = 3
End Sub
```

Output:



SelText: Sets or returns the text that is currently selected or highlighted.

Example:

```
Private Sub Form_Load()  
    Text1.Text = "Visual"  
    Text1.SelLength = 3  
    Text1.Text = Text1.SelText  
End Sub
```

Output:



Properties of CommandButton

Cancel: Indicates whether a command button is the cancel button on the form. That means, if you press ESC key from keyboard, the CommandButton will be fired if the 'Cancel' property is set to true.

Default: Indicates whether a command button is the default button on the form. That means, if you press ENTER key from keyboard, the CommandButton will be fired if the 'Default' property is set to true.

MouseIcon: Sets a custom mouse icon.

MousePointer: Sets the type of mouse pointer that will be shown when the mouse pointer is over the button.

Picture: Sets an image to be displayed when the 'style' property is set to 1.

Run-time property of CommandButton

Value: The value property sets or returns the state of the control. That means, value is true if the CommandButton is pressed. This property is useful for programmatically clicking a button.

Set the Name properties to cmdButton1 and cmdButton2. The default Name properties are Command1 and Command2.

Example:

```
Private Sub cmdButton1_Click()  
    Print "Hello"  
End Sub
```

```
Private Sub cmdButton2_Click()  
    cmdButton1.Value = True  
End Sub
```

When you click on Button2, Button1 is clicked programmatically, then it prints "Hello".

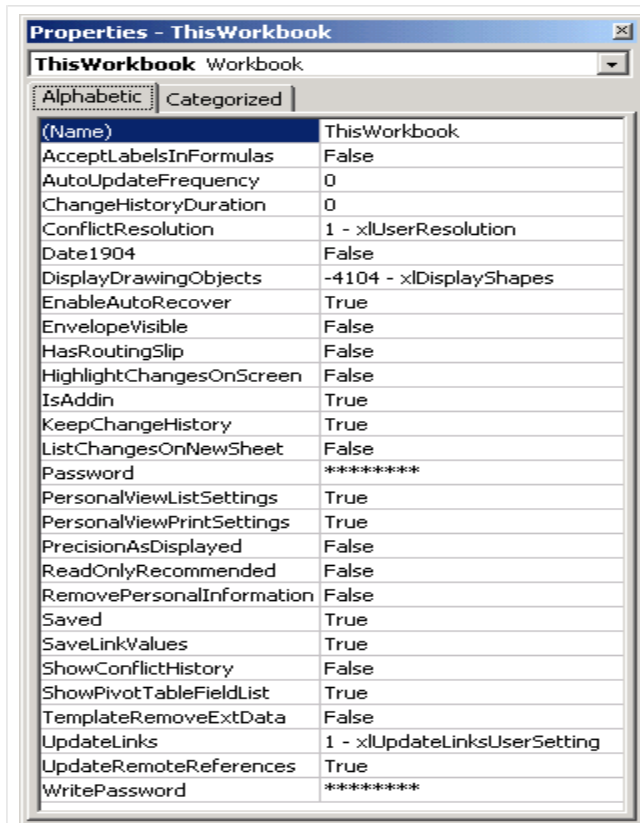
Properties Window and Code Window:

This window allows us to change some of the properties associated with an object at design time.

Displays all the properties of the object (relating to the active project, userforms or control) that can be changed at design time.

(Shift + Tab) - jumps to the object field list

(Ctrl + Shift + X) - jumps to the first property whose name begins with the letter X



© BetterSolutions.com

Note that some of these properties are read only and cannot be changed and others can only be changed at run-time. When a code module is selected in the Project Explorer window the only associated property is the name of the code module. However when a userform or another object is selected then a list of properties will be displayed. Lists the design-time properties for selected objects and their current settings. You can change these properties at design time. When you select multiple controls, the Properties window contains a list of the properties common to all the selected controls.

WINDOW ELEMENTS:

Object Box

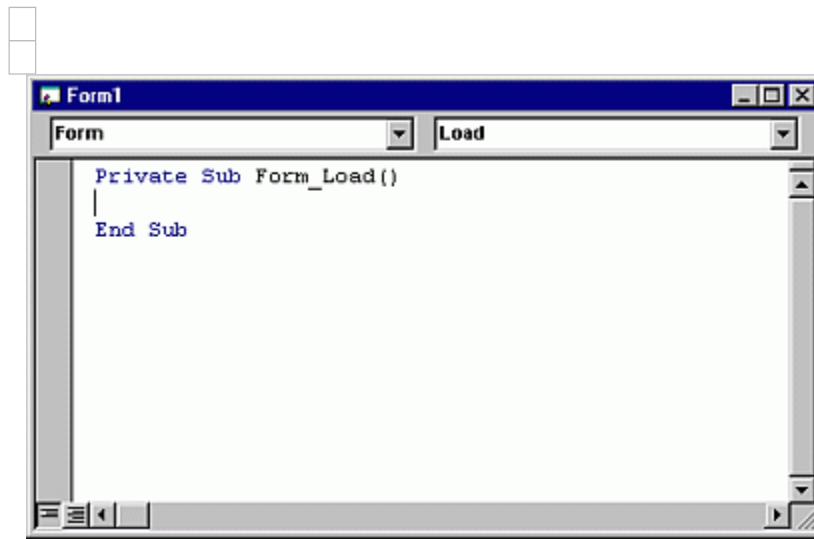
Lists the currently selected object. Only objects from the active form are visible. If you select multiple objects, the properties common to the objects and their settings, based on the first object selected, appear on the Properties List tabs.

Properties List Tabs :

Alphabetic Tab - Alphabetically lists all properties for the selected object that can be changed at design time, as well as their current settings. You can change the property setting by selecting the property name and typing or selecting the new setting.

Categorized Tab - Lists all properties for the selected object by category. For example, BackColor, Caption, and ForeColor are in the Appearance category. You can collapse the list so

that you see the categories or you can expand a category to see the properties. When you expand or collapse the list, you see a plus (+) icon or minus (-) icon to the left of the category name.



Use the **Code** window to write, display, and edit Visual Basic code. You can open as many **Code** windows as you have modules, so you can easily view the code in different forms or modules, and copy and paste between them.

You can open a **Code** window from:

- The **Project** window, by selecting a form or module, and choosing the **View Code** button.
- A **UserForm** window, by double-clicking a control or form, choosing **Code** from the **View** menu, or pressing F7.

You can drag selected text to:

- A different location in the current **Code** window.
- Another **Code** window.
- The **Immediate** and **Watch** windows.
- The **Recycle Bin**.

Window Elements

Object Box

Displays the name of the selected object. Click the arrow to the right of the list box to display a list of all objects associated with the form.

Procedures/Events Box

Lists all the events recognized by Visual Basic for a form or control displayed in the Object box. When you select an event, the event procedure associated with that event name is displayed in the **Code** window.

All the procedures in a module appear in a single, scrollable list that is sorted alphabetically by name. Selecting a procedure using the drop down list boxes at the top of the **Code** window moves the cursor to the first line of code in the procedure you select.

Split Bar

Dragging this bar down, splits the **Code** window into two horizontal panes, each of which scrolls separately. You can then view different parts of your code at the same time. The information that appears in the Object box and Procedures/Events box applies to the code in the pane that has the **focus**. Dragging the bar to the top or the bottom of the window or double-clicking the bar closes a pane.

Margin Indicator Bar

A gray area on the left side of the **Code** window where margin indicators are displayed.

Procedure View Icon

Displays the selected procedure. Only one procedure at a time is displayed in the **Code** window.

Full Module View Icon

Displays the entire code in the module.

Variable Declaration:

A variable is nothing but a name given to a storage area that our programs can manipulate. Each variable in VB.Net has a specific type, which determines the size and layout of the variable's memory; the range of values that can be stored within that memory; and the set of operations that can be applied to the variable.

Type	Example
Integral types	SByte, Byte, Short, UShort, Integer, UInteger, Long, ULong and Char
Floating point types	Single and Double
Decimal types	Decimal
Boolean types	True or False values, as assigned
Date types	Date

VB.Net also allows defining other value types of variable like **Enum** and reference types of variables like **Class**. We will discuss date types and Classes in subsequent chapters.

Variable Declaration in VB.Net

The **Dim** statement is used for variable declaration and storage allocation for one or more variables. The Dim statement is used at module, class, structure, procedure or block level.

Syntax for variable declaration in VB.Net is:


```
[ <attributelist> ] [ accessmodifier ] [[ Shared ] [ Shadows ] | [ Static ] ]  
[ ReadOnly ] Dim [ WithEvents ] variablelist
```

Where,

- **attributelist** is a list of attributes that apply to the variable. Optional.
- **accessmodifier** defines the access levels of the variables, it has values as - Public, Protected, Friend, Protected Friend and Private. Optional.
- **Shared** declares a shared variable, which is not associated with any specific instance of a class or structure, rather available to all the instances of the class or structure. Optional.
- **Shadows** indicate that the variable re-declares and hides an identically named element, or set of overloaded elements, in a base class. Optional.
- **Static** indicates that the variable will retain its value, even when the after termination of the procedure in which it is declared. Optional.
- **ReadOnly** means the variable can be read, but not written. Optional.
- **WithEvents** specifies that the variable is used to respond to events raised by the instance assigned to the variable. Optional.
- **Variablelist** provides the list of variables declared.

Each variable in the variable list has the following syntax and parts:

```
variablename[ ( [ boundslist ] ) ] [ As [ New ] datatype ] [ = initializer ]
```

Where,

- **variablename**: is the name of the variable
- **boundslist**: optional. It provides list of bounds of each dimension of an array variable.
- **New**: optional. It creates a new instance of the class when the Dim statement runs.
- **datatype**: Required if Option Strict is On. It specifies the data type of the variable.
- **initializer**: Optional if New is not specified. Expression that is evaluated and assigned to the variable when it is created.

Some valid variable declarations along with their definition are shown here:

```
Dim StudentID As Integer
```

```
Dim StudentName As String
```

```
Dim Salary As Double
```

```
Dim count1, count2 As Integer
Dim status As Boolean
Dim exitButton As New System.Windows.Forms.Button
Dim lastTime, nextTime As Date
```

Variable Initialization in VB

Variables are initialized (assigned a value) with an equal sign followed by a constant expression. The general form of initialization is:

```
variable_name = value;
```

for example,

```
Dim pi As Double
pi = 3.14159
```

You can initialize a variable at the time of declaration as follows:

```
Dim StudentID As Integer = 100
Dim StudentName As String = "Bill Smith"
```

Example

Try the following example which makes use of various types of variables:

```
Module variablesNdatatypes
    Sub Main()
        Dim a As Short
        Dim b As Integer
        Dim c As Double
        a = 10
        b = 20
        c = a + b
        Console.WriteLine("a = {0}, b = {1}, c = {2}", a, b, c)
        Console.ReadLine()
    End Sub
End Sub
```

End Module

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

```
a = 10, b = 20, c = 30
```

Accepting Values from User

The Console class in the System namespace provides a function **ReadLine** for accepting input from the user and store it into a variable. For example,

```
Dim message As String
```

```
message = Console.ReadLine
```

The following example demonstrates it:

```
Module variablesNdataypes
```

```
Sub Main()
```

```
Dim message As String
```

```
Console.Write("Enter message: ")
```

```
message = Console.ReadLine
```

```
Console.WriteLine()
```

```
Console.WriteLine("Your Message: {0}", message)
```

```
Console.ReadLine()
```

```
End Sub
```

```
End Module
```

When the above code is compiled and executed, it produces the following result (assume the user inputs Hello World):

```
Enter message: Hello World
```

```
Your Message: Hello World
```

Lvalues and Rvalues

There are two kinds of expressions:

- **lvalue** : An expression that is an lvalue may appear as either the left-hand or right-hand side of an assignment.
- **rvalue** : An expression that is an rvalue may appear on the right- but not left-hand side of an assignment.

Variables are lvalues and so may appear on the left-hand side of an assignment. Numeric literals are rvalues and so may not be assigned and can not appear on the left-hand side. Following is a valid statement:

```
Dim g As Integer = 20
```

But following is not a valid statement and would generate compile-time error:

```
20 = g
```

UNIT-II

Scope of Variables:

The *scope* of a declared element is the set of all code that can refer to it without qualifying its name or making it available . An element can have scope at one of the following levels:

Level	Description
Block scope	Available only within the code block in which it is declared
Procedure scope	Available to all code within the procedure in which it is declared
Module scope	Available to all code within the module, class, or structure in which it is declared
Namespace scope	Available to all code in the namespace in which it is declared

These levels of scope progress from the narrowest (block) to the widest (namespace), where *narrowest scope* means the smallest set of code that can refer to the element without qualification. For more information, see "Levels of Scope" on this page.

Specifying Scope and Defining Variables

You specify the scope of an element when you declare it. The scope can depend on the following factors:

P.INDHU MCA.,M.Phil., ASST. PROFESSOR,ANNAI WOMEN'S COLLEGE,KARUR.

- The region (block, procedure, module, class, or structure) in which you declare the element
- The namespace containing the element's declaration
- The access level you declare for the element

Use care when you define variables with the same name but different scope, because doing so can lead to unexpected results.

Levels of Scope

A programming element is available throughout the region in which you declare it. All code in the same region can refer to the element without qualifying its name.

Block Scope

A block is a set of statements enclosed within initiating and terminating declaration statements, such as the following:

- Do and Loop
- For [Each] and Next
- If and End If
- Select and End Select
- SyncLock and End SyncLock
- Try and End Try
- While and End While
- With and End With

If you declare a variable within a block, you can use it only within that block. In the following example, the scope of the integer variable `cube` is the block between `If` and `End If`, and you can no longer refer to `cube` when execution passes out of the block.

```
If n < 1291 Then
    Dim cube As Integer
    cube = n ^ 3
End If
```

Procedure Scope:

An element declared within a procedure is not available outside that procedure. Only the procedure that contains the declaration can use it. Variables at this level are also known as *local variables*. You declare them with the Dim Statement, with or without the Static keyword.

Procedure and block scope are closely related. If you declare a variable inside a procedure but outside any block within that procedure, you can think of the variable as having block scope, where the block is the entire procedure.

Module Scope

For convenience, the single term *module level* applies equally to modules, classes, and structures. You can declare elements at this level by placing the declaration statement outside of any procedure or block but within the module, class, or structure.

When you make a declaration at the module level, the access level you choose determines the scope. The namespace that contains the module, class, or structure also affects the scope.

Elements for which you declare Private access level are available to every procedure in that module, but not to any code in a different module. The Dim statement at module level defaults to Private if you do not use any access level keywords. However, you can make the scope and access level more obvious by using the Private keyword in the Dim statement.

Minimizing Scope

In general, when declaring any variable or constant, it is good programming practice to make the scope as narrow as possible (block scope is the narrowest). This helps conserve memory and minimizes the chances of your code erroneously referring to the wrong variable. Similarly, you should declare a variable to be Static only when it is necessary to preserve its value between procedure calls.

Constant:

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

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

The constants are treated just like regular variables except that their values cannot be modified after their definition.

An **enumeration** is a set of named integer constants.

Declaring Constants

In VB constants are declared using the **Const** statement. The Const statement is used at module, class, structure, procedure, or block level for use in place of literal values.

The syntax for the Const statement is:

```
[ < attributelist > ] [ accessmodifier ] [ Shadows ]
```

```
Const constantlist
```

Where,

- **attributelist**: specifies the list of attributes applied to the constants; you can provide multiple attributes separated by commas. Optional.
- **accessmodifier**: specifies which code can access these constants. Optional. Values can be either of the: Public, Protected, Friend, Protected Friend, or Private.
- **Shadows**: this makes the constant hide a programming element of identical name in a base class. Optional.
- **Constantlist**: gives the list of names of constants declared. Required.

Where, each constant name has the following syntax and parts:

```
constantname [ As datatype ] = initializer
```

- **constantname**: specifies the name of the constant
- **datatype**: specifies the data type of the constant
- **initializer**: specifies the value assigned to the constant

For example,

```
'The following statements declare constants.'
```

```
Const maxval As Long = 4999
```

```
Public Const message As String = "HELLO"
```

```
Private Const piValue As Double = 3.1415
```

Example

The following example demonstrates declaration and use of a constant value:

```
Module constantsNenum
```

P.INDHU MCA.,M.Phil., ASST. PROFESSOR,ANNAI WOMEN'S COLLEGE,KARUR.

```

Sub Main()
    Const PI = 3.14149
    Dim radius, area As Single
    radius = 7
    area = PI * radius * radius
    Console.WriteLine("Area = " & Str(area))
    Console.ReadKey()
End Sub
End Module

```

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

```
Area = 153.933
```

Print and Display Constants in VB

VB provides the following print and display constants:

Constant	Description
vbCrLf	Carriage return/linefeed character combination.
vbCr	Carriage return character.
vbLf	Linefeed character.
vbNewLine	Newline character.
vbNullChar	Null character.
vbNullString	Not the same as a zero-length string (""); used for calling external procedures.
vbObjectError	Error number. User-defined error numbers should be greater than this value. For example:

	Err.Raise(Number) = vbObjectError + 1000
vbTab	Tab character.
vbBack	Backspace character.

Declaring Enumerations

An enumerated type is declared using the **Enum** statement. The Enum statement declares an enumeration and defines the values of its members. The Enum statement can be used at the module, class, structure, procedure, or block level.

The syntax for the Enum statement is as follows:

```
[ < attributelist > ] [ accessmodifier ] [ Shadows ]
Enum enumerationname [ As datatype ]
    memberlist
End Enum
```

Where,

- **attributelist**: refers to the list of attributes applied to the variable. Optional.
- **accessmodifier**: specifies which code can access these enumerations. Optional. Values can be either of the: Public, Protected, Friend or Private.
- **Shadows**: this makes the enumeration hide a programming element of identical name in a base class. Optional.
- **enumerationname**: name of the enumeration. Required
- **datatype**: specifies the data type of the enumeration and all its members.
- **memberlist**: specifies the list of member constants being declared in this statement. Required.

Each member in the memberlist has the following syntax and parts:

```
[< attribute list>] member name [ = initializer ]
```

Where,

- **name**: specifies the name of the member. Required.
- **initializer**: value assigned to the enumeration member. Optional.

For example,

P.INDHU MCA.,M.Phil., ASST. PROFESSOR,ANNAI WOMEN'S COLLEGE,KARUR.

```
Enum Colors
```

```
red = 1
```

```
orange = 2
```

```
yellow = 3
```

```
green = 4
```

```
azure = 5
```

```
blue = 6
```

```
violet = 7
```

```
End Enum
```

Example

The following example demonstrates declaration and use of the Enum variable *Colors*:

```
Module constantsNenum
```

```
Enum Colors
```

```
red = 1
```

```
orange = 2
```

```
yellow = 3
```

```
green = 4
```

```
azure = 5
```

```
blue = 6
```

```
violet = 7
```

```
End Enum
```

```
Sub Main()
```

```
Console.WriteLine("The Color Red is : " & Colors.red)
```

```
Console.WriteLine("The Color Yellow is : " & Colors.yellow)
```

```
Console.WriteLine("The Color Blue is : " & Colors.blue)
```

```
Console.WriteLine("The Color Green is : " & Colors.green)
```

```
Console.ReadKey()
```

```
End Sub
```

```
End Module
```

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

```
The Color Red is: 1
```

```
The Color Yellow is: 3
```

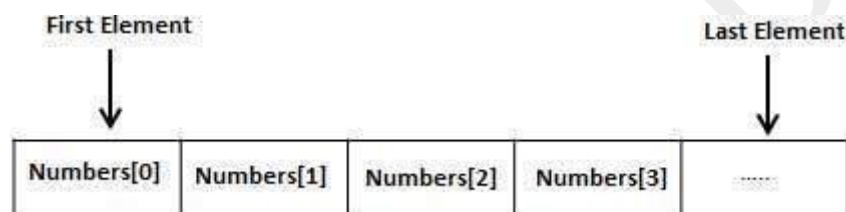
```
The Color Blue is: 6
```

```
The Color Green is: 4
```

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

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
```

You can also initialize the array elements while declaring the array. For example,

```
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}
```

P.INDHU MCA.,M.Phil., ASST. PROFESSOR,ANNAI WOMEN'S COLLEGE,KARUR.

The elements in an array can be stored and accessed by using the index of the array. The following program demonstrates this:

```
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
```

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

```
Element(0) = 100
Element(1) = 101
Element(2) = 102
Element(3) = 103
Element(4) = 104
Element(5) = 105
Element(6) = 106
Element(7) = 107
Element(8) = 108
Element(9) = 109
```

```
Element(10) = 110
```

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 for ReDim statement:

```
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
```

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

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

Multi-Dimensional Arrays

VB allows multidimensional arrays. Multidimensional arrays are also called rectangular arrays.

You can declare a 2-dimensional array of strings as:

```
Dim twoDStringArray(10, 20) As String
```

or, a 3-dimensional array of Integer variables:

```
Dim threeDIntArray(10, 10, 10) As Integer
```

The following program demonstrates creating and using a 2-dimensional array:

```
Module arrayApl
```

```
Sub Main()
```

```
    ' an array with 5 rows and 2 columns
```

```
    Dim a(.) As Integer = {{0, 0}, {1, 2}, {2, 4}, {3, 6}, {4, 8}}
```

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



```

' output each array element's value '
For i = 0 To 4
    For j = 0 To 1
        Console.WriteLine("a[{0},{1}] = {2}", i, j, a(i, j))
    Next j
Next i
Console.ReadKey()
End Sub
End Module

```

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

```

a[0,0]: 0
a[0,1]: 0
a[1,0]: 1
a[1,1]: 2
a[2,0]: 2
a[2,1]: 4
a[3,0]: 3
a[3,1]: 6
a[4,0]: 4
a[4,1]: 8

```

The Array Class

The Array class is the base class for all the arrays in VB.Net. It is defined in the System namespace. The Array class provides various properties and methods to work with arrays.

Properties of the Array Class

The following table provides some of the most commonly used **properties** of the **Array** class:

S.N	Property Name & Description
-----	-----------------------------

1	IsFixedSize Gets a value indicating whether the Array has a fixed size.
2	IsReadOnly Gets a value indicating whether the Array is read-only.
3	Length Gets a 32-bit integer that represents the total number of elements in all the dimensions of the Array.
4	LongLength Gets a 64-bit integer that represents the total number of elements in all the dimensions of the Array.
5	Rank Gets the rank (number of dimensions) of the Array.

Methods of the Array Class

The following table provides some of the most commonly used **methods** of the **Array** class:

S.N	Method Name & Description
1	Public Shared Sub Clear (array As Array, index As Integer, length As Integer) Sets a range of elements in the Array to zero, to false, or to null, depending on the element type.
2	Public Shared Sub Copy (sourceArray As Array, destinationArray As Array, length As Integer) Copies a range of elements from an Array starting at the first element and pastes them into another Array starting at the first element. The length is specified as a 32-bit integer.
3	Public Sub CopyTo (array As Array, index As Integer) Copies all the elements of the current one-dimensional Array to the specified one-dimensional Array starting at the specified

	destination Array index. The index is specified as a 32-bit integer.
4	Public Function GetLength (dimension As Integer) As Integer Gets a 32-bit integer that represents the number of elements in the specified dimension of the Array.
5	Public Function GetLongLength (dimension As Integer) As Long Gets a 64-bit integer that represents the number of elements in the specified dimension of the Array.
6	Public Function GetLowerBound (dimension As Integer) As Integer Gets the lower bound of the specified dimension in the Array.
7	Public Function GetType As Type Gets the Type of the current instance (Inherited from Object).
8	Public Function GetUpperBound (dimension As Integer) As Integer Gets the upper bound of the specified dimension in the Array.
9	Public Function GetValue (index As Integer) As Object Gets the value at the specified position in the one-dimensional Array. The index is specified as a 32-bit integer.
10	Public Shared Function IndexOf (array As Array,value As Object) As Integer Searches for the specified object and returns the index of the first occurrence within the entire one-dimensional Array.
11	Public Shared Sub Reverse (array As Array) Reverses the sequence of the elements in the entire one-dimensional Array.
12	Public Sub SetValue (value As Object, index As Integer) Sets a value to the element at the specified position in the one-

	dimensional Array. The index is specified as a 32-bit integer.
13	Public Shared Sub Sort (array As Array) Sorts the elements in an entire one-dimensional Array using the IComparable implementation of each element of the Array.
14	Public Overridable Function ToString As String Returns a string that represents the current object (Inherited from Object).

For complete list of Array class properties and methods, please consult Microsoft documentation.

Example:

The following program demonstrates use of some of the methods of the Array class:

```
Module arrayApl
  Sub Main()
    Dim list As Integer() = {34, 72, 13, 44, 25, 30, 10}
    Dim temp As Integer() = list
    Dim i As Integer
    Console.WriteLine("Original Array: ")
    For Each i In list
      Console.WriteLine("{0} ", i)
    Next i
    Console.WriteLine()
    ' reverse the array
    Array.Reverse(temp)
    Console.WriteLine("Reversed Array: ")
    For Each i In temp
      Console.WriteLine("{0} ", i)
    Next i
    Console.WriteLine()
  End Sub
End Module
```

```
'sort the array
Array.Sort(list)
Console.Write("Sorted Array: ")
For Each i In list
    Console.Write("{0} ", i)
Next i
Console.WriteLine()
Console.ReadKey()
End Sub
End Module
```

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

```
Original Array: 34 72 13 44 25 30 10
```

```
Reversed Array: 10 30 25 44 13 72 34
```

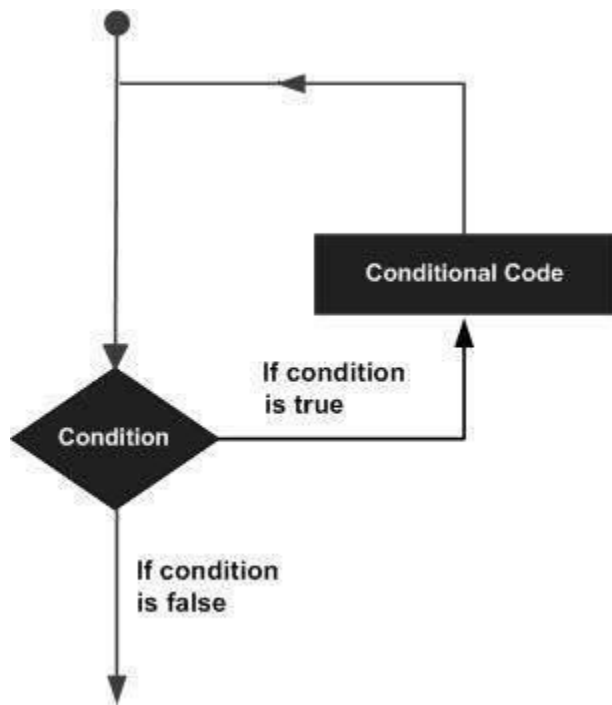
```
Sorted Array: 10 13 25 30 34 44 72
```

Loops in Visual Basic:

There may be a situation when you need to execute a block of code several number of times. In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on.

Programming languages provide various control structures that allow for more complicated execution paths.

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:



VB provides following types of loops to handle looping requirements. Click the following links to check their details.

Loop Type	Description
<u>Do Loop</u>	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.
<u>For...Next</u>	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.
<u>For Each...Next</u>	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.
<u>While... End While</u>	It executes a series of statements as long as a given condition is True.

<u>With... End With</u>	It is not exactly a looping construct. It executes a series of statements that repeatedly refer to a single object or structure.
<u>Nested Loops</u>	You can use one or more loops inside any another While, For or Do loop.

An object is a type of user interface element you create on a Visual Basic form by using a toolbox control. In fact, in Visual Basic, the form itself is an object. Every Visual Basic control consists of three important elements:

- **Properties** which describe the object,
- **Methods** cause an object to do something and
- **Events** are what happens when an object does something.

Control Properties

All the Visual Basic Objects can be moved, resized or customized by setting their properties. A property is a value or characteristic held by a Visual Basic object, such as Caption or Fore Color.

Properties can be set at design time by using the Properties window or at run time by using statements in the program code.

Object. Property = Value

Where

- **Object** is the name of the object you're customizing.
- **Property** is the characteristic you want to change.
- **Value** is the new property setting.

For example,

Form1.Caption = "Hello"

You can set any of the form properties using Properties Window. Most of the properties can be set or read during application execution. You can refer to Microsoft documentation for a complete list of properties associated with different controls and restrictions applied to them.

Control Methods

A method is a procedure created as a member of a class and they cause an object to do something. Methods are used to access or manipulate the characteristics of an object or a variable. There are mainly two categories of methods you will use in your classes:

- If you are using a control such as one of those provided by the Toolbox, you can call any of its public methods. The requirements of such a method depend on the class being used.
- If none of the existing methods can perform your desired task, you can add a method to a class.

For example, the *MessageBox* control has a method named *Show*, which is called in the code snippet below:

```
Public Class Form1
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
        Handles Button1.Click
            MessageBox.Show("Hello, World")
    End Sub
End Class
```

Control Events

An event is a signal that informs an application that something important has occurred. For example, when a user clicks a control on a form, the form can raise a **Click** event and call a procedure that handles the event. There are various types of events associated with a Form like click, double click, close, load, resize, etc.

Following is the default structure of a form **Load** event handler subroutine. You can see this code by double clicking the code which will give you a complete list of the all events associated with Form control:

```
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    'event handler code goes here
End Sub
```

Here, **Handles MyBase.Load** indicates that **Form1_Load()** subroutine handles **Load** event. Similar way, you can check stub code for click, double click. If you want to initialize some variables like properties, etc., then you will keep such code inside **Form1_Load()** subroutine. Here, important point to note is the name of the event handler, which is by default **Form1_Load**, but you can change this name based on your naming convention you use in your application programming.

UNIT - III

Using Visual Basic's Standard Controls

1. Using the Text Box Control

The text box control is used to display information entered by the user at run time, or assigned to the Text property of the control at design or run time.

The text box control:



In general, the text box control should be used for editable text, although you can make it read-only by setting its Locked property to True. Text boxes also allow you to display multiple lines, to wrap text to the size of the control, and to add basic formatting.

The Text Property:

Text entered into the text box control is contained in the Text property. By default, you can enter up to 2048 characters in a text box. If you set the MultiLine property of the control to True, you can enter up to 32K of text.

Formatting Text:

When text exceeds the boundaries of the control, you can allow the control to automatically wrap text by setting the MultiLine property to True and add scroll bars by setting the ScrollBars property to add either a horizontal or vertical scroll bar, or both. Automatic text wrapping will be unavailable, however, if you add a horizontal scroll bar because the horizontal edit area is increased by the presence of the scroll bar.

When the MultiLine property is set to True, you can also adjust the alignment of the text to either Left Justify, Center, or Right Justify. The text is left-justified by default. If the MultiLine property is False, setting the Alignment property has no effect.

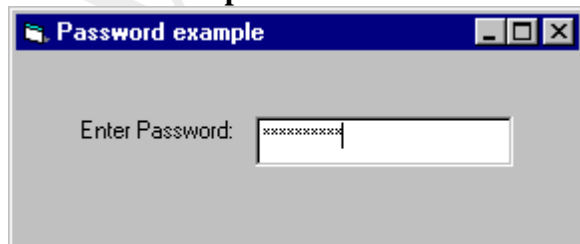
Selecting Text:

You can control the insertion point and selection behavior in a text box with the SelStart, SelLength and SelText properties.

Creating a Password Text Box:

A password box is a text box that allows a user to type in his or her password while displaying placeholder characters, such as asterisks. Visual Basic provides two text box properties, PasswordChar and MaxLength, which make it easy to create a password text box. PasswordChar specifies the character displayed in the text box. For example, if you want asterisks displayed in the password box, you specify * for the PasswordChar property in the Properties window. Regardless of what character a user types in the text box, an asterisk is displayed, as shown in Figure 7.45.

Password example



With `MaxLength`, you determine how many characters can be typed in the text box. After `MaxLength` is exceeded, the system emits a beep and the text box does not accept any further characters.

Canceling Keystrokes in a Text Box:

You can use the `KeyPress` event to restrict or transform characters as they are typed. The `KeyPress` event uses one argument, *keyascii*. This argument is an integer that represents the numeric (ASCII) equivalent of the character typed in the text box.

The next example demonstrates how to cancel keystrokes as they are typed. If the character typed is not within the specified range, the procedure cancels it by setting `KeyAscii` to 0. The text box for this example is named `txtEnterNums`, and the procedure prevents the text box from receiving any characters other than digits. Compare `KeyAscii` directly to the numeric (`Asc`) values of various characters.

```
Private Sub txtEnterNums_KeyPress (KeyAscii As Integer)
    If KeyAscii < Asc("0") Or KeyAscii > Asc("9") Then
        KeyAscii = 0 ' Cancel the character.
        Beep        ' Sound error signal.
    End If
End Sub
```

Creating a Read-Only Text Box

You can use the `Locked` property to prevent users from editing text box contents. Set the `Locked` property to `True` to allow users to scroll and highlight text in a text box without allowing changes. With the `Locked` property set to `True`, a `Copy` command will work in a text box, but `Cut` and `Paste` commands will not. The `Locked` property only affects *user interaction* at run time. You can still change text box contents *programmatically* at run time by changing the `Text` property of the text box.

Printing Quotation Marks in a String:

Sometimes quotation marks (" ") appear in a string of text.

She said, "You deserve a treat!"

Because strings assigned to a variable or property are surrounded by quotation marks (" "), you must insert an additional set of quotation marks for each set to display in a string. Visual Basic interprets two quotation marks in a row as an embedded quotation mark.

For example, to create the preceding string, use the following code:

```
Text1.Text = "She said, ""You deserve a treat!"" "
```

To achieve the same effect, you can use the ASCII character (34) for a quotation mark:

```
Text1.Text = "She said, " & Chr(34) + "You deserve a treat!" & Chr(34)
```

2. Using the Command Button Control

The command button control is used to begin, interrupt, or end a process. When clicked, it invokes a command that has been written into its `Click` event procedure.

The command button control



Most Visual Basic applications have command buttons that allow the user to simply click them to perform actions. When the user chooses the button, it not only carries out the appropriate

action, it also looks as if it's being pushed in and released and is therefore sometimes referred to as a push button.

Adding a Command Button to a Form

You will likely use one or more command buttons in your application. To add command buttons to a form, draw them on as you would any other control. Command buttons can be sized with the mouse or by setting their Height and Width properties.

Setting the Caption:

To change the text displayed on the command button, use the Caption property. At design time, you can set this property by selecting it from the control's Properties window. When you set the Caption property at design time, the button text will be updated dynamically.

You can set the Caption property up to 255 total characters. If your caption exceeds the width of the command button, it will wrap to the next line. However, it will be clipped if the control cannot accommodate its overall height.

You can change the font displayed on the command button by setting its Font property.

Creating Keyboard Shortcuts:

You can use the Caption property to create access key shortcuts for your command buttons by adding an ampersand (&) before the letter you want to use as the access key. For example, to create an access key for the caption "Print" you add an ampersand before the letter "P": "&Print". At run time, the letter "P" will be underlined and the user can select the command button by simultaneously pressing ALT+P.

Specifying the Default and Cancel Properties:

On each form, you can select a command button to be the default command button — that is, whenever the user presses the ENTER key the command button is clicked regardless of which other control on the form has the focus. To specify a command button as default set the Default property to True.

You can also specify a default cancel button. When the Cancel property of a command button is set to True, it will be clicked whenever the user presses the ESC key, regardless of which other control on the form has the focus.

Selecting the Command Button:

A command button can be selected at run time by using the mouse or keyboard in the following ways:

- Use a mouse to click the button.
- Move the focus to the button by pressing the TAB key, and then choose the button by pressing the SPACEBAR or ENTER.
- Press an access key (ALT+ the underlined letter) for a command button.
- If the command button is the *default command button* for the form, pressing ENTER chooses the button, even if you change the focus to a different control.
- If the command button is the *default Cancel button* for the form, then pressing ESC chooses the button, even if you change the focus to another control.

The Value Property:

Whenever the command button is selected, its Value property is set to True and the Click event is triggered. False (default) indicates the button isn't chosen. You can use the Value property in code to trigger the command button's Click event. For example:

```
cmdClose.Value = True
```

The Click Event

When clicked, the command button's Click event is triggered and the code you've written in the Click event procedure is invoked.

Clicking a command button control also generates the MouseDown and MouseUp events. If you intend to attach event procedures for these related events, be sure that their actions don't conflict. The order in which these three events occur varies from control to control. In the command button control, these events occur in this order: MouseDown, Click, MouseUp.



DriveListBox, DirListBox and FileListBox Controls

To work with the file system, the DriveListBox, DirListBox and FileListBox are used together that will access the files stored in the secondary memory of your computer. The DriveListBox shows all the drives on your computer, the DirListBox displays all the sub-directories of a given directory, and the FileListBox shows the files in a particular directory.

Most of the properties, events and methods are same as the ListBox and ComboBox controls. So the common things have not been explained in this section.

Selected Items

The Drive, Path and FileName properties return the selected item in the DriveListBox, DirListBox and FileListBox respectively. The Drive property returns the selected disk drive from

your computer.

Setting the Drive

Example:

```
Private Sub Form_Load()  
    Drive1.Drive = "G"  
End Sub
```

Output:



Working with the three list boxes together

When the user selects a drive, this change should be reflected in DirListBox and when the user selects a directory, this change should be reflected in the FileListBox. Write the following code for that.

Example:

```
Private Sub Dir1_Change()  
    File1.Path = Dir1.Path  
End Sub
```

```
Private Sub Drive1_Change()  
    Dir1.Path = Drive1.Drive  
End Sub
```

Showing the selected file name

To show only the selected file name, write the following code.

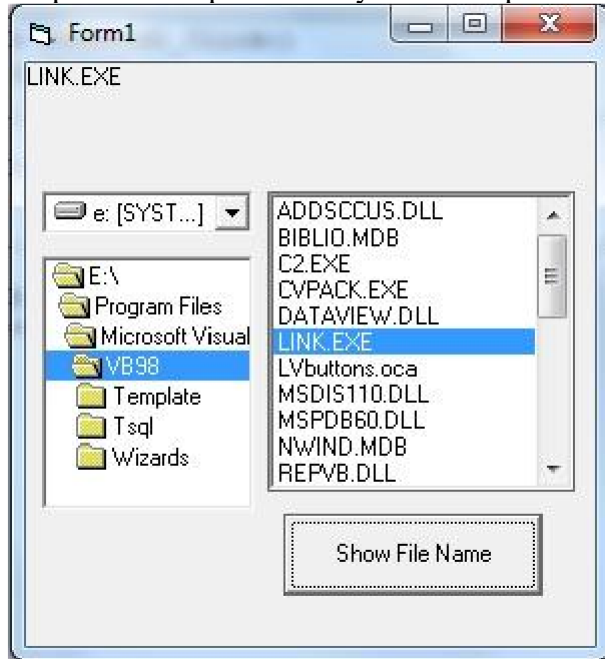
Example: This program will show only the name of the selected file.

```
Private Sub cmdShowFileName_Click()  
    Print File1.FileName  
End Sub
```

```
Private Sub Dir1_Change()  
    File1.Path = Dir1.Path  
End Sub
```

```
Private Sub Drive1_Change()
    Dir1.Path = Drive1.Drive
End Sub
```

Output: The output will vary from computer to computer.



The Pattern property of FileListBox

The *Pattern* property sets which files to be shown in the list. The default value is *.* (all files). You can specify which types of files to be shown. To enter multiple specifications, use semicolon as a separator. You can set the *Pattern* in run-time also.

Example:

```
File1.Pattern = "*.bmp;*.ico;*.wmf;*.gif;*.jpg"
```

Showing the complete file path with file name

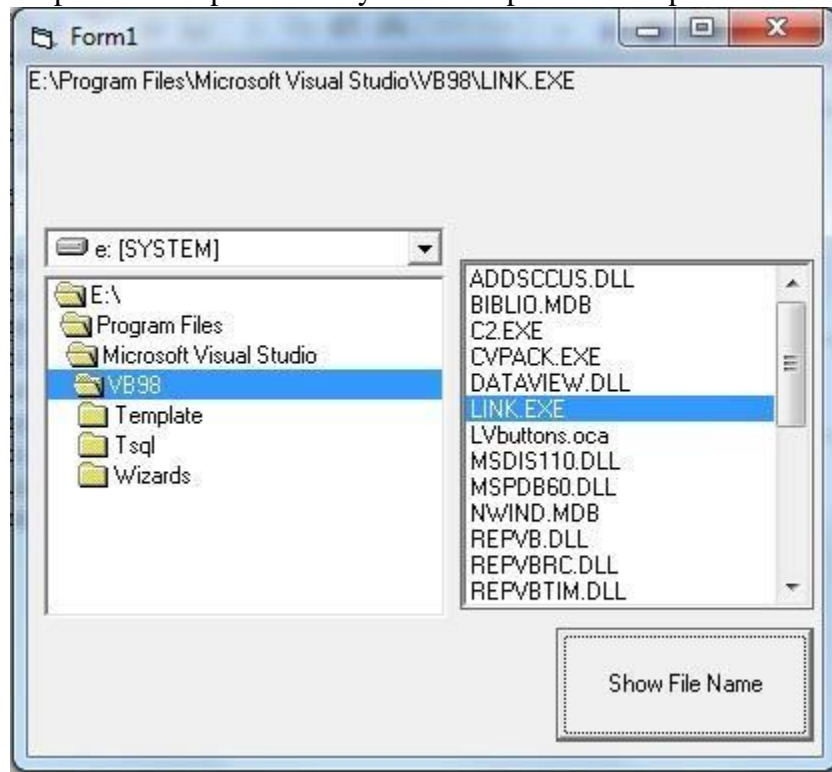
Example:

```
Private Sub cmdShowFileName_Click()
    f = File1.Path
    If Right(f, 1) <> "\" Then
        f = File1.Path + "\"
    End If
    Print f + File1.FileName
End Sub
```

```
Private Sub Drive1_Change()  
    Dir1.Path = Drive1.Drive  
End Sub
```

```
Private Sub Dir1_Change()  
    File1.Path = Dir1.Path  
End Sub
```

Output: The output will vary from computer to computer.



Three of the controls on the Toolbox let you access the computer's file system. They are DriveListBox, DirListBox and FileListBox controls .

The Drive ListBox is used to display a list of drives available in your computer. When you place this control into the form and run the program, you will be able to select different drive from your computer.

The DirListBox is Directory ListBox is used to display the list of directories or folder in a selected drive. When you place this control into the form and run the program, you will be able to select different directories from a selected drive in your computer.

Using VB File System Controls:

VB provides three native toolbox controls for working with the file system:

the **DriveListBox**, **DirListBox**, and **FileListBox**. You can use these controls independently, or in concert with one another to navigate the file system.

The **DriveListBox** control is a specialized drop-down list that displays a list of all the valid drives on the user's system. The most important property of the DriveListBox is the **Drive** property, which is set when the user selects an entry from the drop-down list or when you assign a drive string (such as "C:") to the Drive property in code. You can also read the Drive property to see which drive has been selected.

To make a DirListBox display the directories of the currently selected drive, you would set the **Path** property of the DirListBox control to the **Drive** property of the DriveListBox control in the **Change** event of the DriveListBox, as in the following statement:

```
Dir1.Path = Drive1.Drive
```

The **DirListBox** control displays a hierarchical list of the user's disk directories and subdirectories and automatically reacts to mouse clicks to allow the user to navigate among them. To synchronize the path selected in the DirListBox with a FileListBox, assign the **Path** property of the DirListBox to the **Path** property of the FileListBox in the **Change** event of the DirListBox, as in the following statement:

```
File1.Path = Dir1.Path
```

The **FileListBox** control lists files in the directory specified by its Path property. You can display all the files in the current directory, or you can use the **Pattern** property to show only certain types of files.

Similar to the standard ListBox and ComboBox controls, you can reference the **List**, **ListCount**, and **ListIndex** properties to access items in a DriveListBox, DirListBox, or FileListBox control. In addition, the FileListBox has a **MultiSelect** property which may be set to allow multiple file selection.

Sample Program Overview

The sample program is a "Text File Viewer". The sample program uses the DriveListBox, DirListBox, and FileListBox to allow the user to navigate his or her file system. When the user selects a file that the program deems to be a "plain text" file, and that file is not "too large", the contents of that file is displayed in a multi-line, scrollable textbox.

In the screen-shot below, the user has navigated to the directory "C:\Program Files\CesarFTP" and selected the file "tests.txt" from that directory. The content of "tests.txt" is displayed in the multi-line textbox:

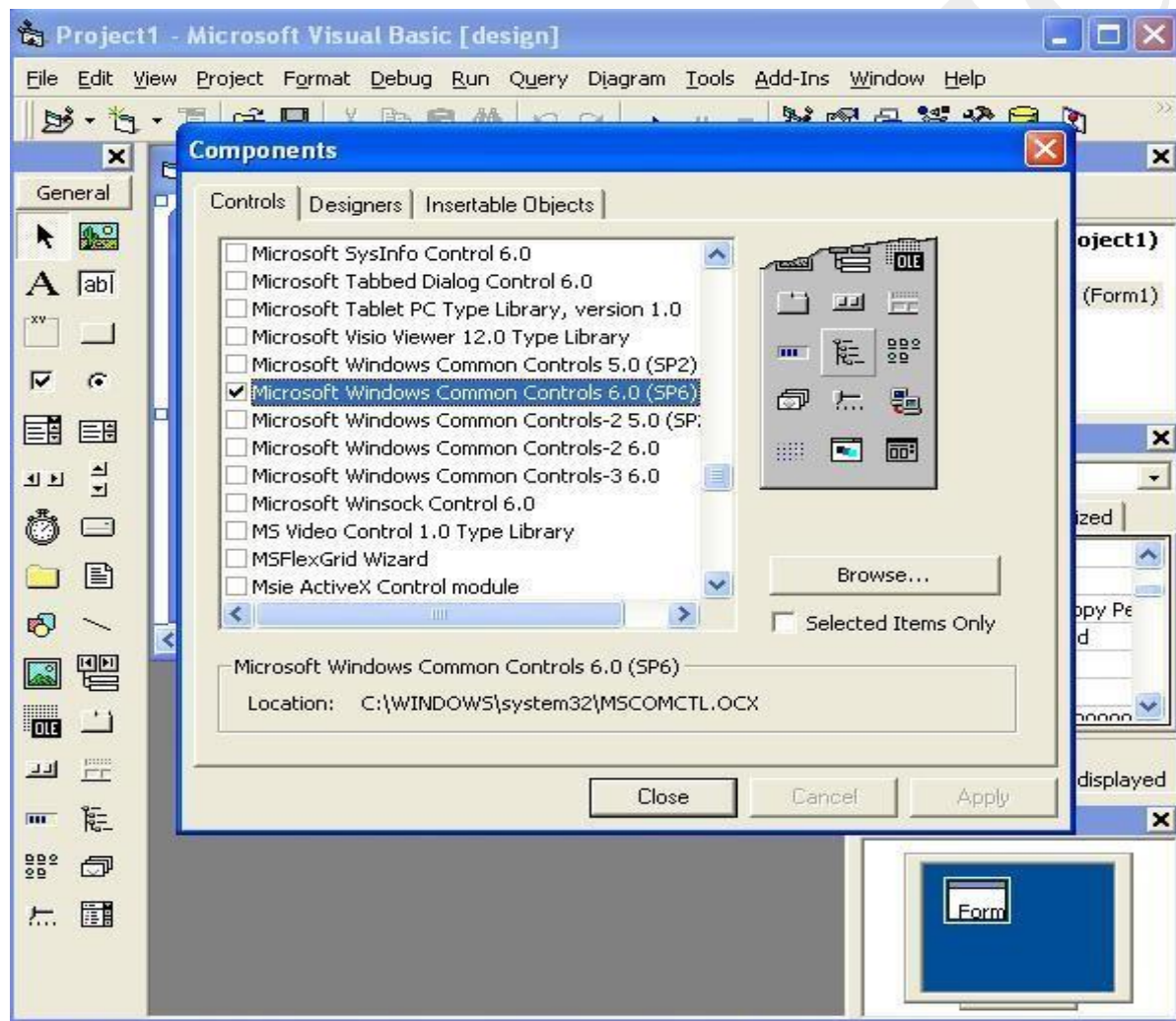
ActiveX Controls :

One of the most exciting feature of Visual Basic is the ActiveX. It is the technology developed by Microsoft that defines a communication standard between applications. The ActiveX control exist as separate files with a .ocx file name extension. These include controls that are available in all editions of Visual Basic and those that are available only in the Professional and Enterprise editions (such as Listview, Toolbar, Animation, and Tabbed Dialog).

Adding Built-in ActiveX Controls to the Toolbox:

1. From the **Project** Menu, choose **Components**.you can also right click on toolbox.A popup menu is open select **components**.
2. A component dialogbox is open .Select the Microsoft windows common controls 6.0(SP6).
3. Then click on apply button .you can see some other components like Listview,

Toolbar, Animation, and Tabbed Dialog show on your toolbox. Click on close button.Use this control as you want



To build an ActiveX control from scratch and compile it so that it is usable as a command button or a listbox is in other projects, you write the ActiveX Control and then compile it to an OCX file. The ocx file contains only the user controls that were part of the VB project you built it from and no extra stuff like forms or full programs

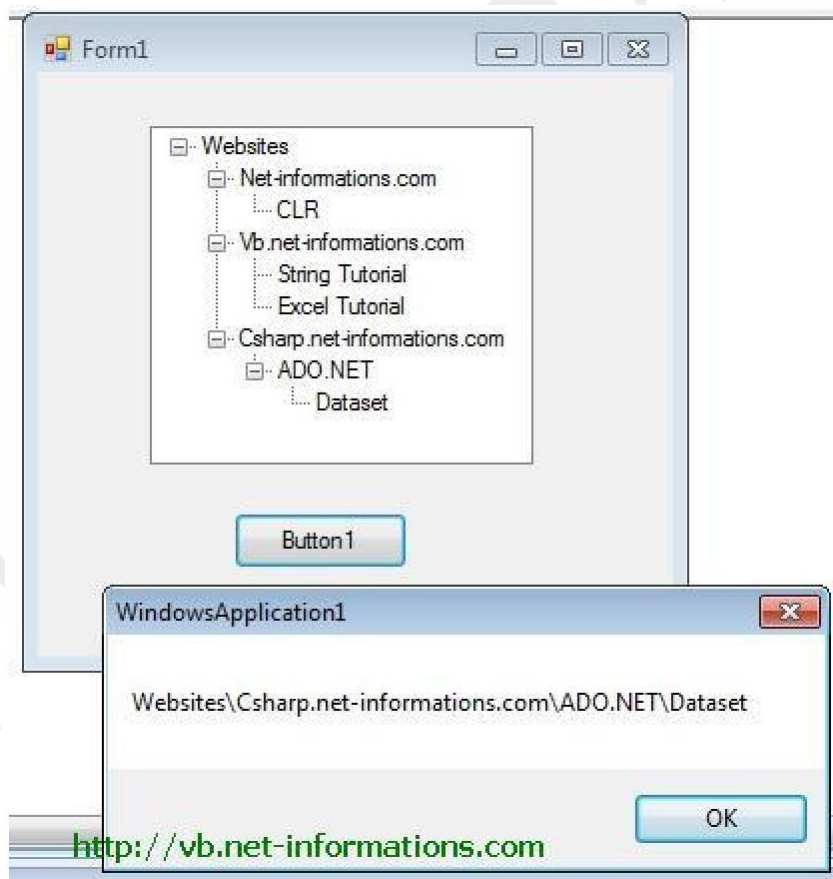
Your computer also contains many ActiveX controls that were installed by Excel and other programs, such as Calendar Control 12.0 and Windows Media Player.

Important: Not all ActiveX controls can be used directly on worksheets; some can be used only on Visual Basic for Applications (VBA) UserForms. If you try to add any one of these particular ActiveX controls to a worksheet, Excel displays the message "Cannot insert object."

However, ActiveX controls cannot be added to chart sheets from the user interface or to XLM macro sheets. You also cannot assign a macro to run directly from an ActiveX control the same way you can from a Form control.

Treeview Control

TreeView control is used to display hierarchical tree like information such as a directory hierarchy. The top level in a tree view are root nodes that can be expanded or collapsed if the nodes have child nodes.



The user can expand the TreeNode by clicking the plus sign (+) button, if one is displayed next to the TreeNode, or you can expand the TreeNode by calling the

TreeNode.Expand method. When a parent node is expanded, its child nodes are visible. You can also navigate through tree views with various properties: FirstNode, LastNode, NextNode, PrevNode, NextVisibleNode, PrevVisibleNode.

The fullpath method of treeview control provides the path from root node to the selected node.

TreeView1.SelectedNode.FullPath

Tree nodes can optionally display check boxes. To display the check boxes, set the CheckBoxes property of the TreeView to true.

TreeView1.CheckBoxes = True

The following Vb.Net program shows a simple demonstration of treeview control

The **TreeView** control display a hierarchical display node object. Each node consist of label ,optional bitmap.You use this control to display the information in hierarchical tree.

The TreeView control is designed to display data that is hierarchical in nature, such as organization trees, the entries in an index, the files and directories on a disk.

Uses of TreeView

- To create an organization tree that can be manipulated by the user.
- To create a tree that shows at least two or more levels of a database.

Tree Terms

Node: A node can be best thought of as a branch on our tree. Each node represents 1 item. A node can hold any number of sub nodes.

Sibling: A sibling is another node that exists on the same branch as the referring node.

Children: Children are sub-nodes of the current node. Sometimes called Child nodes.

Parent: A parent node is the node —upl from your current node. All nodes except the root node have a parent node.

Tree View Creation:

In a Windows application a tree view is primarily a control like any other. To use it in your application, you can click the TreeView button in the Toolbox and click a form or other control in your application. This is equivalent to programmatically declaring a variable of type **TreeView**, using the **new** operator to instantiate it and adding it to its container's list of controls through a call to the **Controls.Add()** method. Here is an example:

```
Imports System.Drawing
Imports System.Windows.Forms
```

Module Exercise

```
Public Class Starter  
    Inherits Form
```

```
    Private tvwCountries As TreeView
```

```
    Dim components As System.ComponentModel.Container
```

```
    Public Sub New()  
        InitializeComponent()  
    End Sub
```

```
    Public Sub InitializeComponent()
```

```
        Text = "Countries Statistics"  
        Size = New Size(242, 280)
```

```
        tvwCountries = New TreeView()
```

```
        tvwCountries.Location = New Point(12, 12)  
        tvwCountries.Width = 210  
        tvwCountries.Height = 230
```

```
        Controls.Add(tvwCountries)  
    End Sub
```

```
End Class
```

```
Function Main() As Integer
```

```
    Dim frmStart As Starter = New Starter
```

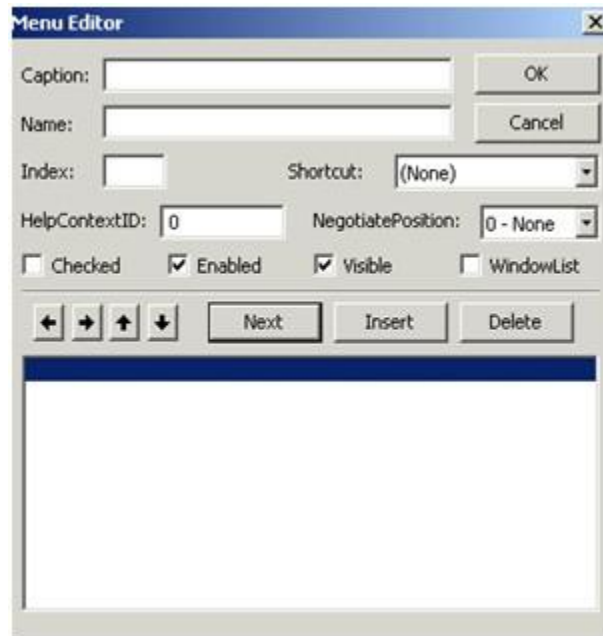
```
    Application.Run(frmStart)
```

```
    Return 0  
End Function
```

```
End Module
```

MENU EDITOR

Menus are features that are available in nearly all programs nowadays . Making menus is very easy in Visual Basic using the menu editor . To start the menu editor you go to **Tools | Menu Editor**. The menu editor now appears , here is what is displayed on screen.



Caption : this is the name that the user will see.

Name : this is the name that the programmer uses , we use the mnu prefix for all menu items(mnuFile).

Shortcut : this assigns a shortcut to a menu item this is a combination of keys which access a menu item for example Ctrl + c is commonly used for copy

Checked : this allows the programmer to place a check beside a menu item , this is unchecked by default.

Enabled : This specifies whether the menu item is accessible to the user , if this is checked the menu item is grayed out and inaccessible.

Visible: this determines whether the menu item is visible if this is not checked then the menu item will not appear at run time.

WindowList : determines whether the menu item applies to an MDI document (Word and Excel are examples of MDI applications).

HelpContextID : this matches a help description if you have any in your program.

Index : this specifies a menus index in a control array.

You will also have noticed the arrows these are used to manipulate manu items the up and down arrows move the menu items up and down the list and the left / right arrows are used to indent the

menu items.

MENU EDITOR

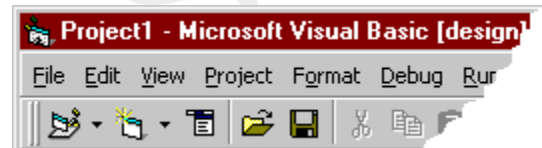
Preface

In other programming systems, you write code to generate menus in forms. If you don't write it directly, the programming system will write it for you. Not so in Visual Basic.

When you create a menu in Visual Basic, no sign of it will ever appear in your code. That's why the Menu Editor is your only way of creating, editing, and removing menus in your forms. If you don't know how to use it, you're pretty much doomed from the get-go.

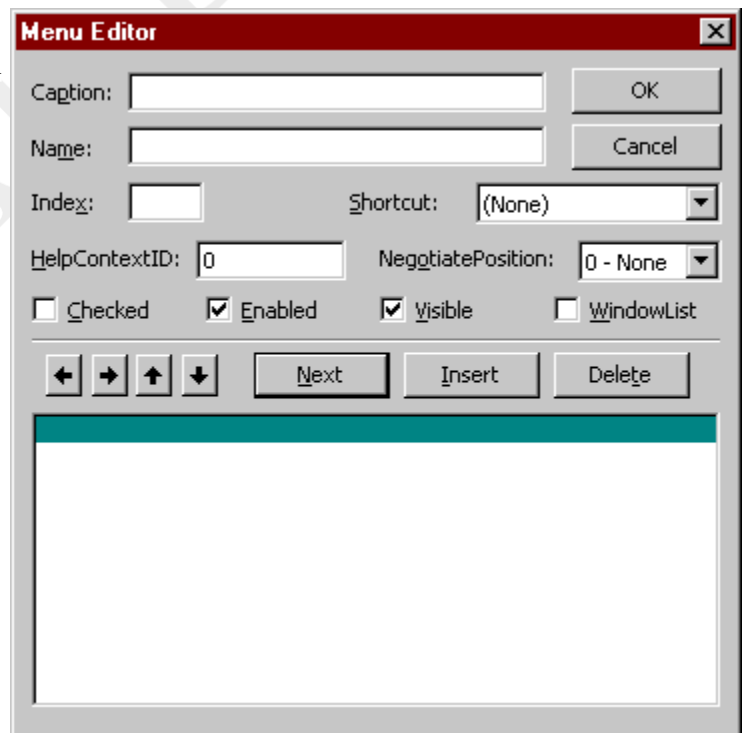
Getting Started

First of all, start Visual Basic and add a new project. When you've started it, go looking in the toolbar for the third icon on the left. When you press it, the Menu Editor should appear.



The Editor Window

The first thing you'll see is an empty list and a lot of text fields and drop-down lists. It can be quite confusing, and it's not obvious how you design the menus.



Creating Menu Items

Let's start with explaining the basic properties of a menu item:

Caption The text that appears in the menu bar

Name The name of the menu item, just like the name of a textbox or label

Index If the menu is part of a control array, this is the array index

Your basic menu item would look like the ones in Figure 1: The Caption follows basic naming in Windows, and the names have the standard mnu preface.

From this picture, you also see how the menus will look in the list. The principle of designing menus is rather backwards, considering all the other things you can do with Visual Basic:

Each menu item has its own line in the big list. To tell Visual Basic that one menu item should be a child of (that is, be sorted under) another menu item, you indent it. The two buttons with arrows pointing left and right serve that purpose -- unindent and indent (left and right).

The three dots in front of *New*, *Open*, and *Save* tell Visual Basic that these menu items are to be placed in the File menu.

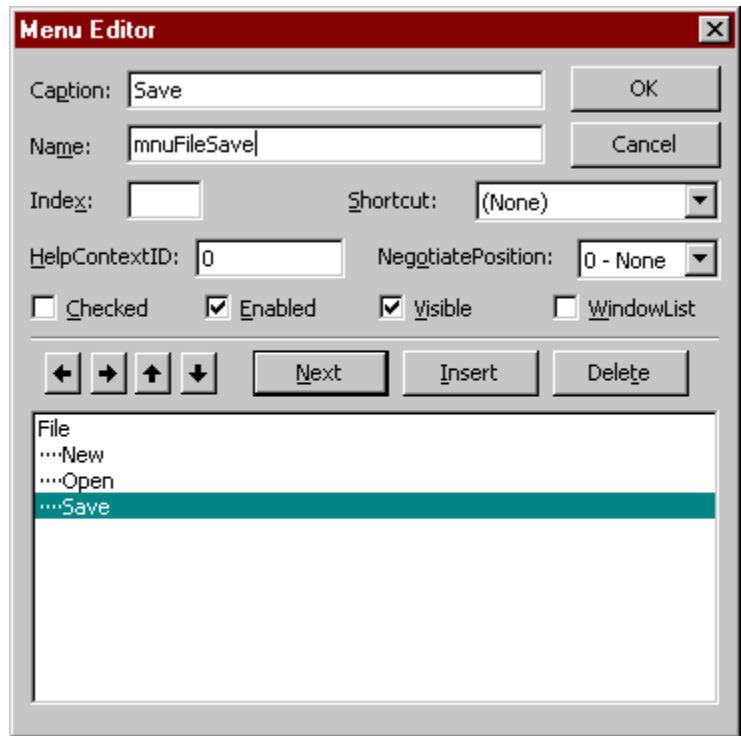


Figure 1: The menu editor -- adding menu items

When you're done adding items to the File menu, it's time to press the unindent button to go back to the "top" level. If you want a new menu, f. ex. an Edit menu, you would repeat the procedure of inserting lines and indenting them to the correct level. See Figure 2 for an example.

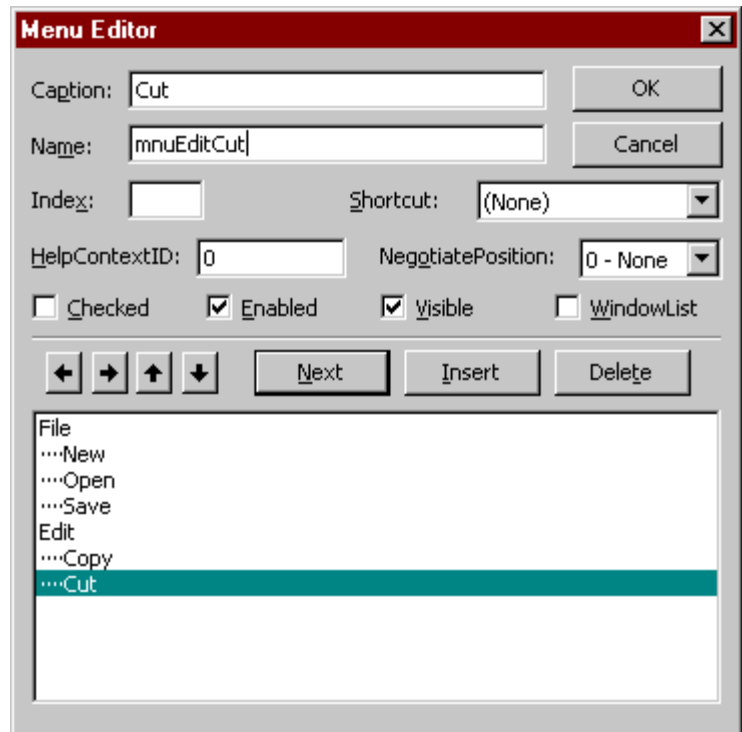


Figure 2: Adding several top-level menu items

Short-cuts and hot keys

In Visual Basic, the & in front of a character tells Visual Basic that Alt+[the character] will be a hot key for that item. The same applies to menus, except that you don't press Alt. Hot keys are only in effect when the menu has been pulled down and has focus.

Short-cuts are the key combinations you press to active a menu item when the menu does not have focus. The three most commonly used short-cuts are Ctrl-C, Ctrl-X, and Ctrl-V. These are typically bound to Copy, Cut, and Paste, but you can bind them to any of your menu items if you choose.

If you want to set the short-cut key for a menu item, you have the options listed in the Shortcut: list.

Other Properties of a Menu Item

As we said earlier, there are a lot of confusing items in the Menu Editor, and the help file doesn't do much to help. These are the other menu item properties you'll be using:

HelpContextId If you have made a help file for your application, you will know that a help context id is a unique identifier assigned to each "chapter" in the help file. This context id will let you assign one chapter to each menu item, so that the correct help text will be displayed when the user presses F1

NegotiatePosition 0 points to Microsoft for enlightening help texts. I still haven't found out which

	effect this property has on a menu
Checked	Menu items can be checked or unchecked (you know, the little check mark beside them). This property lets you check or uncheck them.
Enabled	If this is checked, the menu item will look normal and be clickable. If it isn't checked, the menu item is grayed out and isn't clickable.
Visible	Same as Enabled, but will hide the menu item instead of graying it out
WindowList	If this item is checked, all open forms will be added as children of this menu item. The active form will be checked, and if you click the name of a form, it will become the active form. Used in MDI applications.

Using the Common Dialog Control

The common dialog control provides a standard set of dialog boxes for operations such as opening and saving files, setting print options, and selecting colors and fonts. The control also has the ability to display Help by running the Windows Help engine.



The common dialog control

The common dialog control provides an interface between Visual Basic and the procedures in the Microsoft Windows dynamic-link library Commdlg.dll. To create a dialog box using this control, Commdlg.dll must be in your Microsoft Windows \System directory. You use the common dialog control in your application by adding it to a form and setting its properties. The dialog displayed by the control is determined by the methods of the control. At run time, a dialog box is displayed or the Help engine is executed when the appropriate method is invoked; at design time, the common dialog control is displayed as an icon on a form. This icon can't be sized.

The common dialog control allows you to display these commonly used dialog boxes:

- Open
- Save As
- Color
- Font
- Print

To use the common dialog control

1. If you haven't already done so, add the common dialog control to the toolbox by selecting **Components** from the **Project** menu. Locate and select the control in the **Controls** tabbed dialog, then click the **OK** button.
2. On the toolbox, click the **CommonDialog** control and draw it on a form.

When you draw a common dialog control on a form, it automatically resizes itself. Like the timer control, the common dialog control is invisible at run time.

3. At run time, use the appropriate method, as listed in the following table, to display the desired dialog.

Method	Dialog displayed
ShowOpen	Open
ShowSave	Save As
ShowColor	Color
ShowFont	Font
ShowPrinter	Print
ShowHelp	Invokes Windows Help

RichTextBox Control

The **RichTextBox** control allows the user to enter and edit text while also providing more advanced formatting features than the conventional **TextBox** control.

The **RichTextBox** control provides a number of properties you can use to apply formatting to any portion of text within the control. To change the formatting of text, it must first be selected. Only selected text can be assigned character and paragraph formatting. Using these properties, you can make text bold or italic, change the color, and create superscripts and

subscripts. You can also adjust paragraph formatting by setting both left and right indents, as well as hanging indents.

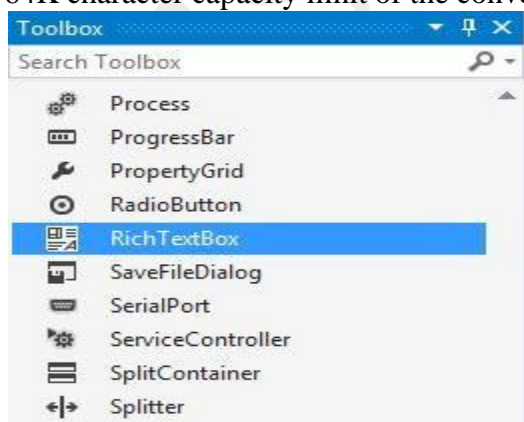
The **RichTextBox** control opens and saves files in both the RTF format and regular ASCII text format. You can use methods of the control (**LoadFile** and **SaveFile**) to directly read and write files, or use properties of the control such as **SelRTF** and **TextRTF** in conjunction with Visual Basic's file input/output statements.

The **RichTextBox** control supports object embedding by using the **OLEObjects** collection. Each object inserted into the control is represented by an **OLEObject** object. This allows you to create documents with the control that contain other documents or objects. For example, you can create a document that has an embedded Microsoft Excel spreadsheet or a Microsoft Word document or any other OLE object registered on your system. To insert objects into the **RichTextBox** control, you simply drag a file (from the Windows 95 Explorer for example), or a highlighted portion of a file used in another application (such as Microsoft Word), and drop the contents directly onto the control.

The **RichTextBox** control supports both clipboard and OLE drag/drop of OLE objects. When an object is pasted in from the clipboard, it is inserted at the current insertion point. When an object is dragged and dropped into the control, the insertion point will track the mouse cursor until the mouse button is released, causing the object to be inserted. This behavior is the same as Microsoft Word. To print all or part of the text in a **RichTextBox** control use the **SelPrint** method.

Because the **RichTextBox** is a data-bound control, you can bind it with a **Data** control to a Binary or Memo field in a Microsoft Access database or a similar large capacity field in other databases (such as a TEXT data type field in SQL Server).

The **RichTextBox** control supports almost all of the properties, events and methods used with the standard **TextBox** control, such as **MaxLength**, **MultiLine**, **ScrollBars**, **SelLength**, **SelStart**, and **SelText**. Applications that already use **TextBox** controls can easily be adapted to make use of **RichTextBox** controls. However, the **RichTextBox** control doesn't have the same 64K character capacity limit of the conventional **TextBox** control.



For example, the following code sets the **selection's color** to black and makes its font **bold**.

```
1RichTextBox1.SelectionColor = Color.Black
```

```
2RichTextBox1.SelectionFont = New Font(RichTextBox1.Font, FontStyle.Bold)
```

UNIT-V:

Working with Databases

Introduction to Database

Every day we come across all types of information or data such as names, addresses, money, date, stock quotes, statistics and more. If you are in business or working as a professional, you have to handle even more data. For example, a doctor needs to keep track of patients' information such as names, addresses, phone numbers as well as blood pressure readings, blood sugar readings, surgical history, medicines prescribed in the past and more. On the other hand, businesses usually have a large amount of data pertaining to products and customers. All these data need to be organized into a database. The database can then be used to perform MIS functions such as data mining and big data analytics.

In the past, people usually deal with data manually like using cards and folders. However, in present-day fast paced global environment and Information age, it is no longer feasible to manage data manually. Most data are now managed using computer-based database management systems. Computer-based Database management systems can handle data much faster and more efficient than human beings do. With the advent of the network and the Internet technologies, data can now be managed locally and remotely. Companies usually invest heavily in database management systems in order to run the organizations efficiently and effectively.

Creating a Database Application

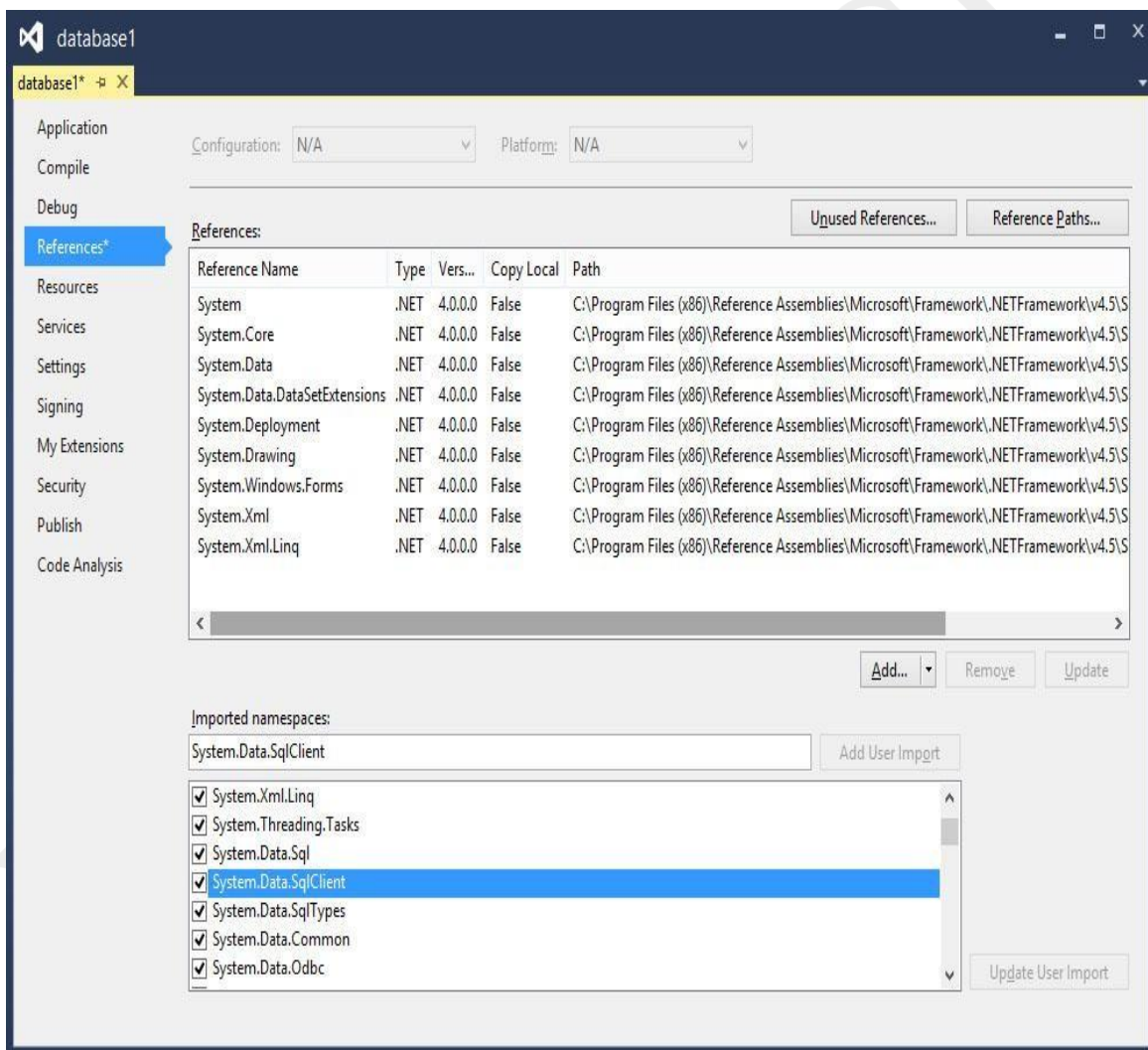
A database management system typically deals with storing, modifying, and extracting information from a database. It can also add, edit and delete records from the database. However, a DBMS can be very difficult to handle by ordinary people or businessmen who have no technological backgrounds. Fortunately, we can create user-friendly database applications to handle the aforementioned jobs with the DBMS running in the background. One of the best programs that can create such database application is none other than Visual Basic 2013.

To begin building the database project in Visual Basic 2012, launch Visual Basic 2012. You can name your project as Database Project 1 or whatever name you wish to call it. Next, change

the default form's Text property to Contacts as we will be building a database of a contact list. There are a few objects in ADO.NET that are required to build the database. There are:

- *SqlConnection*–to connect to a data source in SQL Server
- *DataTable* –to store data for navigation and manipulation
- *DataAdapter*– to populate a DataReader

The aforementioned objects belong to the System.Data and the System.Xml namespace. Therefore, we need to reference them in the beginning before we can work with them. To reference the ADO.NET object, choose project from the menu then select Database Project 1 properties to display the project properties. Next click the References tab to show the active references for the project,



Under imported namespaces, make sure system.data, System.Data.SqlClient is selected, otherwise, check them. Having done that you need to click the Save All button on the toolbar and

P.INDHU MCA.,M.Phil., ASST. PROFESSOR,ANNAI WOMEN'S COLLEGE,KARUR.

then return to the Visual Basic 2013 IDE.

Introduction to Access programming

When you create a new database, you typically begin by creating several database objects such as tables, forms, and reports. Eventually, you reach a point where you have to add some programming to automate certain processes and tie your database objects together.

Use the Command Button Wizard to perform common programming tasks

If you are adding a command button to a form, the Command Button Wizard can help you get started with programming. The wizard helps you create a command button that performs a specific task. In an Access (.accdb) file, the wizard creates a macro that is embedded in the **OnClick** property of the command button. In an .mdb or .adp file, the wizard creates VBA code, because embedded macros are not available in those file formats. In either case, you can then modify or enhance the macro or VBA code to better suit your needs.

1. In the Navigation Pane, right-click the form to which you want to add the command button, and then click **Design View**.
2. On the **Design** tab, click the down arrow to display the **Controls** gallery, and then ensure that **Use Control Wizards** is selected.
3. On the **Design** tab, in the **Controls** gallery, click **Button**.
4. In the form design grid, click where you want the command button to be placed.

The Command Button Wizard starts.

5. On the first page of the wizard, click each category in the **Categories** list to see which actions the wizard can program the command button to perform. In the **Actions** list, select the action that you want, and then click **Next**.
6. Click either the **Text** option or the **Picture** option, depending on whether you want text or a picture to be displayed on the command button.
 - o If you want text to be displayed, you can edit the text in the box next to the **Text** option.
 - o If you want a picture to be displayed, the wizard suggests a picture in the list. If you want to select a different picture, select the **Show All Pictures** check box to display a list of all the command button pictures that Access provides, or click **Browse** to select a picture that is stored elsewhere.


Click **Next**.

7. Enter a meaningful name for the command button. This is an optional step, and this name is not displayed on the command button. However, it is a good idea to enter a meaningful name so that when you need to refer to the command button later (for example, if you are setting the tab order for controls on your form), it will be much easier to differentiate between the command buttons. If the command button closes the form, for example, you might name it cmdClose or CommandClose.
8. Click **Finish**.

Access places the command button on the form.

9. If you want to see what the wizard "programmed" for you, follow these optional steps:

P.INDHU MCA.,M.Phil., ASST. PROFESSOR,ANNAI WOMEN'S COLLEGE,KARUR.

- a. If the property sheet is not already displayed, press F4 to display it.
- b. Click the **Event** tab in the property sheet.
- c. In the On Click property box, click the Build button .

Access starts the Macro Builder and displays the macro that the wizard created. You can edit the macro if you want (for more information about how to edit a macro, see the section Understand macros). When you are finished, on the **Design** tab, in the **Close** group, click **Close** to close the Macro Builder. If Access prompts you to save the changes and update the property, click **Yes** to save the changes or **No** to reject the changes.

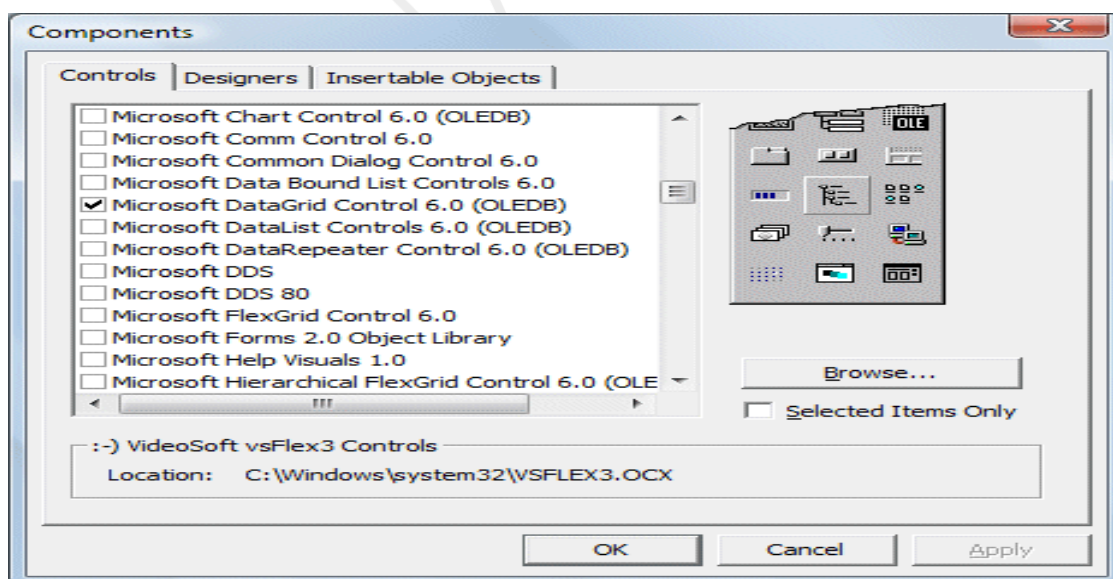
On the **Design** tab, in the **Views** group, click **View**, and then click **Form View**. Click the new command button to confirm that it works as you expected.

Using DataGrid Control

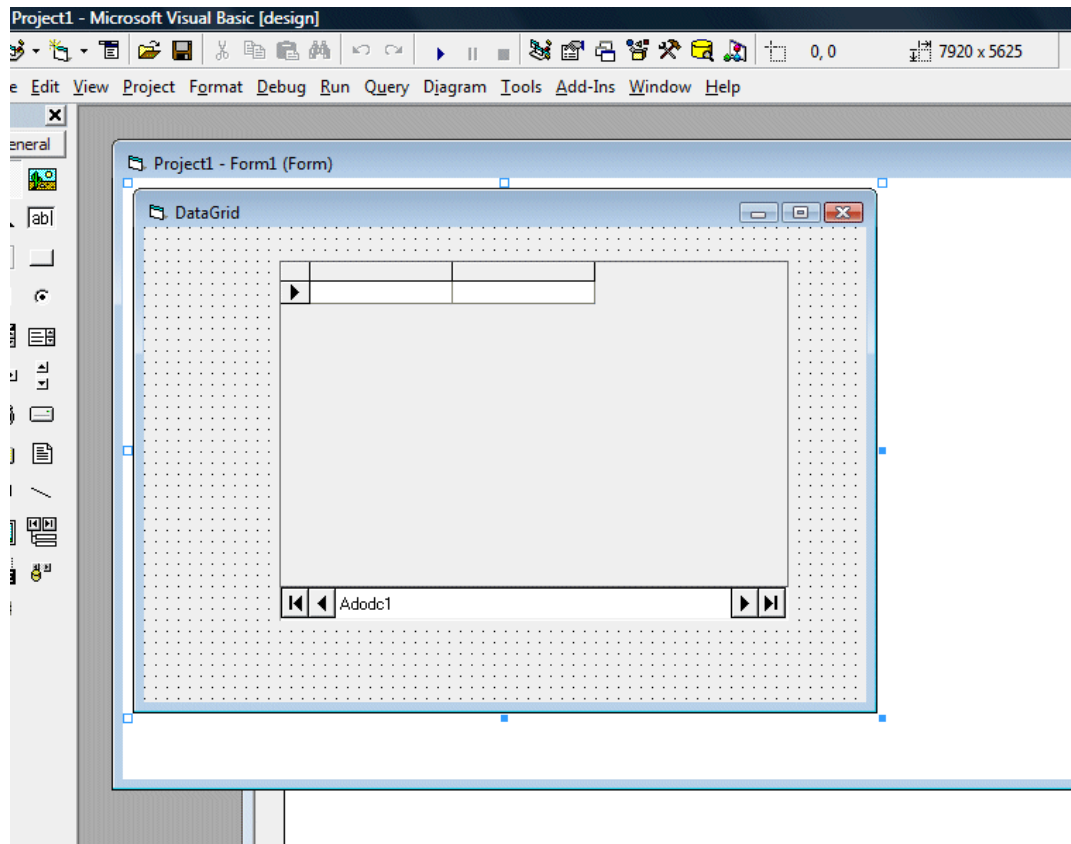
In the previous lesson, we use textboxes or labels to display data by connecting them to a database via Microsoft ADO data Control 6.0. The textbox is not the only control that can display data from a database, many other controls in Visual Basic can display data. One of them is the **DataGrid control**. DataGrid control can be used to display the entire table of a recordset of a database. It allows users to view and edit data.

DataGrid control is not the default item in the Visual Basic control toolbox, you have to add it from the VB6 components. To add the DataGrid control, click on the Project on the menu bar and select components to access the dialog box that displays all the available VB6 components, as shown in the diagram below. Select **Microsoft DataGrid Control 6.0** by clicking the checkbox beside this item. Before you exit the dialog box, you also need to select the **Microsoft ADO data control** so that you are able to access the database. Last, click on the OK button to exit the dialog box. Now you should be able to see that the DataGrid control and the ADO data control are added to the toolbox. The next step is to drag the DataGrid control and the ADO data control into the form.

The components dialog box is shown below:



The Components Dialog Box



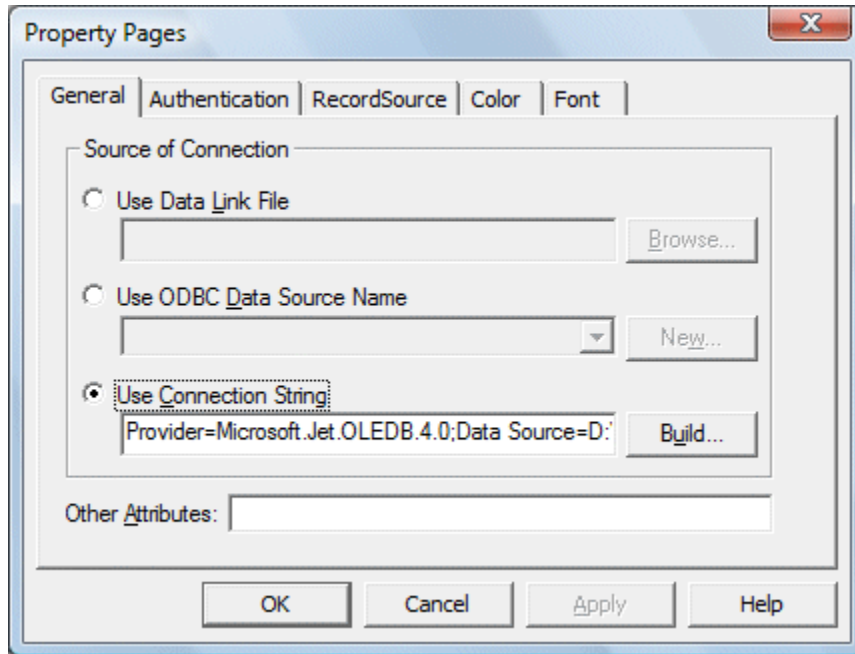
The Design Interface

Before you proceed, you need to create a database file using Microsoft Access. Here we create a database file to store the information of books and we named the table **book**. Having created the table, enter a few records, as shown in Figure 26.3 below:

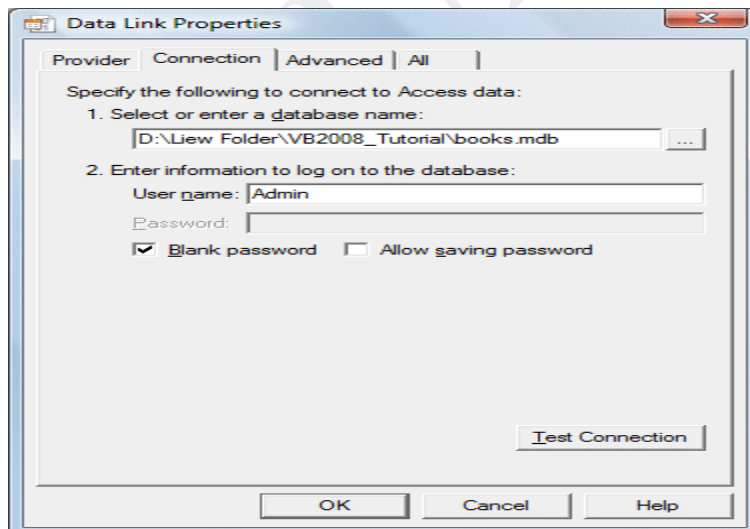
The screenshot shows the Microsoft Access interface. The title bar reads 'Microsoft Access'. The menu bar includes 'File', 'Edit', 'View', 'Insert', 'Format', 'Records', 'Tools', 'Window', and 'Help'. The toolbar contains various icons for database operations. The main window shows a database named 'books : Database (Access 2000 file format)'. A table named 'book : Table' is displayed with the following data:

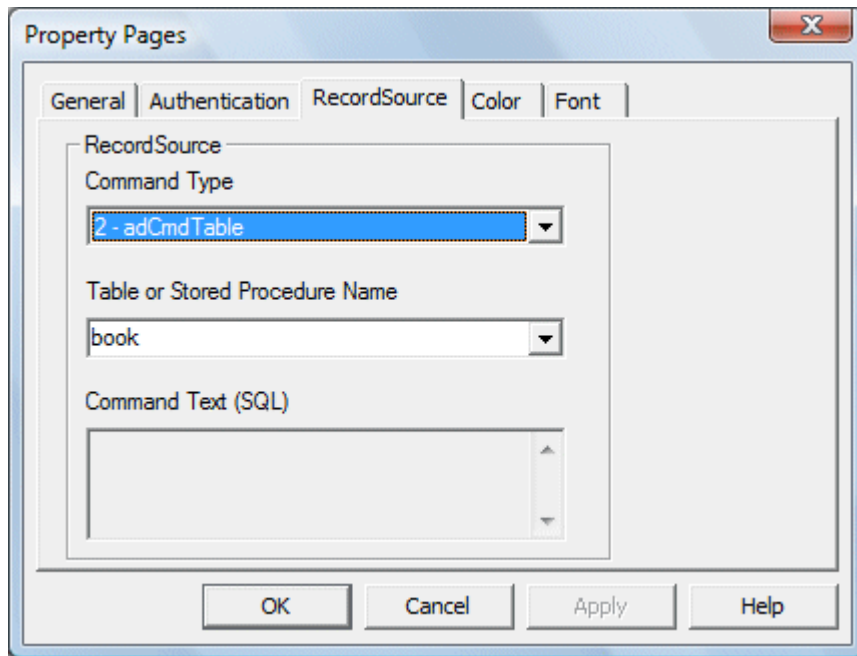
ID	Title	Author	Year	ISBN	Publisher	Price
1	Visual Basic 6	Deitel & Deitel	1999	ISBN 0-13-1228	Prentice-Hall, Ir	\$77.60
2	Secrets of Inter	Stuart Tan	2007	ISBN 978-981-0	Seng Lee Press	\$49.90
3	Visual Basic 6	P.Sellappan	2001	ISBN 983-2017	Sejana Publishi	\$32.00
*	(AutoNumber)					\$0.00

Next, you need to connect the database to the ADO data control. To do that, right click on the ADO data control and select the **ADODC** properties, the following dialog box will appear.

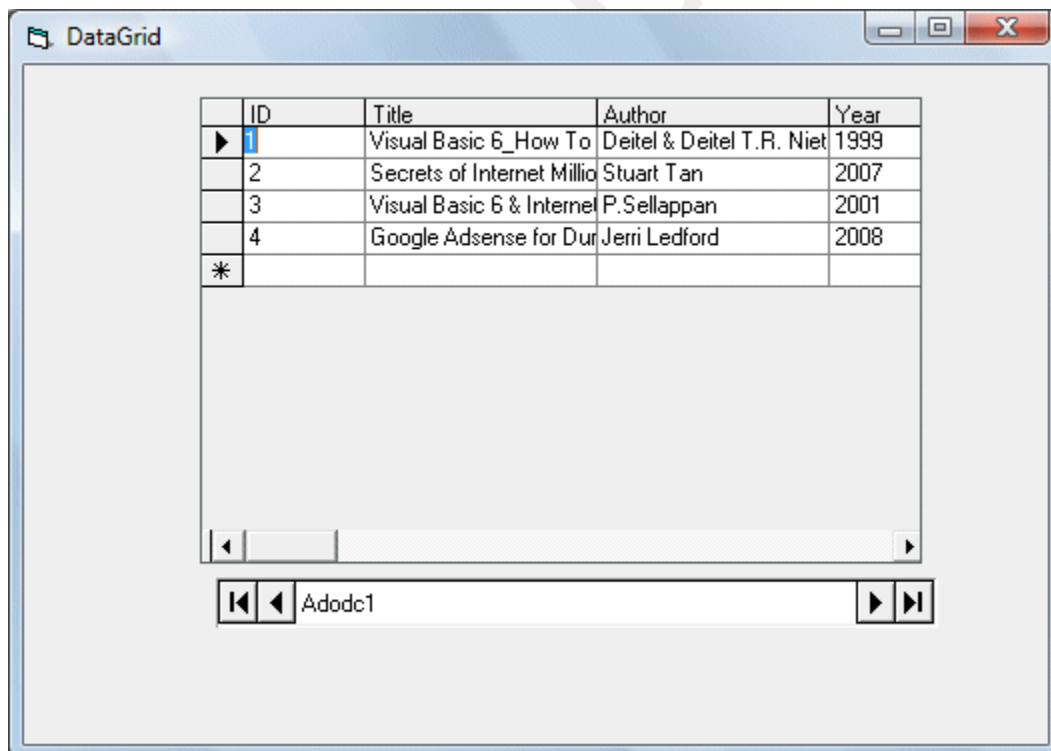


Next click on the Build button and the Data Link Properties dialog box will appear . In this dialog box, select the database file you have created, in my case, the file name is **books.mdb**. Press test connection to see whether the connection is successful. If the connection is successful, click OK to return to the ADODC property pages dialog box. At the ADODC property pages dialog box, click on the Recordsource tab and select 2-adCmdTable under command type and select book as the table name, then click OK.





Finally you need to display the data in the DataGrid control. To accomplish this, go to the properties window and set the DataSource property of the DataGrid to Adodc1. You can also permit the user to add and edit your records by setting the AllowUpdate property to True. If you set this property to false, the user cannot edit the records. Now run the program and the runtime interface.

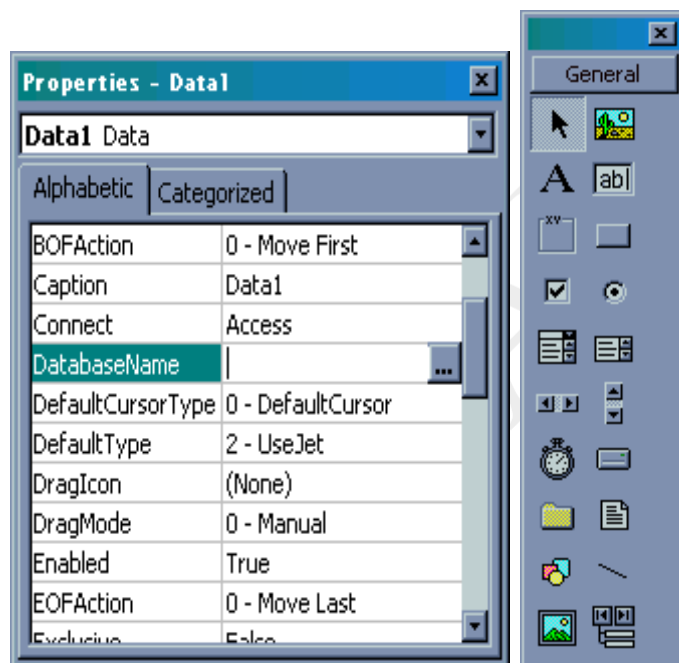


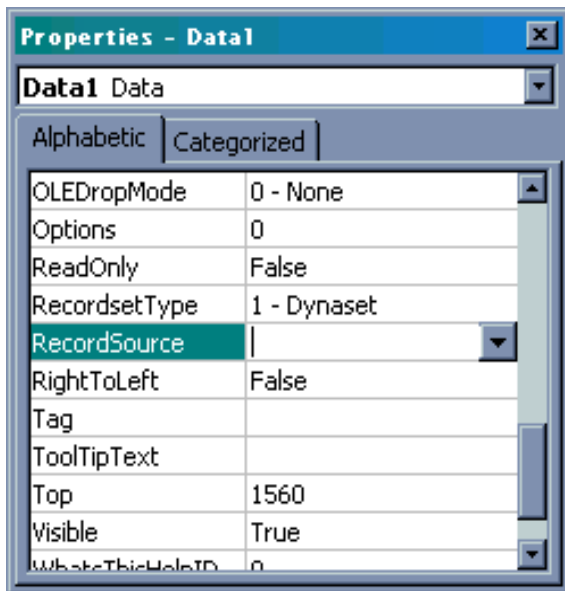
Data Access Objects

Declaring a Data Bound Control

A data bound control connects a data control in Visual Basic to a database table or query. Data controls are visual objects that are said to be data-aware. Data controls may include check boxes, images, labels, picture boxes, and text boxes (see figure A). Visual Basic's data controls allow users to access stored database records. The data control establishes a link between the database and other controls in the interface, in a process called binding. When a control is bound, it displays database field contents when the Data control is present.

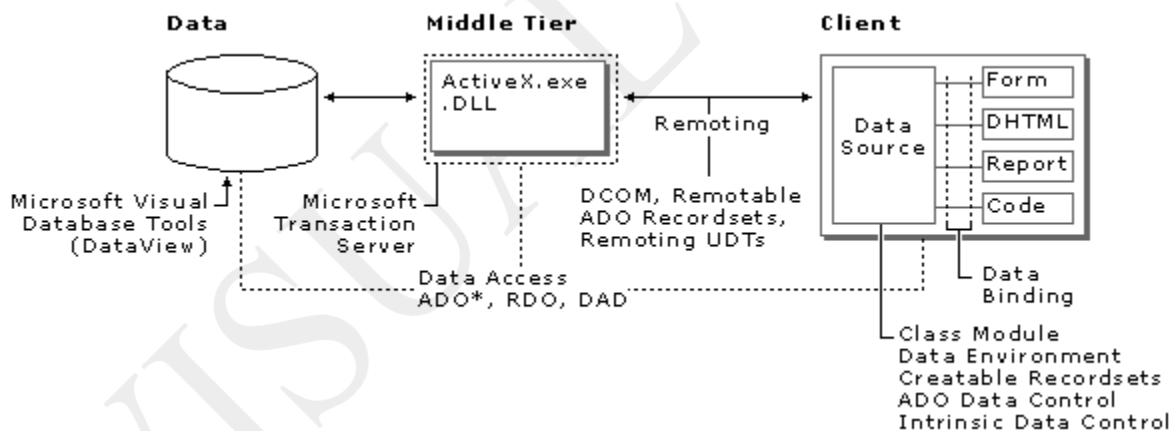
Required property settings for the Data control include the **DatabaseName** property which specifies the complete path to the database , and the **RecordSource** property, which indicates the table to use from the database . The **DataSource** property (specifies the name of the Data control to which it is bound) and the **DataField** property (specifies the name of a field in the database to which the control is linked) should be set for all form controls that access database information (Cashman, VB 4).





Accessing Data Using Visual Basic

The figure below is a roadmap of data access technologies found in Visual Basic. The figure features "hot" zones, which you can click to find out more information about any particular set of data, access tools or technologies.



Microsoft Visual Data Tools

Using Visual Basic 6.0 you can create components that encapsulate every step in a data access system. Beginning with the data source, Microsoft Visual Data Tools (accessible through the Data View window) give you the ability to view and manipulate tables, views, stored procedures, and database schemas on SQL Server and Oracle systems.

Middle Tier Components and Microsoft Transaction Server

The power of Visual Basic is also leveraged to create the middle tier components in your application, as you make your own ActiveX DLLs and EXEs. Visual Basic now includes enhancements that tailor applications to work with Microsoft Transaction Server.

ActiveX Data Objects (ADO)

The bridge between the data providers and data consumers is through data sources created using Microsoft ActiveX Data Objects (ADO), which is the primary method in Visual Basic to access data in any data source, both relational and non-relational. For backward compatibility and project maintenance, Remote Data Objects (RDO) and Data Access Objects (DAO) are still supported.

Data Sources and Data Controls

On the client side, several new data sources are available, including the Data Environment, a graphical designer that allows you to quickly create ADO Connections and Commands to access your data. The Data Environment designer provides a dynamic programmatic interface to the data access objects in your project. In addition, the Data Environment provides advanced data shaping services — the ability to create hierarchies of related data, aggregates, and automatic groupings, all without code.

The new ADO Data control is similar to the intrinsic data control and Remote Data control, except that it uses ADO to access data. You can now use an ADO Recordset as a data source for your controls and objects in Visual Basic.

In Visual Basic you can now create your own data sources either as user controls or classes, to encapsulate business rules or proprietary data structures. The class module now features the DataSourceBehavior property and the GetDataMember event, which allow you to configure a class as a data source.

Dynamic Data Binding

The ability to dynamically bind a data source to a data consumer is now possible in Visual Basic. At run time, you can now set the DataSource property of a data consumer (such as the DataGrid control) to a data source (such as the ADO Data control). This capability, unavailable in previous versions of Visual Basic, allows you to create applications, which can access a multitude of data sources.

Presenting Data to the End User

Visual Basic offers a variety of rich ways to present data to your end users. ADO/OLE DB-based versions of all the data bound controls are included in Visual Basic:

- The DataList and DataCombo controls are the ADO/OLE DB equivalents of DBList and DBCombo controls.
- The DataGrid is the successor to DBGrid.
- The Chart control is now data bound.
- A new version of the FlexGrid control, called the Hierarchical FlexGrid, supports the hierarchical abilities of the Data Environment.
- The new DataRepeater control functions as a scrolling container of data bound user controls where each control views a single record.

The Data Report is a new ActiveX designer that creates reports from any data source, including the Data Environment. With the Data Report designer, formatted reports can be viewed online, printed, or exported to text or HTML pages.

Data Formatting and Data Validation

The new DataFormat object allows you to display data with custom formatting, but write it back to the database in the native format. For example, you can now display dates in the format appropriate to a country, while the actual data is stored in a date format. Data is formatted coming out of the source, and unformatted going back in. You can also do custom formatting and perform additional checks using the Format and Unformat events.

Data validation is also enhanced using the CausesValidation property with the Validate event. By setting the CausesValidation property to True, the Validate event for the previous control in the tab order will occur. Thus, by programming the Validate event, you can prevent a control from losing focus until the information it contains has been validated.

Language Features

New data-related enhancements to the Visual Basic language include the ability to pass User-defined Types (UDTs) and arrays across processes. You can now define a UDT and pass it as a parameter to another process, such as an ActiveX EXE or DLL.

DHTML and Data Access

Using Visual Basic, you can create complete web applications for data access. All of the data tools and technologies can also be used in DHTML pages, and on web server (IIS) applications.

OPEN DATABASE CONNECTIVITY (ODBC)

The main proponent and supplier of ODBC programming support is Microsoft, but ODBC is based on and closely aligned with The Open Group standard Structured Query Language (SQL) Call-Level Interface (CLI). The Open Group is sponsored by many major vendors, including Oracle, IBM and Hewlett Packard Enterprise, and this consortium develops and manufactures The Open Group Architecture Framework (TOGAF). In addition to CLI specifications from The Open Group, ODBC also aligns with the ISO/IEC for database APIs.

Microsoft database programming object sets:

- Data Access Objects (DAO),
- Remote Data Objects (RDO), and
- ActiveX Data Objects (ADO).

DAO:

When Visual Basic first started working with databases, it used the Microsoft Jet database engine, which is what Microsoft Access uses. To support the Jet database engine, Microsoft added the data control to Visual Basic, and you can use that control to open Jet database (.mdb) files. Microsoft also added a set of Data Access Objects (DAO) to Visual Basic:

- *DBEngine*—The Jet database engine
- *Workspace*—An area can hold one or more databases
- *Database*—A collection of tables
- *TableDef*—The definition of a table
- *QueryDef*—The definition of a query
- *Recordset*—The set of records that make up the result of a query
- *Field*—A column in a table

- *Index*—An ordered list of records
- *Relation*—Stored information about the specific relationship between tables

The Data Control

The *Data control* gives you access to databases without any programming. You can set a few properties of the control and use regular controls such as textboxes to display the values of the fields in the database.

The data control's **Database** and **Recordset** properties refer to those Database and Recordset objects, and you can manipulate the data using those properties.

RDO:

Remote Data Objects (RDO) connects to databases using ODBC. You set up ODBC connections to databases using the ODBC item in the Windows Control Panel, and then use one of those connections with the RDO objects. The Remote Data Objects are designed in parallel with the Data Access Objects; for example, the database engine is *rdoEngine* instead of *DBEngine*, Recordsets have become *rdoResultsets*, *TableDefs* became *rdoTables*, *Workspaces* became *rdoEnvironments*, *Field* objects became *rdoColumn* objects, and so on. Although the names have changed, the command set is very similar to DAO.

The Remote Data Control

Like the data control, the remote data control gives you access to a database and displays data in bound controls. Unlike the data control, however, you use the remote data control to access ODBC data sources.

In fact, the remote data control behaves like the data control in most respects, with some differences; for example, you can treat the remote data control's **SQL** property like the data control's **RecordSource** property, but it cannot accept the name of a table by itself unless you populate the **rdoTables** collection first.

ADO:

Microsoft's latest set of data access objects are the ActiveX Data Objects (ADO). These objects let you access data in a database server through any OLE DB provider. Here are the ADOs:

- *Connection*
- *Command*
- *Parameter*
- *Recordset*
- *Field*
- *Erro*
- *Collection*
- *Event*

The ADO Data Control

The ADO data control is similar to the data control and the remote data control. The ADO data control is designed to create a connection to a database using Microsoft ActiveX Data Objects (ADO). At design time, you create a connection by setting

the **ConnectionString** property to a valid connection string, then set the **RecordSource** property to a statement appropriate to the database manager.

Introduction:

Visual Basic .NET is an Object-Oriented programming language designed by Microsoft. With the word -Basic| being in the name of the language, you can already see that this is a language for beginners. Although the language is aimed at noobs and novices, you should not underestimate the power of the language itself. There are people who criticize VB.NET because of the simplicity of the syntax, but VB.NET has the ability to create very powerful and sophisticated applications. VB.NET is a great place to start because of how easy and straight forward it is. The syntax is easy and you will not find yourself writing hundreds of lines of code as there are many shortcuts that make coding so much easier in this language.

If you have had any experience in computer programming, you should understand what a syntax is and the purpose of it. If not, let's take a look at the VB.NET syntax. The purpose of typing code is to instruct the application what to do. It's not as easy as typing -Hey application, multiply 5 by 8" but it's pretty darn close! If you wanted to tell your application to show a Message Box telling you that HowToStartProgramming.com is awesome, this would be the code you would use:

```
MessageBox.Show("HowToStartProgramming.com is awesome")
```

Visual Basic .NET (VB.NET or VB .NET)

Visual Basic .NET (VB.NET or VB .NET) is a version of Microsoft's Visual Basic that was designed, as part of the company's .NET product group, to make Web services applications easier to develop. According to Microsoft, VB .NET was reengineered, rather than released as VB 6.0 with added features, to facilitate making fundamental changes to the language. VB.NET is the first fully object-oriented programming (OOP) version of Visual Basic, and as such, supports OOP concepts such as abstraction, inheritance, polymorphism, and aggregation.

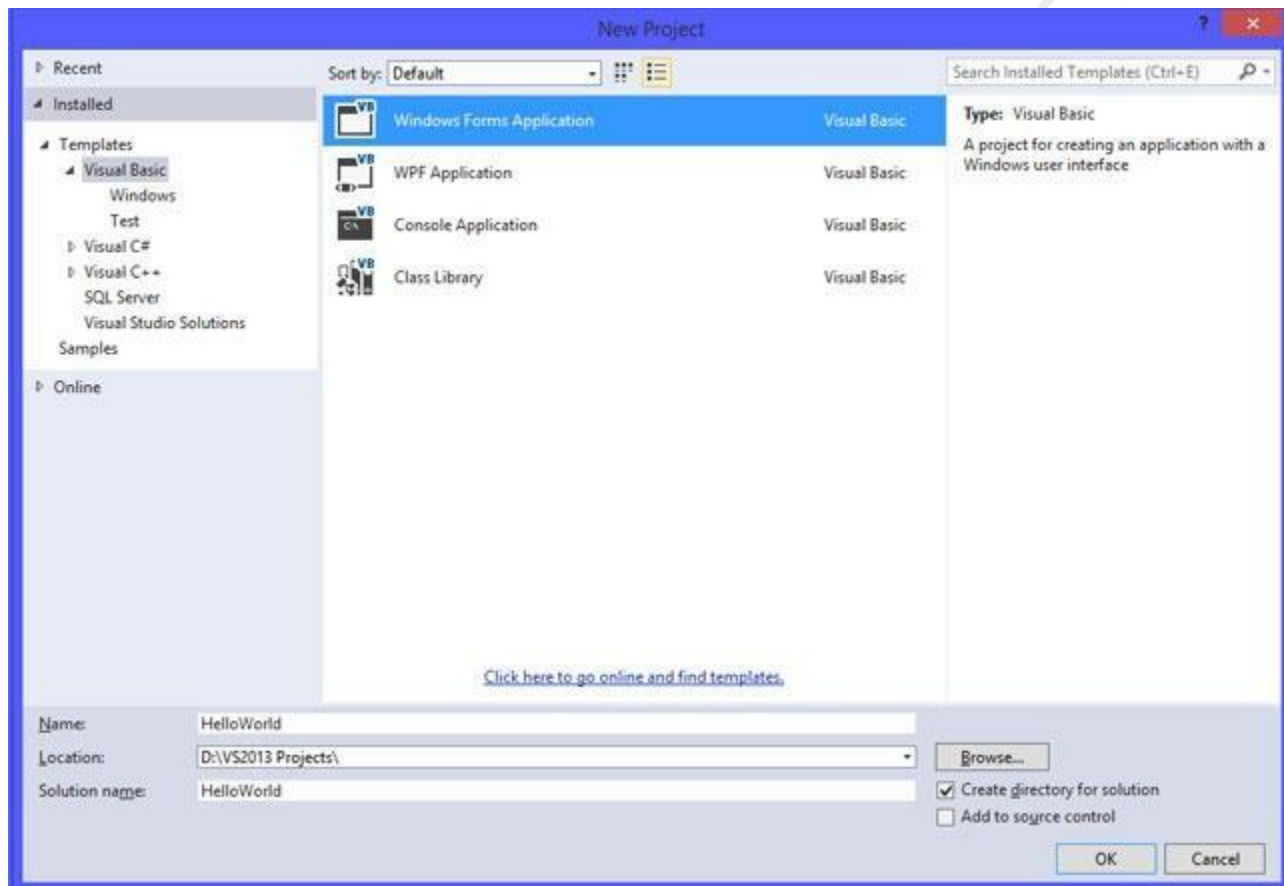
Getting to know the IDE - Visual Basic .NET

Visual Basic applications and how to use the IDE to debug your programs. Visual Studio IDE is extremely customizable which means that you can move, hide, or modify the menus, toolbars, and windows. You can create your own toolbars and then dock, undock, or rearrange them. Finally you can change the behavior of the built-in text editors and much more.

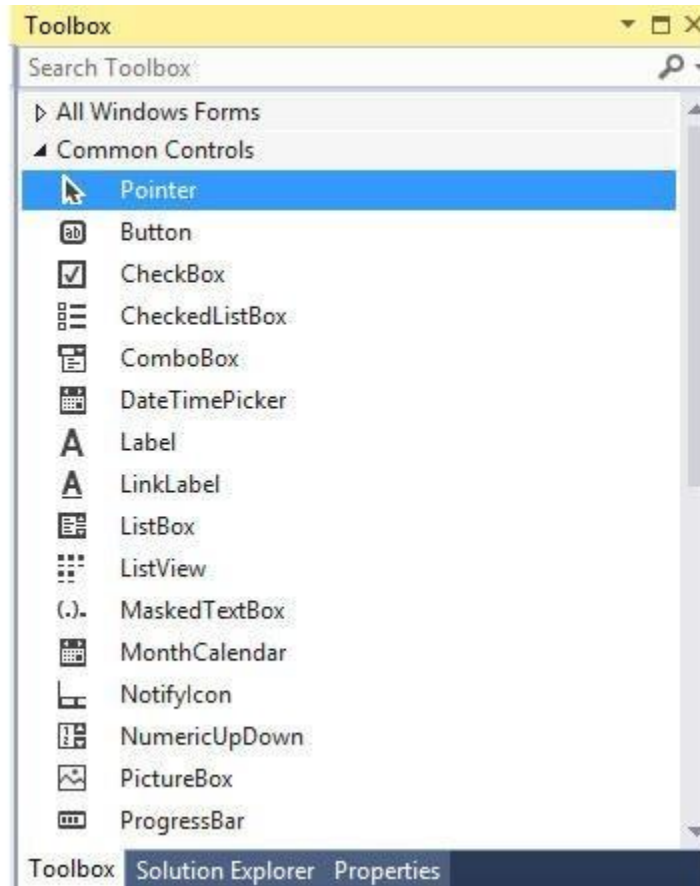
However as beginner you should probably not customize the IDE's basic menus and toolbars too much because it will only cause confusion later.

1. The New Project dialog lets you start a new project.

Use the Project Types tree view on the left to select the project category that you want. After that select a specific project type on the right (as you see we have selected the Windows Forms Application project type). Finally set the location (default is My Documents), enter a name for the new project and click OK to create the project.



2. The Toolbox window displays tools that you could use with the currently active document. These tools are available when you are editing an item that can contain objects such as controls and components. By the way, these tools are grouped into sections callif you right-click a certain tab and select one of the commands in the context menu.

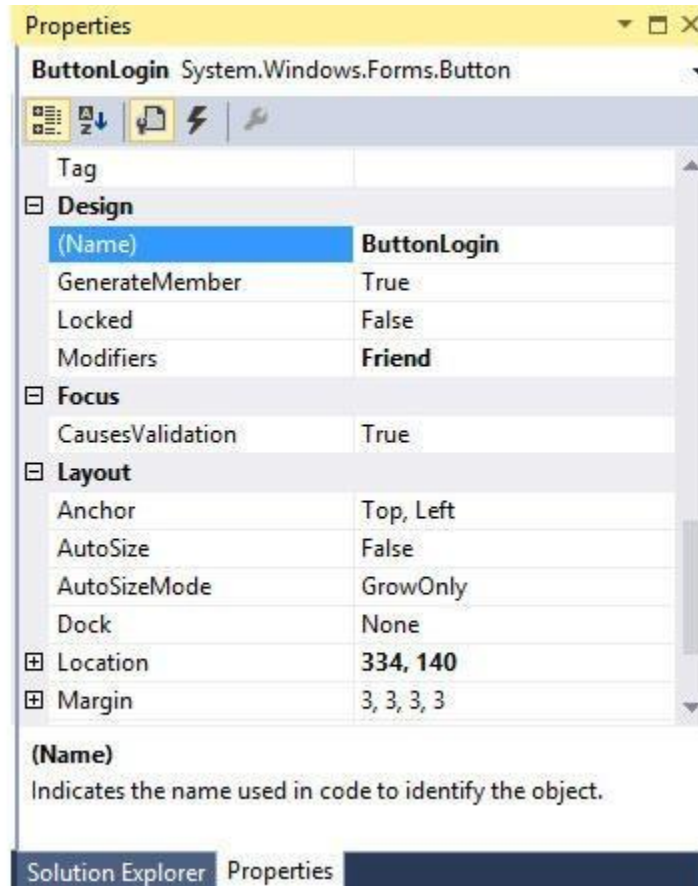


3. The Properties window allows you to view and modify the properties of the form and of the controls that it contains.

On our screenshot you may see the Properties window displaying properties for a Button control named ButtonLogin. You can see in the figure that the Text property of this control is "Login" and that's what the button displays to the user.

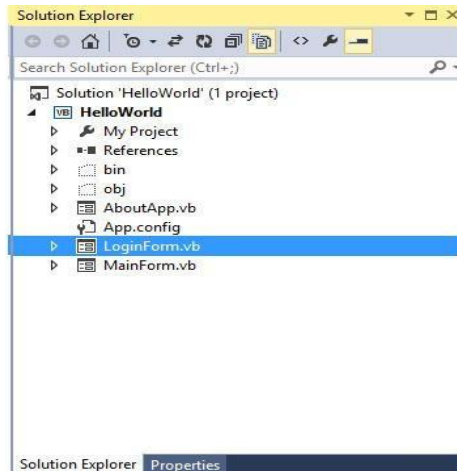
In addition please note the drop-down list at the top of the window. Well that list holds the names of all of the controls on the form. To select a certain control, you can either click it on the Designer or select it from this drop-down list.

The small icons below the dropdown determine what items are displayed in the window and how they are arranged. For example if you click the leftmost button, the window lists properties grouped by category and if you click the second icon that holds the letters A and Z, the window lists the control's properties alphabetically.



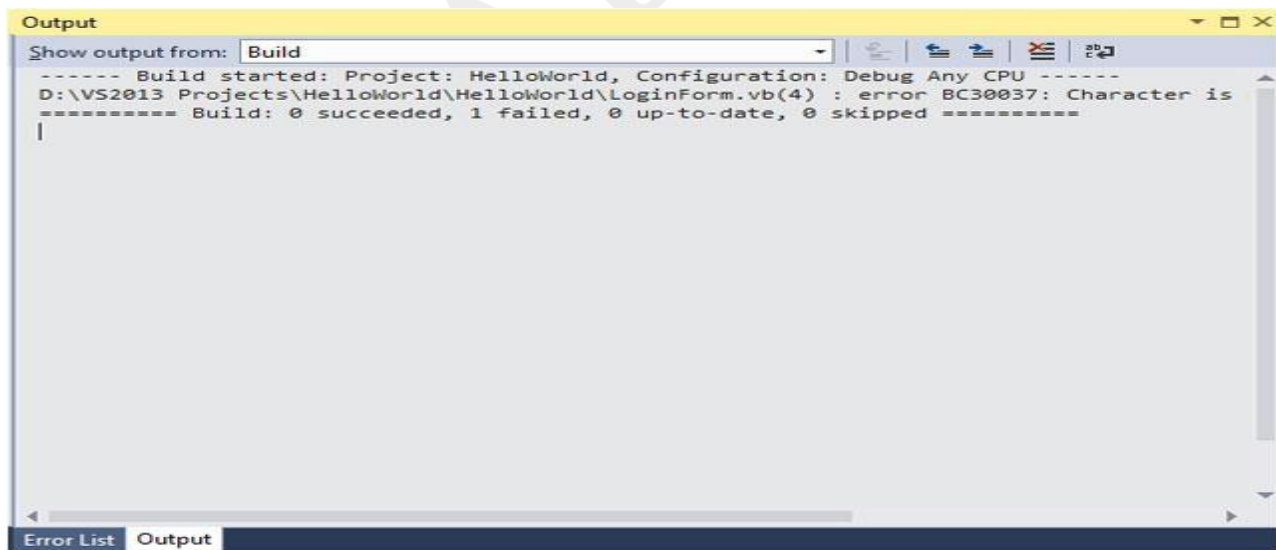
4. The Solution Explorer lets you manage the files associated with the current solution. For example, in the Figure below, you could select `LoginForm.vb` in the Solution Explorer and then click the View Code button (the third icon from the right at the top of the Solution Explorer) to open the form's code editor.

You can also right-click an object in the Solution Explorer to get a list of appropriate commands for that object. This window makes it easier to find a command by right-clicking an object related to whatever you want to do than it is to wander through the menus.

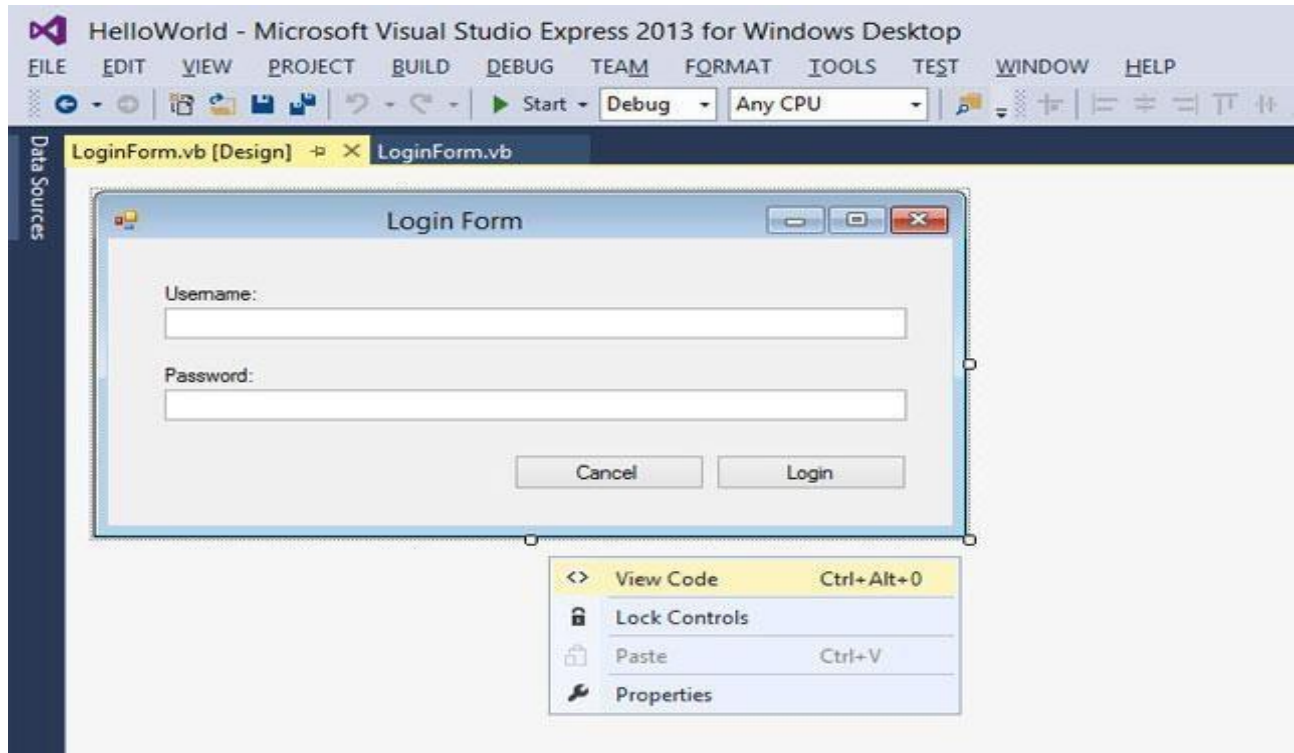


5. The Error List window shows errors and warnings in the current project. For example, if the code contains invalid character, this list will say so. It's extremely useful as it clearly tells you the type of the error(s) showing a full description, file name, line and everything else that helps you easily find and fix the error. If you don't see the Error List it is probably hidden. You can display it by selecting the appropriate item in the View menu.

6. The Output window displays compilation results and output printed by the application. Usually an application interacts with the user through its forms and dialog boxes, but it can display information here, usually to help you debug the code. The Output window also shows informational messages generated by the IDE. For example, when you compile an application, the IDE sends messages here to tell you what it is doing and whether it succeeded.



7. The Windows Forms Designer allows you to design forms for typical Windows applications. It lets you add, size, and move controls on a form using your mouse. Together with the Properties window, it lets you view and modify control properties, and create event handlers to interact with the controls.



8. You use the Visual Basic Code Editor to write a code that responds to control events.

The most obvious feature of the code editor is that it lets you type code, but the code editor is far more than a simple text editor such as Notepad.

It provides many features to make writing correct Visual Basic code much easier. For example you can create an event handler within the code editor. The upper left part of the code editor displays a drop-down listing all the controls.

If you select a control from the list, you can then pick an event for that control from a second dropdown in the code editor's upper right. If you select an event, the code editor generates a corresponding empty event handler for you.

To make referring to the code lines easier, code editor may display line numbers too.

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

Public Class LoginForm

    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        Application.Exit()
    End Sub

    Private Sub ButtonLogin_Click(sender As Object, e As EventArgs) Handles ButtonLogin.Click
        If Username.Text = "admin" AndAlso Password.Text = "mypwd" Then
            Dim mform As Form = MainForm
            mform.ShowDialog()
            Me.Close()
        Else
            MessageBox.Show("You entered a wrong username or password. Please try again.")
        End If
    End Sub
End Class
```

The Common Language Runtime (CLR)

As introduced in the preceding section, C# applications are compiled to IL, which is executed by the CLR. This section highlights several features of the CLR. You'll also see how the CLR manages your application during execution.

Why Is the CLR Important?

In many traditional execution environments of the past, programmers needed to perform a lot of the low-level work (plumbing) that applications needed to support. For example, you had to build custom security systems, implement error handling, and manage memory.

The degree to which these services were supported on different language platforms varied considerably. Visual Basic (VB) programmers had built-in memory management and an error-handling system, but they didn't always have easy access to all the features of COM+, which opened up more sophisticated security and transaction processing. C++ programmers have full access to COM+ and exception handling, but memory management is a totally manual process.

In a later section, you learn about how .NET supports multiple languages, but knowing just a

P.INDHU MCA.,M.Phil., ASST. PROFESSOR,ANNAI WOMEN'S COLLEGE,KARUR.

little about a couple of popular languages and a couple of the many challenges they must overcome can help you to understand why the CLR is such a benefit for a C# developer.

The CLR solves many problems of the past by offering a feature-rich set of plumbing services that all languages can use. The features described in the next section further highlight the value of the CLR.

CLR Features

This section describes, more specifically, what the CLR does for you. Table 1.1 summarizes CLR features with descriptions and chapter references (if applicable) in this book where you can find more detailed information.

Table 1.1. CLR Features

Feature	Description
.NET Framework Class Library support	Contains built-in types and libraries to manage assemblies, memory, security, threading, and other runtime system support
Debugging	Facilities for making it easier to debug code. (Chapter 7)
Exception management	Allows you to write code to create and handle exceptions. (Chapter 11)
Execution management	Manages the execution of code
Garbage collection	Automatic memory management and garbage collection (Chapter 15)
Interop	Backward-compatibility with COM and Win32 code. (Chapter 41)
Just-In-Time (JIT) compilation	An efficiency feature for ensuring that the CLR only compiles code just before it executes

Feature	Description
Security	Traditional role-based security support, in addition to Code Access Security (CAS) (Chapter 44)
Thread management	Allows you to run multiple threads of execution (Chapter 39)
Type loading	Finds and loads assemblies and types
Type safety	Ensures references match compatible types, which is very useful for reliable and secure code (Chapter 4)

In addition to the descriptions provided in Table 1.1, the following sections expand upon a few of the CLR features. These features are included in the CLR execution process.

CommonTypeSystem (CTS)

It describes set of data types that can be used in different .Net languages in common. (i.e), CTS ensures that objects written in different .Net languages can interact with each other. For Communicating between programs written in any .NET complaint language, the types have to be compatible on the basic level.

The common type system supports two general categories of types:

Valuetypes:

Value types directly contain their data, and instances of value types are either allocated on the stack or allocated inline in a structure. Value types can be built-in (implemented by the runtime), user-defined, or enumerations.

Referencetypes:

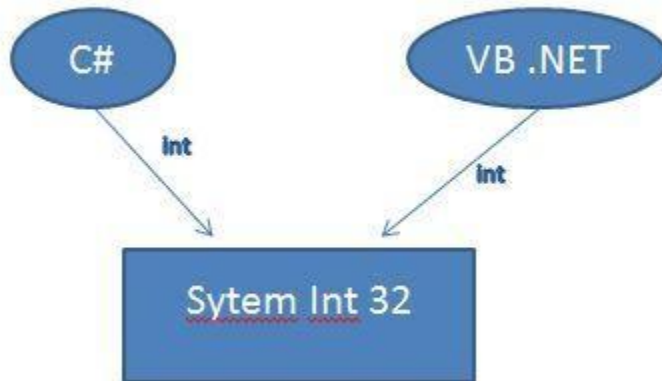
Reference types store a reference to the value's memory address, and are allocated on the heap. Reference types can be self-describing types, pointer types, or interface types. The type of a reference type can be determined from values of self-describing types. Self-describing types are further split into arrays and class types. The class types are user-defined classes, boxed value types, and delegates.

CTS:

The Common Type System (CTS) standardizes the data types of all programming languages using .NET under the umbrella of .NET to a common data type for easy and smooth communication among these .NET languages.

How CTS converts the data type to a common data type

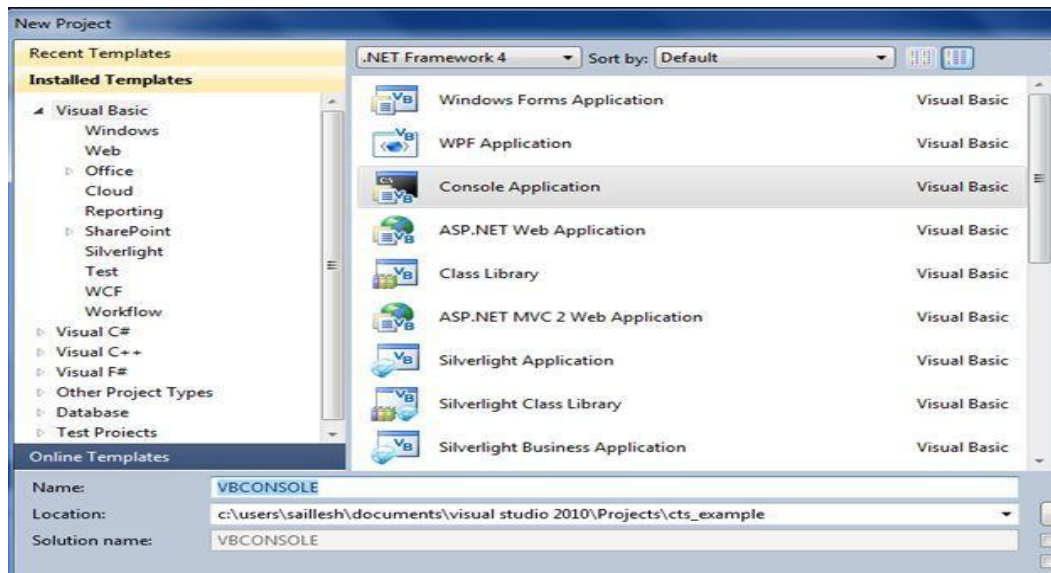
To implement or see how CTS is converting the data type to a common data type, for example, when we declare an int type data type in C# and VB.Net then they are converted to int32. In other words, now both will have a common data type that provides flexible communication between these two languages.



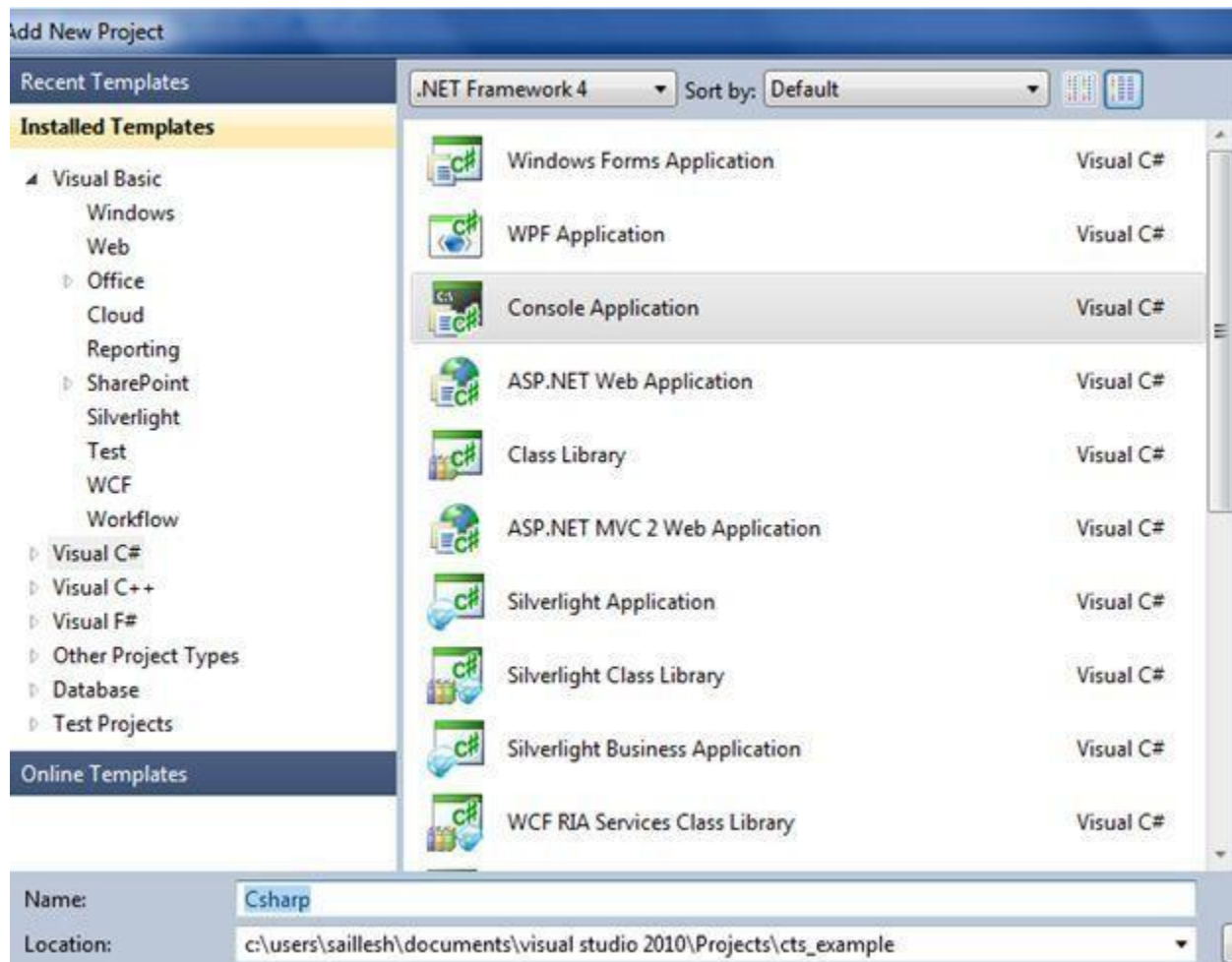
Let's take a live example using a Double data type and we will see how .NET helps to make the data types of C# and VB.NET common to each other for easy communication.

Live Example

We will create a new Visual Basic (Console Application) project.



Now we will add one more project of Visual C# (Console Application).



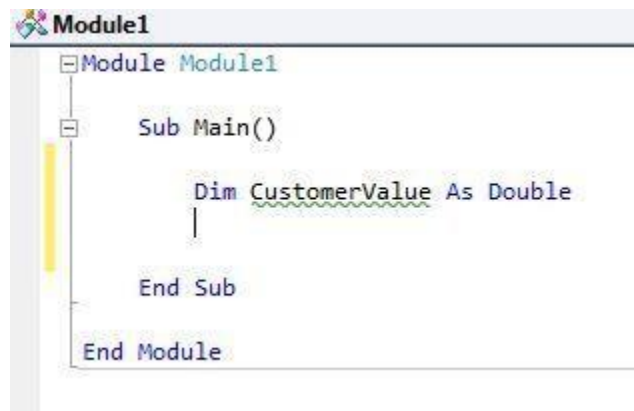
Now to see how CTS is helping with the data types of the programming languages and providing a common environment we will declare a Double data type (*note you can use other variables for your reference) variable in both the C# program.cs and Module1.vb.

C# Data type Declaration

```
Csharp.Program
{
    using System;
    using System.Collections.Generic;
    using System.Linq;
    using System.Text;

    namespace Csharp
    {
        class Program
        {
            static void Main(string[] args)
            {
                Double CustomerValue;
            }
        }
    }
}
```

Visual Basic Data type Declaration



```
Module1
├── Module Module1
│   ├── Sub Main()
│   │   ├── Dim CustomerValue As Double
│   │   └──
│   └── End Sub
└── End Module
```

Now start debugging both of these Console Applications. Once you are done with Debugging the application, we will now use an important tool provided with Visual Studio Called *-ILDASM*.