Shiksha Mandal's

G.S. College of Commerce, Wardha

Department of B. Com Computer Application

BCCA Part-II Sem-III

Visual Basics Programming

UNIT-I

Introduction to Visual Basics

Presented By

Prof. Amol Raut

Assistant Professor

Dr. Revati Bangre

Co-ordinator

BCCA Department

Dr. Anil Ramteke

Principal (Officiating)

Visual Basics Programming

Unit-I

Introduction to 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** means **B**eginners' **A**ll-purpose **S**ymbolic **I**nstruction **C**ode. Since thenMicrosoft 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.

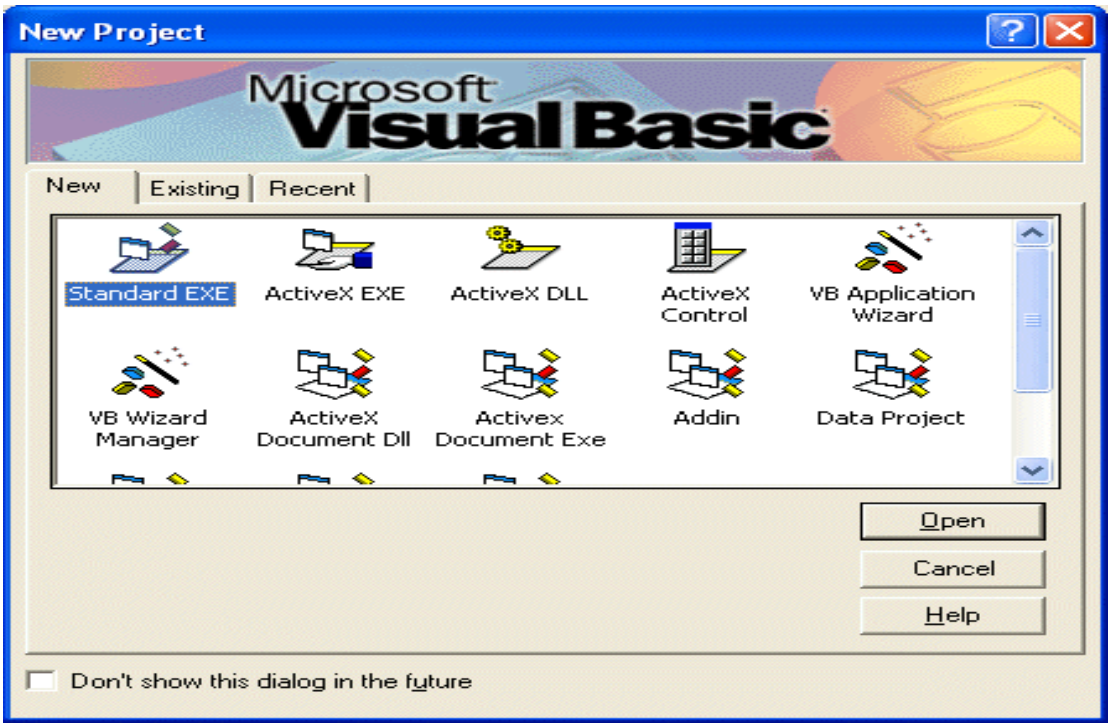
In 2002, Microsoft released Visual Basic.NET (VB.NET) to replace Visual Basic 6. Thereafter, Microsoft declared VB6 a legacy programming language in 2008. Fortunately, Microsoft still provides some form of support for VB6. VB.NET is a fully object-oriented programming language implemented in the .NET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

What is Visual Basic?

VISUAL BASIC is a high-level programming language which evolved from the earlier DOS version called BASIC. BASIC means Beginners' All-purpose Symbolic Instruction Code. It is a relatively easy programming language to learn. The code looks a lot like English Language. people prefer to use Microsoft Visual Basic today, as it is a well-developed programming language and supporting resources are available everywhere. Now, there are many versions of VB exist in the market, the most popular one and still widely used by many VB programmers is none other than Visual Basic 6. We also have VB.net, VB2005, VB2008 and the latest VB2010. Both Vb2008 and VB2010 are fully object-oriented programming (OOP) languages. We can use the Event driven program also object-oriented programming methods use in VB 2008 and VB 2010. Now it has released VB 2012 version with Visual studio 2012.it just work with the Windows 8.

VISUAL BASIC is a VISUAL and events driven Programming Language. These are the main divergence from the old BASIC. In BASIC, programming is done in a text-only environment and the program is executed sequentially. In VB, programming is done in a graphical environment. (It's provide the fully graphical environment for developer) In the old BASIC, you have to write program code for each graphical object you wish to display it on

screen, including its position and its color. However, In VB, you just need to drag and drop any graphical object anywhere on the form, and you can change its color any time using the properties window. (you can see later in my posts) On the other hand, because the user may click on a certain object randomly, so each object has to be programmed independently to be able to response to those actions (events). Therefore, a VB Program is made up of many subprograms, each has its own program code, and each can be executed independently and at the same time each can be linked together in one way or another (Example: Child widows and parent windows). Here the start page of VB6.0 software.



Advantages of Visual Basic

1. The structure of the Basic programming language is very simple, particularly as to the executable code.
2. VB is not only a language but primarily an integrated, interactive development environment ("IDE").
3. The VB-IDE has been highly optimized to support rapid application development ("RAD"). It is particularly easy to develop graphical user interfaces and to connect them to handler functions provided by the application.
4. The graphical user interface of the VB-IDE provides intuitively appealing views for the management of the program structure in the large and the various types of entities (classes, modules, procedures, forms, …).
5. VB provides a comprehensive interactive and context-sensitive online help system.

6. When editing program texts the "IntelliSense" technology informs you in a little popup window about the types of constructs that may be entered at the current cursor location.
7. VB is a component integration language which is attuned to Microsoft's Component Object Model ("COM").
8. COM components can be written in different languages and then integrated using VB.
9. Interfaces of COM components can be easily called remotely via Distributed COM ("DCOM"), which makes it easy to construct distributed applications.
10. COM components can be embedded in / linked to your application's user interface and also in/to stored documents (Object Linking and Embedding "OLE", "Compound Documents").
11. There is a wealth of readily available COM components for many different purposes.

Disadvantages

1. Visual basic is a proprietary programming language written by Microsoft, so programs written in Visual basic cannot, easily, be transferred to other operating systems.
2. There are some, fairly minor disadvantages compared with C. C has better declaration of arrays – it's possible to initialize an array of structures in Cat declaration time; this is impossible in VB.

Feature of visual basic 6.0

1. Learning Consists of all necessary tools required to build main stream Windows Applications

2. Professional Includes advanced features such as tools to develop ActiveX and Internet controls.

3. Enterprise In addition to all Professional features; it also includes tools such as Visual basics.

Features of Visual basic

GUI Interface

Modularization

Object Oriented

Debugging

Macros IDE

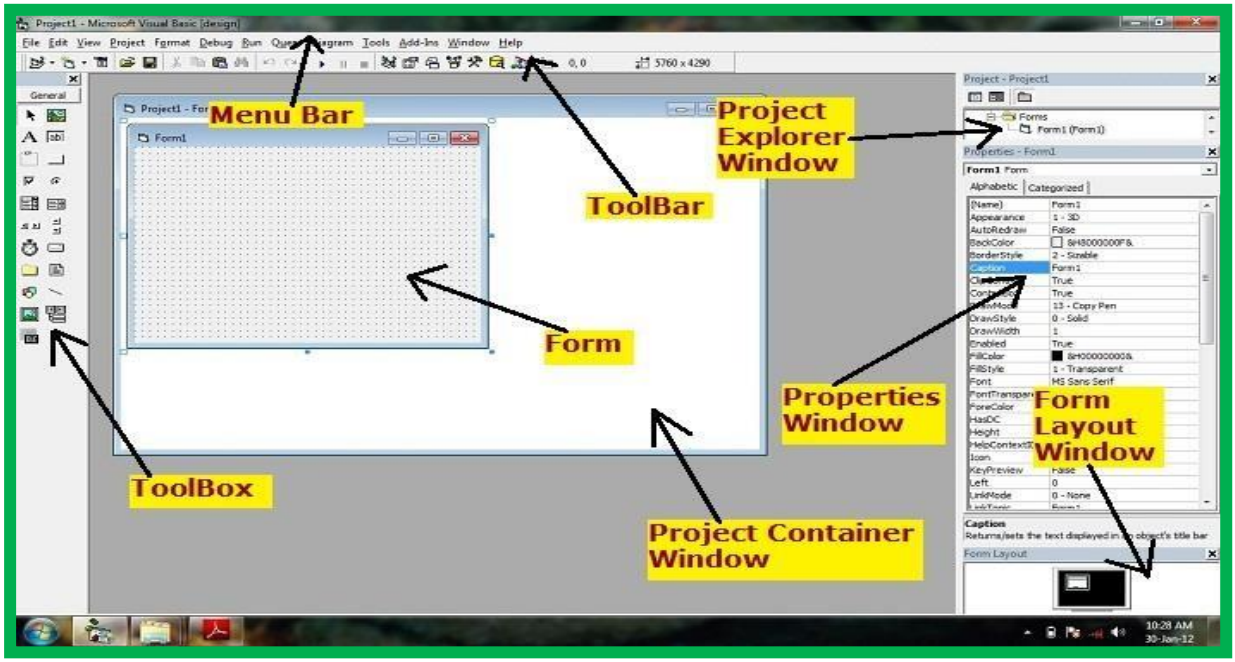
Data access feature

- **GUI Interface**: - VB is a Graphical User Interface language. This means that a VB program will always show something on the screen that the user can interact with to get a job done.

- **Modularization**: - It is considered good programming practice to modularize your programs. Small modules where it is clearly indicated what comes into the module and what goes out makes a program easy to understand.
- **Object Oriented**: - Object Oriented Programming is a concept where the programmer thinks of the program in "objects" that interact with each other. Visual Basic forces this good programming practice.
- **Debugging**: - Visual Basic offers two different options for code debugging: - Debugging Managed Code Runtime Debugger The Debugging Managed Code individually debugs C and C++ applications and Visual Basic Windows applications. The Runtime Debugger helps to find and fix bugs in programs at runtime.
- **Data Access Feature**: - By using data access features, we can create databases, scalable server-side components for most databases, including Microsoft SQL Server and other enterprise-level database.
- **Macros IDE**: - The Macros integrated development environment is similar in design and function to the Visual Studio IDE. The Macros IDE includes a code editor, tool windows, the properties windows and editors.

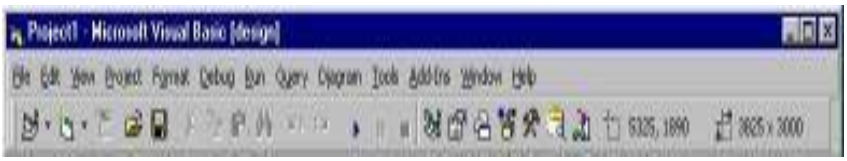
Visual Basic Window Components

The Integrated Development Environment of VB



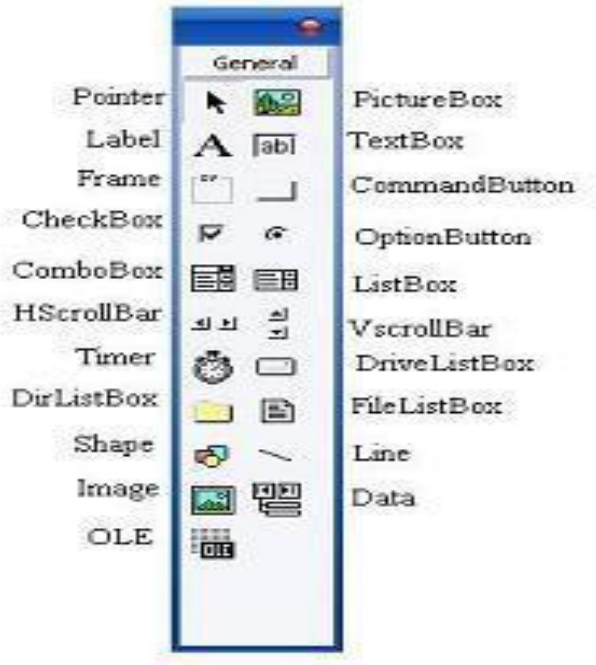
1.Menu Bar

This Menu Bar displays the commands that are required to build an application. The main menu items have sub menu items that can be chosen when needed. The toolbars in the menu bar provide quick access to the commonly used commands and a button in the toolbar is clicked once to carry out the action represented by it.



2.Toolbox

The Toolbox contains a set of controls that are used to place on a Form at design time thereby creating the user interface area. Additional controls can be included in the toolbox by using the Components menu item on the Project menu. A Toolbox is represented in figure 2 shown below.



Toolbox window with its controls available commonly.

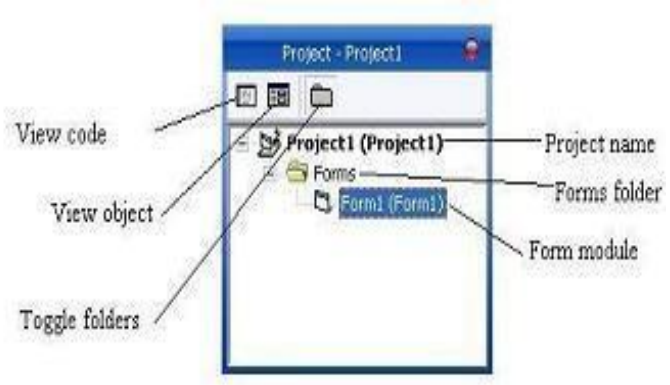
Control	Description

Pointer	Provides a way to move and resize the controls form
Picture Box	Displays icons/bitmaps and metafiles. It displays text or acts as avisual container for other controls.
Textbox	Used to display message and enter text.
Frame	Serves as a visual and functional container for controls
Command Button	Used to carry out the specified action when the user chooses it.
Checkbox	Displays a True/False or Yes/No option.
Option Button	Option Button control which is a part of an option group allows theuser to select only one option even it displays mulitiple choices.
ListBox	Displays a list of items from which a user can select one.
Combo Box	Contains a TextBox and a ListBox. This allows the user to select anietm from the dropdown ListBox, or to type in a selection in the TextBox.
HScrollBar andVScrollBar	These controls allow the user to select a value within the specifiedrange of values
Timer	Executes the timer events at specified intervals of time
DriveListBox	Displays the valid disk drives and allows the user to select one ofthem.
DirListBox	Allows the user to select the directories and paths, which are displayed.
FileListBox	Displays a set of files from which a user can select the desired one.
Shape	Used to add shape (rectangle, square or circle) to a Form
Line	Used to draw straight line to the Form

Image	used to display images such as icons, bitmaps and metafiles. But lesscapability than the PictureBox
Data	Enables the use to connect to an existing database and display information from it.
OLE	Used to link or embed an object, display and manipulate data from other windows-based applications.
Label	Displays a text that the user cannot modify or interact with.

3.Project Explorer Window

Docked on the right side of the screen, just under the tollbar, is the Project Explorer window. The Project Explorer as shown in in figure servres as a quick reference to the various elements of a project namely form, classes *and* modules. All of the object that make up the application are packed in a project. A simple project will typically contain oneform, which is a window that is designed as part of a program's interface. It ispossible to develop any number of forms for use in a program, although a program may consist of a single form. In addition to forms, the Project Explorer window also lists code modules and classes.



4.Properties Window

The Properties Window is docked under the Project Explorer window. The Properties Window exposes the various characteristics of selected objects. Each and every form in an

application is considered an object. Now, each object in Visual Basic has characteristics such as color and size. Other characteristics affect not just the appearance of the object but the way it behaves too. All these characteristics of an object are called its properties. Thus, a form has properties and any controls placed on it will have properties too. All of these properties are displayed in the Properties Window.

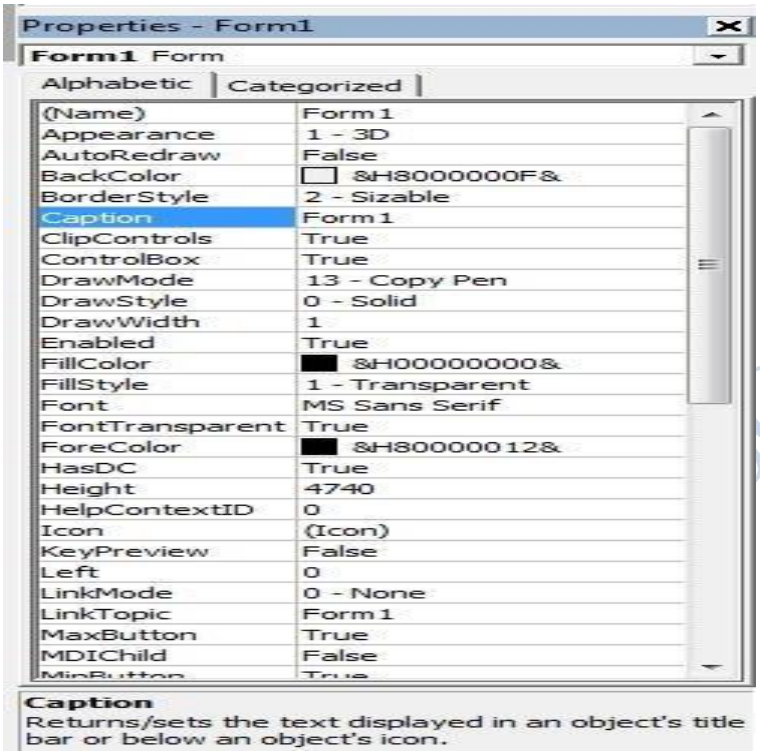


Figure 4. Property Window

5.Object Browser

The Object Browser allows us to browse through the various properties, events and methods that are made available to us. It is accessed by selecting Object Browser from the View menu or pressing the key F2. The left column of the Object Browser lists the objects and classes that are available in the projects that are opened and the controls that have been referenced in them. It is possible for us to scroll through the list and select the object or class that we wish to inspect. After an object is picked up from the Classes list, we can see its members (properties, methods and events) in the right column.

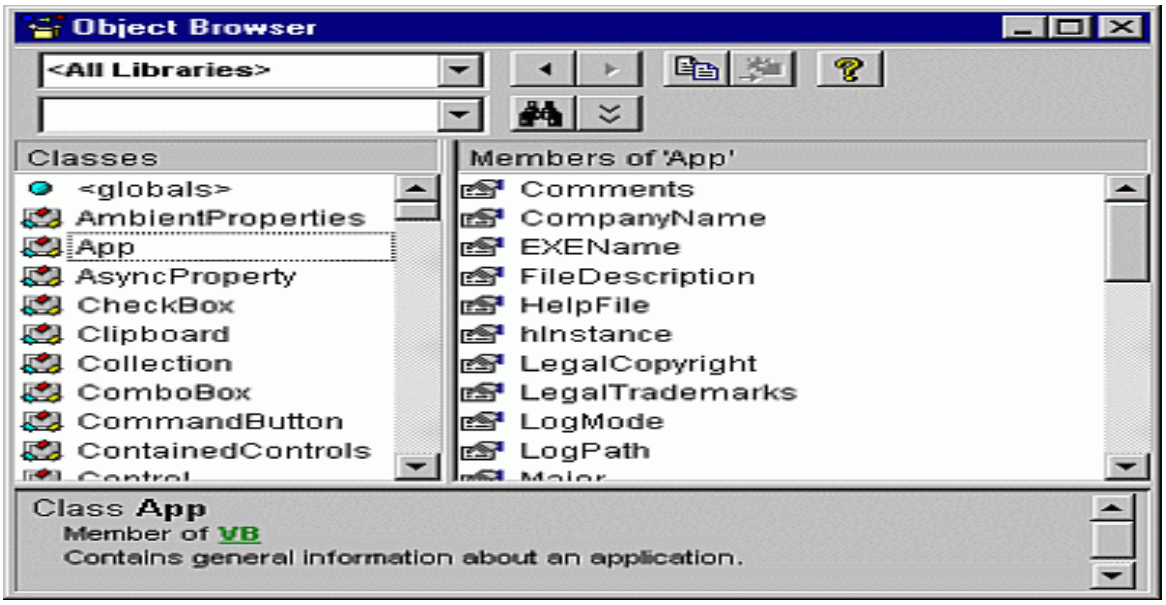


Fig.5. Object browser

6.Code Editor Window

- To open the Code Editor Window, click on view code button from the Project Explorer Window.
- Another way of opening the window is double click on the form or on any object for which you want to add the code.
- It's a word processor specially designed for writing the code for Visual Basics applications.
- Whenever type the name of a function the argument template gets displayed automatically. Again while typing the name of an object or control the drop-down lists displays the properties of that control.

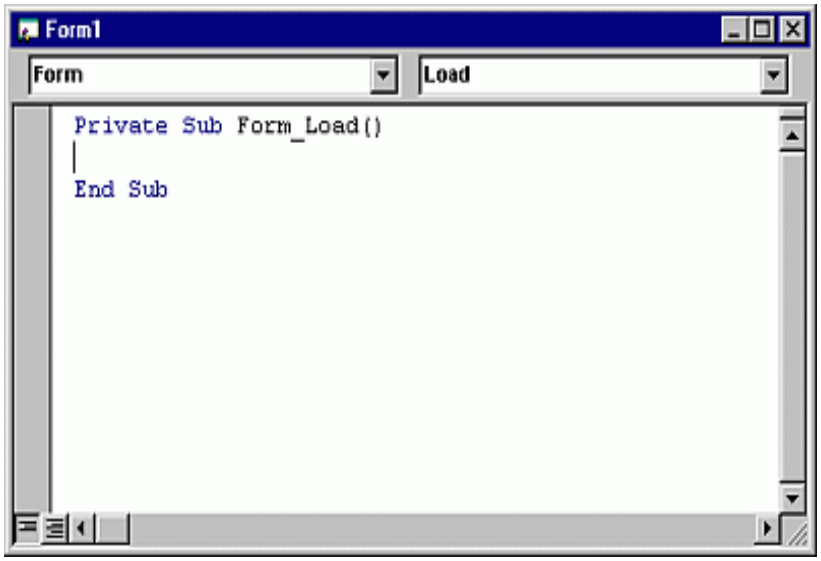


Fig.6 Code Editor Window

7.Standard Toolbar

Standard Toolbar

Fig 7. Standard Toolbar

View Application - Toggles between the host application and the Visual Basic window.

Insert Userform - Defaults to a userform but also allows you to insert modules and procedures.

Save - Saves the application including all the Visual Basic components.

Cut - Removes the selected control or text and places it on the clipboard.

Copy - Copies the selected control or text and places it on the clipboard.

Paste - Inserts the contents of the clipboard at the current location.

Find - Displays the Find dialog box allowing you to search for particular text.

Undo - Reverse the last text editing action.

Redo - Restores the last text editing Undo actions.

Run - Runs the current procedure if the curosr is in a procedure. Runs a userform if a userform is currently displayed. Displays the Run dialog box otherwise ??

Break - Stops execution of a program while it is running and switches into break mode.

Reset - Clears the execution stakc module level variables and resets the project.

Design Mode - Toggles design mode.

Project Explorer - Displays (or toggles) the Project Explorer window.

Properties Window - Displays (or toggles) the Properties window.

Object Browser - Displays (or toggles) the display of the Object Browser window.

Toolbox - Toggles the display of the toolbox that contains userform controls. This is only available when a userform is active.

Help - Opens the Office Assistant.

8. Form Layout Window

- Form Layout window is a simple visual basic design tool whose purpose is to give the user a thumbnail view of the current form. This helps in controlling the form position in the Windows environment. When a more number of forms are there in a particular program, this "Form Layout" window is helpful to arrange all the forms onscreen, in exactly the way in which the user needs them.
- By simply dragging the miniature form to a particular location, the position of a form can be set in the "Form Layout" window.

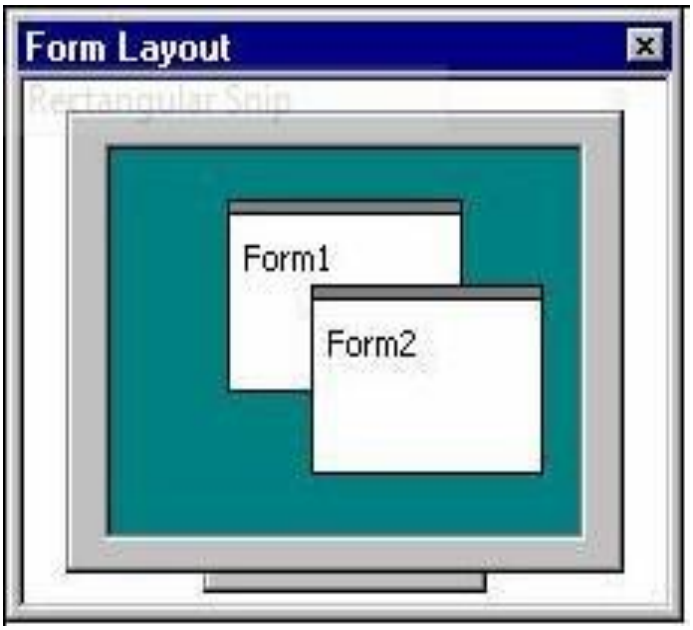


Fig.8 Form Layout Window

Working with Forms:

The main characteristic of a Form is the title bar on which the Form's caption is displayed. On the left end of the title bar is the Control Menu icon. Clicking this icon opens the Control Menu. Maximize, Minimize and Close buttons can be found on the right side of the Form. Clicking on these buttons performs the associated function.

The control menu contains the following commands:

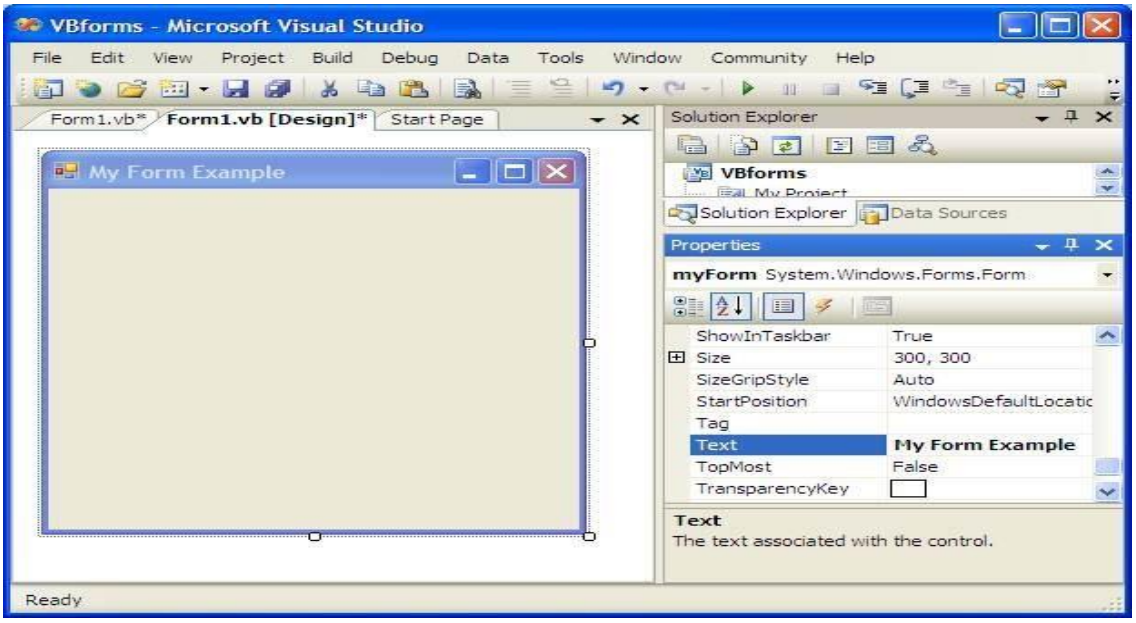
- Restore: Restores a maximized Form to the size it was before it was maximized; available only if the Form has been maximized.
- Move: Lets the user moves the Form around with the mouse
- Size: Lets the user resizes the control with the mouse
- Minimize: Minimizes the Form
- Maximize: Maximizes the Form
- Close: Closes the Form

Extension & with Function of the File

While save a Standard EXE projects many files are created. The Form is saved in an .FRM extension file. Again the project is saved in a. VBP extension file. The extension and functions of the associated file.

Extension	Function of the file
.FRM	Each Forms of the projects are saved with .FRM extension. It stores the details information of the form, controls used in the form and the code have added for the form.
.VBP	To save the project after saving the forms and modules etc. It stores the information about the project type, project type, project properties and the names of the forms present in the application in .VBP extension file.
Form Modules -.FRM Standard Modules -. BAS Class Modules - .CLS	Modules are created to share the code with all the forms associated with project. For examples you are working with many forms and they use common code. So instead of repeating, the code modules are created to share the code.
.EXE	After creating an application you need to create an executable file so that end user can view the application after opening the executable file or you can say the .EXE file .
.FRX	The. FRX file contains the binary information and graphics information of the Form. Whenever you set values for graph properties or controls ofthe form an .FRX file is created automatically.

Properties, Events and Methods of the Form:



Properties describe objects. Methods cause an object to do something. Events are what happens when an object does something.

Every object, such as a form or control, has a set of properties that describe it.

Another important property to consider is Border Style, which determines the window elements (title bar, Maximize and Minimize buttons, and so forth) a form will have.
The Table shows the list of properties of the Form.

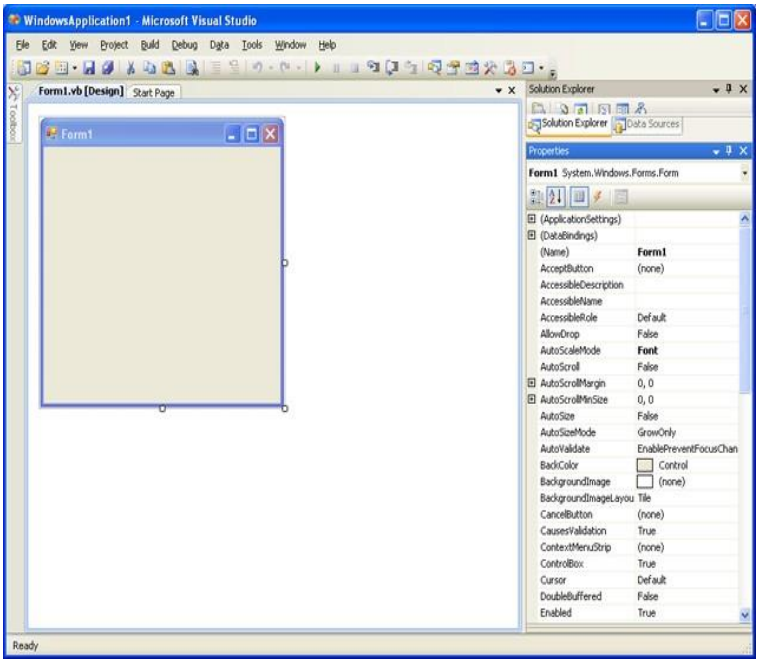
Active Control	Active Form	Appearance	Auto Redraw
Back Color	Border Style	Caption	Clip Controls
Control Box	Controls	Count	CurrentX
CurrentY	Draw Mode	Draw Style	Draw Width
Enabled	Fill Color	Fill Style	Font
Font Bold	Font Italic	Font Name	Font Size
FontStrikethru	Font Transparent	Font Underline	Forecolor
HDC	Height	HelpContexID	HWnd
Icon	Image	Key Preview	Left
Link Mode	Link Topic	MaxButton	MDIChild
MinButton	Mouse Icon	Mouse Pointer	Movable
Name	Negotiate Menus	Picture	Scale Height
Scale Left	Scale Mode	Scale Top	Scale Width
ShowIn Taskbar	Tag	Top	Visible
WhatsThisButton	WhatsThisHelp	Width	Window State

To view the properties of the form, Scroll the properties Window and will find all the properties listed there. Unfortunately, use only some properties others are rarely used.

The important properties of the form are name, caption, MinButton, MaxButton, Close, Icon, Visible, Window State and startupPosition.

Changing the Name Property

Select the Form by clicking on the Form or on the Title Bar. Actually to view the properties of the objects have to select that object.



- Click on Name from the Properties Window.
- Remove the default name Form1 and type the new name. For Example, set the name as Frmnew.It can set any name according to your choice, but remember set an identical name related with the Form otherwise while doing practically will be confused with your Form name.

Form Controls:

Controls are objects that are contained within a form objects. Each type of control has its own set of properties, methods and events make it suitable for a particular purpose. Some of the control we can use in our application are best suited for entering or displaying text.

1.Standard controls

a. Label control
b. Command button control
c. Text box control
d. Check box control
e. Option Button control
f. Frame control
g. Combobox control
h. Image Control
i. Scroll Control
j. Picture Control
k. Sing Timer control
l. Drivelistbox control
m. Dirlistbox
n. Filelistbox Control

A) Label Control

Generally, label control is used to display information. It's commonly used control, and mainly used for identification of other controls. View the list of properties of Label control. The commonly used properties are Caption, Alignment, Autosize, Back color, Forecolor, Border Style, Back style, Font Name, WordWrap. The prefix of Label control is lbl. While changing the name property write lbl (label name). Add the prefix, which will help to identify the objects easily while writing the code.

Label and Frame controls have a few features in common, so it makes sense to explain them together. First, they're mostly "decorative" controls that contribute to the user interface but are seldom used as programmable objects. In other words, you often place them on the form and arrange their properties as your user interface needs dictate, but you rarely write code to serve their events, generally, or manipulate their properties at run time.

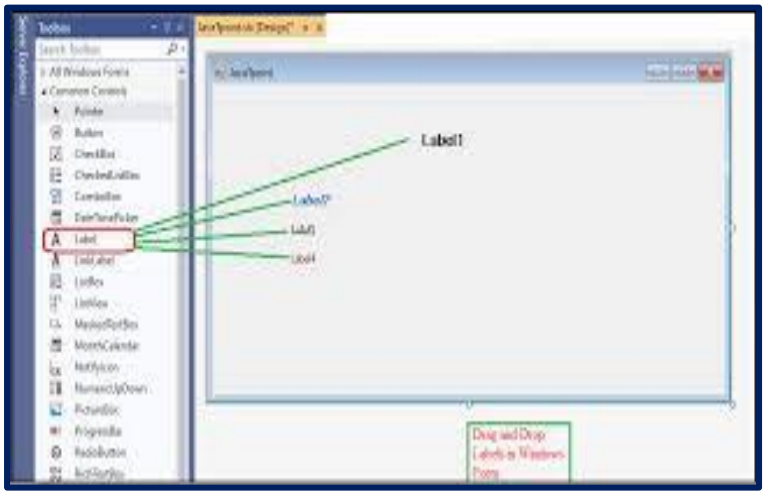
Most people use Label controls to provide a descriptive caption and possibly an associated hot key for other controls, such as TextBox, ListBox, and ComboBox, that don't expose the Caption property. In most cases, you just place a Label control where you need it, set its Caption property to a suitable string (embedding an ampersand character in front of the hot key you want to assign), and you're done. Caption is the default property for Label controls.

Events

Labels are generally used to display information, so events are less triggered. Still some common events like Change (), Click (), Mouse Move () and Mouse Down () are used.

Methods

The list of Properties and Events check the list of methods supported by Label control.



B) Command Button Control

When compared to TextBox controls, these controls are really simple. Not only do they expose relatively few properties, they also support a limited number of events, and you don't usually write much code to manage them. Using Command Button controls is trivial. In most cases, you just draw the control on the form's surface, set its Caption property to a suitable string (adding an & character to associate a hot key with the control if you so choose), and you're finished, at least with user-interface issues. To make the button functional, you write code in its Click event procedure, as in this fragment.

One of the common controls of window is Command Button Control. While working with window have clicked in many dialogs the OK button.

The important and commonly used properties are Name, Caption, Enabled, TabIndex, TabStop, default, font and style. Using the (&) sign can set the hot keys for the CommandButton.So if the caption of the button is OK and to set the hot key as O then set the caption as & OK. Then the user can access the button by pressing Alt+O. Enabled property is frequently used to change the enabled property of the button to true or false. Other than these tabs index is set when are working with many controls and to maintain a hierarchy.

```
Private Sub Command1_Click ()
Unload Me
End Sub
```

You can use two other properties at design time to modify the behavior of a Command Button control. You can set the Default property to True if it's the default push button for the form (the button that receives a click when the user presses the Enter key—usually the OK or Save button). Similarly, you can set the Cancel property to True if you want to associate the button with the Escape key. The only relevant Command Button's run-time property is Value, which sets or returns the state of the control (True if pressed, False otherwise). Value is also the default property for this type of control. In most cases, you don't need to query this property because if you're inside

a button's Click event you can be sure that the button is being activated. The Value property is useful only for programmatically clicking a button.

Events:

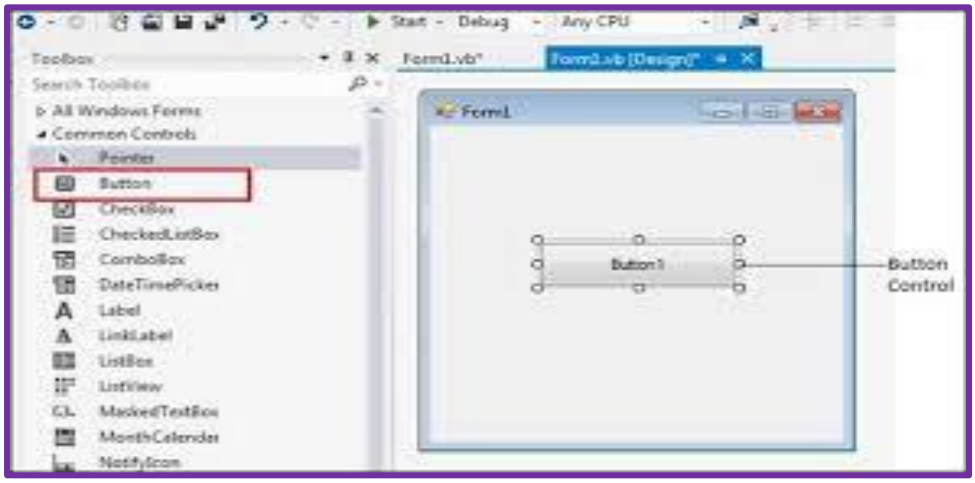
Most of the common event of command Button is Click (). Other than that Mouse Down () and Mouse Up () is also used.

Methods:

Important method of Command Button is Set Focus. Often used to set the Focus on the control. For example, When the users have entered some input in a textbox after pressing tab the focus sets on CommandButton.Command Button control is used in almost in all applications.

Event Supported by Command Button

The CommandButton control supports the usual set of keyboard and mouse events (Key Down, KeyPress, KeyUp, MouseDown, MouseMove, MouseUp, but not the DblClick event) and also the Got Focus and Lost Focus events, but you'll rarely have to write code in the corresponding event procedures.



Properties of a CommandButton control

- To display text on a CommandButton control, set its *Caption* property.
- An event can be activated by clicking on the CommandButton.
- To set the background colour of the CommandButton, select a colour in the BackColor property.
- To set the text colour set the Forecolor property.
- Font for the CommandButton control can be selected using the Font property.
- To enable or disable the buttons set the Enabled property to True or False
- To make visible or invisible the buttons at run time, set the Visible property to True or False.
- Tooltips can be added to a button by setting a text to the Tooltip property of the CommandButton.
- A button click event is handled whenever a command button is clicked. To add a click event handler, double click the button at design time, which adds a subroutine like the one given

below.

Private Sub Command1_Click ()

.....

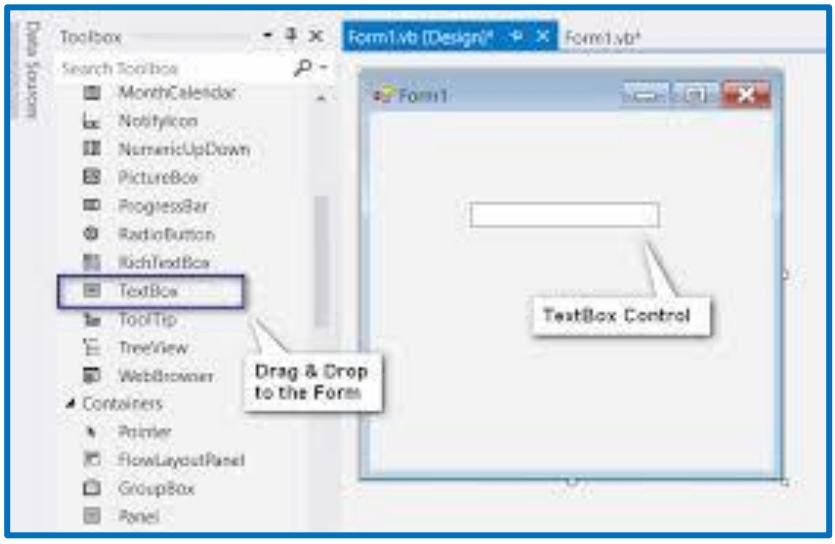
End Sub

C) Using Textbox Control

TextBox controls offer a natural way for users to enter a value in your program. For this reason, they tend to be the most frequently used controls in the majority of Windows applications. TextBox controls, which have a great many properties and events, are also among the most complex intrinsic controls.

Another important and common control is Textbox control mainly used to collect input from the user. In many application data are collected through textbox and according to their input the output is processed.

To work with Textbox control must be familiar with its properties, events and methods. The list of available properties, events and methods.



Text property is used to display text. Password Char will change the character into (*). Mainly used for imputing password. MaxLength will set the maximum length of the field. So, if set it as 10 the user can enter only 10 characters in that field. This property is very useful when creating applications, which handles database. Another commonly used property is Multiline.

Property/ Method	Description
Properties	
Enabled	specifies whether user can interact with this control or not
Index	Specifies the control array index
Locked	If this control is set to True user can use it else if this control is set to false the control cannot be used
MaxLength	Specifies the maximum number of characters to be input. Default value is set to 0 that means user can input any number of characters
MousePointer	Using this we can set the shape of the mouse pointer when over a TextBox
Multiline	By setting this property to True user can have more than one line in the TextBox
PasswordChar	This is to specify mask character to be displayed in the TextBox

ScrollBars	This to set either the vertical scrollbars or horizontal scrollbars to make appear in the TextBox. User can also set it to both vertical and horizontal. This property is used with the Multiline property.
Text	Specifies the text to be displayed in the TextBox at runtime
ToolTipIndex	This is used to display what text is displayed or in the control
Visible	By setting this user can make the Textbox control visible or invisible at runtime
Method	
SetFocus	Transfers focus to the TextBox
Event procedures	
Change	Action happens when the TextBox changes
Click	Action happens when the TextBox is clicked
GotFocus	Action happens when the TextBox receives the active focus
LostFocus	Action happens when the TextBox loses it focus
KeyDown	Called when a key is pressed while the TextBox has the focus
KeyUp	Called when a key is released while the TextBox has the focus

D) Option Button Control:

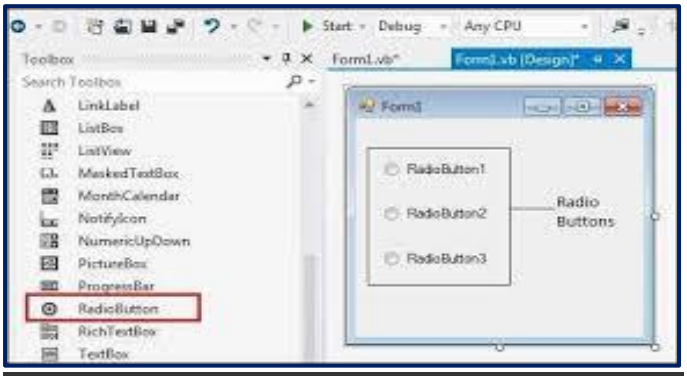
Option Button controls are also known as radio buttons because of their shape. You always use OptionButton controls in a group of two or more because their purpose is to offer a number of mutually exclusive choices. Anytime you click on a button in the group, it switches to a selected state and all the other controls in the group become unselected.

Preliminary operations for an OptionButton control are similar to those already described for Checkbox controls. You set an OptionButton control's Caption property to a meaningful string, and if you want you can change its Alignment property to make the control right aligned. If the control is the one in its group that's in the selected state, you also set its Value property to True. (The Option Button's Value property is a Boolean value because only two states are possible.) Value is the default property for this control. At run time, you typically query the control's Value property to learn which button in its group has been selected. Let's say you have three Option Button controls, named opt Weekly, opt Monthly, and opt Yearly. You can test which one has been selected by the user as follows:

```
If optWeekly.Value = True Then
' User prefers weekly frequency.
ElseIf optMonthly.Value Then
' User prefers monthly frequency.
ElseIf optYearly.Value Then
' User prefers yearly frequency
. End If
```

Strictly speaking, you can avoid the test for the last OptionButton control in its group because all choices are supposed to be mutually exclusive. But the approach I just showed you increases the code's readability.

A group of OptionButton controls is often hosted in a Frame control. This is necessary when there are other groups of OptionButton controls on the form. As far as Visual Basic is concerned, all the OptionButton controls on a form's surface belong to the same group of mutually exclusive selections, even if the controls are placed at the opposite corners of the window. The only way to tell Visual Basic which controls belong to which group is by gathering them inside a Frame control. Actually, you can group your controls within any control that can work as a container PictureBox, for example—but Frame controls are often the most reasonable choice.



Design the Form as per the following specifications table.

Object	Property	Settings
Label	Caption Name	Enter a Number Label1
TextBox	Text Name	(empty) Text1
CommandButton	Caption Name	&Close Command1
OptionButton	Caption Name	&Octal optOct
OptionButton	Caption Name	&Hexadecimal optHex
OptionButton	Caption Name	&Decimal optDec

The application responds to the following events

- The change event of the TextBox reads the value and stores it in a form-level numeric variable.
- The click event of optOct button returns curretval in octal.

- The click event of the optHex button curerntval in hexadecimal
- The click event of the optDec button returns the decimal equivalent of the value held currentval.

The Val function is used to translate string to a number and can recognize Octal and Hexadecimal strings. The LTrim function trims the leading blanks in the text. The following code is entered in the click events of the OptionButton controls.

```
Private Sub optOct_Click ()
Text1.Text = Oct(currentval)
End Sub
Private Sub optHex_Click ()
Text1.Text = Hex(currentval)
End Sub
Private Sub optDec_Click ()
Text1.Text = Format(currentval)
End Sub
```

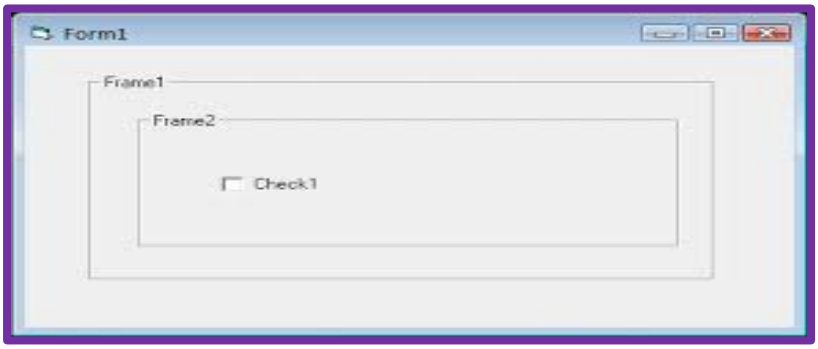
The following code is entered in the click event of the Close button.

```
Private Sub cmd Close_Click ()
Unlod Me
End Sub
```

Frame Control

Frame controls are similar to Label controls in that they can serve as captions for those controls that don't have their own. Moreover, Frame controls can also (and often do) behave as containers and host other controls. In most cases, you only need to drop a Frame control on a form and set its Caption property. If you want to create a borderless frame, you can set its Border Style property to 0-None.

Frame helps in grouping. Normally place Option Button and Checkbox inside the Frame. Another feature of the Frame is, the controls placed inside the Frame moves along with it when shifted. So, add the Frame first and then Option buttons are added.



Controls that are contained in the Frame control are said to be child controls. Moving a control at design time over a Frame control—or over any other container, for that matter—doesn't automatically make that control a child of the Frame control. After you create a Frame control, you can create a child control by selecting the child control's icon in the Toolbox and drawing a new instance inside the Frame's border. Alternatively, to make an existing control a child of a Frame control, you must select the control, press Ctrl+X to cut it to the Clipboard, select the Frame control, and press Ctrl+V to paste the control inside the Frame. If you don't follow this procedure and you simply move the control over the Frame, the two controls remain completely independent of each other, even if the other control appears in front of the Frame control.

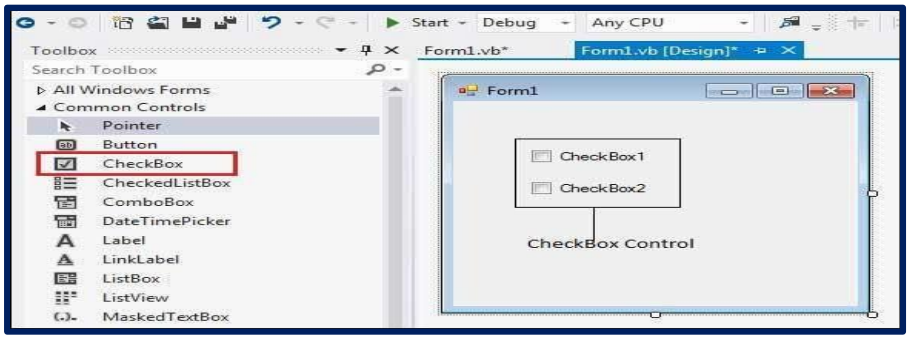
Frame controls, like all container controls, have two interesting features. If you move a Frame control, all the child controls go with it. If you make a container control disabled or invisible, all its child controls also become disabled or invisible. You can exploit these features to quickly change the state of a group of related controls.

Generally, Name property is used frequently for all controls, but except that Caption is quite important for Frame control.

Events are hardly used. Still Mouse Move (), Drag Drop () are quite common. When to drag and drop an object in the frame this event occurs.

Checkbox control

CheckBox is a common control for windows. To select more than one option CheckBox is used in the application. To set the options in the Checkbox, so that they can select multiple choices. Before build application check the list of available properties, events and methods of CheckBox control. CheckBox control is similar to the option button, except that a list of choices can be made using check boxes where you cannot choose more than one selection using an Option Button. By ticking the CheckBox, the value is set to True. This control can also be grayed when the state of the CheckBox is unavailable, but you must manage that state through code.



When you place a Checkbox control on a form, all you have to do, usually, is set its Caption property to a descriptive string. You might sometimes want to move the little check box to the right of its caption, which you do by setting the Alignment property to 1-Right Justify, but in most cases

the default setting is OK. If you want to display the control in a checked state, you set its Value property to 1-Checked right in the Properties window, and you set a grayed state with 2-Grayed.

The only important event for CheckBox controls is the Click event, which fires when either the user or the code changes the state of the control. In many cases, you don't need to write code to handle this event. Instead, you just query the control's Value property when your code needs to process user choices. You usually write code in a CheckBox control's Click event when it affects the state of other controls. For example, if the user clears a check box, you might need to disable one or more controls on the form and reenable them when the user clicks on the check box again. This is how you usually do it (here I grouped all the relevant controls in one frame named Frame1)

```
Private Sub Check1_Click ()
Frame1.Enabled = (Check1.Value = vbChecked)
End Sub
```

Note that Value is the default property for CheckBox controls, so you can omit it in code. I suggest that you not do that, however, because it would reduce the readability of your code.

The following example illustrates the use of CheckBox control

* Open a new Project and save the Form as CheckBox.frm and save the Project as CheckBox.vbp

Object	Property	Setting
Form	Caption Name	CheckBox frmCheckBox
CheckBox	Caption Name	Bold chkBold
CheckBox	Caption Name	Italic chkItalic
CheckBox	Caption Name	Underline chkUnderline
OptionButton	Caption Name	Red optRed
OptionButton	Caption Name	Blue optBlue
OptionButton	Caption Name	Green optGreen
TextBox	Name Text	txtDisplay (empty)

CommandButton	Caption Name	Exit cmdExit

Following code is typed in the Click () events of the Checkboxes

```
Private Sub chkBold_Click ()
If chkBold.Value = 1 Then
  txtDisplay.FontBold = True
Else
  txtDisplay.FontBold = False
End If
End Sub

Private Sub chkItalic_Click ()
If chkItalic.Value = 1 Then
  txtDisplay.FontItalic = True
Else
  txtDisplay.FontItalic = False
End If
End Sub

Private Sub chkUnderline_Click ()
If chkUnderline.Value = 1 Then
  txtDisplay.FontUnderline = True
Else
  txtDisplay.FontUnderline = False
End If
End Sub

Private Sub optBlue_Click ()
  txtDisplay.ForeColor = vbBlue
End Sub

Private Sub optRed_Click ()
  txtDisplay.ForeColor = vbRed
End Sub
```

```
Private Sub optGreen_Click ()
  txtDisplay.ForeColor = vbGreen
End Sub
```

List box Control:

Generally, ListBox is used to show the list of data. User can select the data from the list and according to their selection the data can be processed. A scroll bar is added automatically if the list of data added is more than the control size i.e. the ListBox size. The user can use the scroll bar to view the data of the list. This control presents a set of choices that are displayed vertically in a column. If the number of items exceed the value that be displayed, scroll bars will automatically appear on the control. These scroll bars can be scrolled up and down or left to right through the list.

The common properties are List, List Index, Name, Multiselect, Style, Sorted and Selected. List property sets or return the value of the list. Multiselect is used to select multiple options from the list.

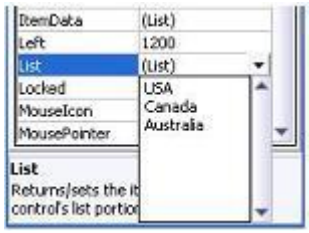
The frequently used event of the list is the DblClick () event. Perhaps might have selected data by double clicking on the ListBox on any Window Application.

Add Item method is most common methods that adds the item in the list. Again, the Refresh method is used to refresh the list.

Adding items to a List

It is possible to populate the list at design time or run time

Design Time: To add items to a list at design time, click on List property in the property box and then add the items. Press CTRL+ENTER after adding each item as shown below.



Run Time: The AddItem method is used to add items to a list at run time. The AddItem method uses the following syntax.

Object.AddItemitem, Index

The **item** argument is a string that represents the text to add to the list

The **index** argument is an integer that indicates where in the list to add the new item. Not giving the index is not a problem, because by default the index is assigned.

Following is an example to add item to a combo box. The code is typed in the Form Load event

```
Private Sub Form_Load ()
```

```
Combo1.AddItem 1
Combo1.AddItem 2
Combo1.AddItem 3
Combo1.AddItem 4
Combo1.AddItem 5
Combo1.AddItem 6
End Sub
```

Removing Items from a List

The RemoveItem method is used to remove an item from a list. The syntax for this is given below.

Object. RemoveItem index

The following code verifies that an item is selected in the list and then removes the selected item from the list.

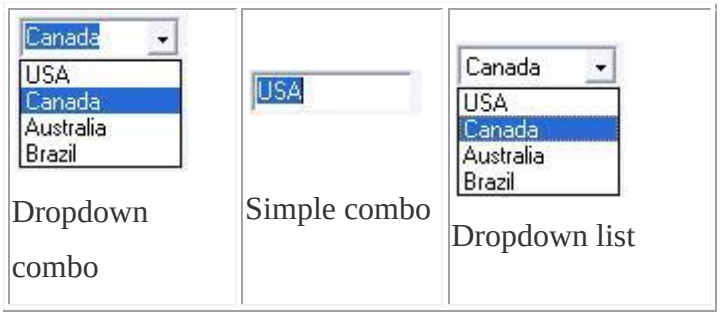
```
Private Sub cmdRemove_Click()
If List1.ListIndex > -1 Then
List1.RemoveItem List1. ListIndex
End If
End Sub
```

Sorting the List

The Sorted property is set to True to enable a list to appear in alphanumeric order and False to display the list items in the order which they are added to the list.

Using Combo Box Control:

A ComboBox combines the features of a TextBox and a ListBox. This enables the user to select either by typing text into the ComboBox or by selecting an item from the list. There are three types of ComboBox styles that are represented as shown below.



- Types of Combo
    1. Dropdown Combo (style 0)
    2. Simple Combo (style 1)
    3. Dropdown List (style 2)

The Simple Combo box displays an edit area with an attached list box always visible immediately below the edit area. A simple combo box displays the contents of its list all the time. The user can select an item from the list or type an item in the edit box portion of the combo box. A scroll bar is displayed beside the list if there are too many items to be displayed in the list box

area. The Dropdown Combo box first appears as only an edit area with a down arrow button at the right. The list portion stays hidden until the user clicks the down-arrow button to drop down the list portion. The user can either select a value from the list or type a value in the edit area.

The Dropdown list combo box turns the combo box into a Dropdown list box. At run time, the control looks like the Dropdown combo box. The user could click the down arrow to view the list. The difference between Dropdown combo & Dropdown list combo is that the edit area in the Dropdown list combo is disabled. The user can only select an item and cannot type anything in the edit area. Anyway, this area displays the selected item.

The style property is important because according to its selection the behavior and appearance of the ComboBox is determined. Other properties like List, ListIndex, Sorted and List Count are often used.

Common events of Combo Box are Change, Click, DblClick and Dropdown.

AddItem, RemoveItem and the Clear methods are often used while creating application.

Image Control:

Image control is used to handle image. The control can display .BMP,.ICO,.JPEG(.JPG),.GIF and .WMF files. It contains less memory than picture control, so for displaying graphics this is an ideal control. Initially check the properties, events and methods of the Image Control. Name, Picture, stretch and Visible properties are used often. Picture property is used to load the picture. Set the stretch property as True and the picture gets resized with the control. Again, while creating animation program visible property is often used.

Image controls are far less complex than Picture Box controls. They don't support graphical methods or the Auto Redraw and the Clip Controls properties, and they can't work as containers, just to hint at their biggest limitations. Nevertheless, you should always strive to use Image controls instead of Picture Box controls because they load faster and consume less memory and system resources. Remember that Image controls are windowless objects that are actually managed by Visual Basic without creating a Windows object. Image controls can load bitmaps and JPEG and GIF images.

When you're working with an Image control, you typically load a bitmap into its Picture property either at design time or at run time using the Load Picture function. Image controls don't expose the AutoSize property because by default they resize to display the contained image (as it happens with PictureBox controls set at AutoSize = True). On the other hand, Image controls support a Stretch property that, if True, resizes the image (distorting it if necessary) to fit the control. In a sense, the Stretch property somewhat remedies the lack of the Paint Picture method for this control. In fact, you can zoom in to or reduce an image by loading it in an Image control and then setting its Stretch property to True to change its width and height:

' Load a bitmap.

Image1.Stretch = False

Image1.Picture = Load Picture("c:\windows\setup.bmp")

' Reduce it by a factor of two.

Image1.Stretch = True

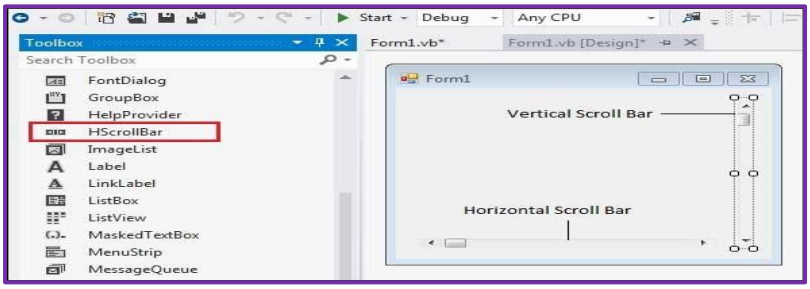
Image1.Move 0, 0, Image1.Width / 2, Image1.Width / 2

Image controls support all the usual mouse events. For this reason, many Visual Basic developers have used Image controls to simulate graphical buttons and toolbars. Now that Visual Basic natively supports these controls, you'd probably better use Image controls only for what they were originally intended.

Scroll Control:

It is a commonly used control, which enables the user to select a value by positioning it at the desired location. It represents a set of values. The Min and Max property represents the minimum and maximum value. The value **property** of the ScrollBar represents its current value, that may be any integer between minimum and maximum values assigned.

The HScrollBar and the VScrollBar controls are perfectly identical, apart from their different orientation. After you place an instance of such a control on a form, you have to worry about only a **few properties:** Min and Max represent the valid range of values, Small Change is the variation in value you get when clicking on the scroll bar's arrows, and LargeChange is the variation you get when you click on either side of the scroll bar indicator. The default initial value for those two properties is 1, but you'll probably have to change LargeChange to a higher value. For example, if you have a scroll bar that lets you browse a portion of text, SmallChange should be 1 (you scroll one line at a time) and LargeChange should be set to match the number of visible text lines in the window.



The most important run-time property is Value, which always returns the relative position of the indicator on the scroll bar. By default, the Min value corresponds to the leftmost or upper end of the control:

' Move the indicator near the top (or left) arrow.

VScroll1.Value = VScroll1.Min

' Move the indicator near the bottom (or right) arrow.

VScroll1.Value = VScroll1.Max

While this setting is almost always OK for horizontal scroll bars, you might sometimes need to reverse the behavior of vertical scroll bars so that the zero is near the bottom of your form. This arrangement is often desirable if you want to use a vertical scroll bar as a sort of slider. You

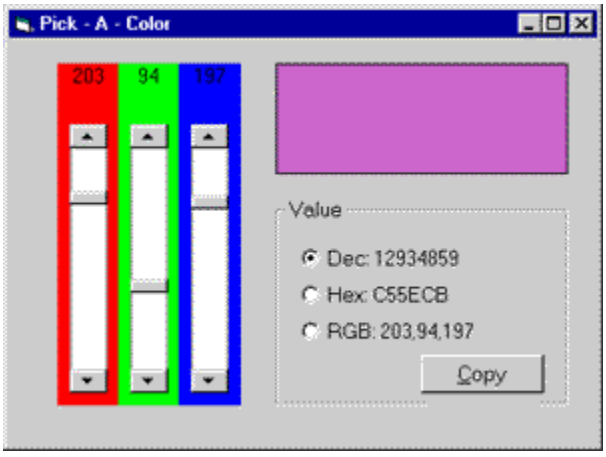
obtain this behavior by simply inverting the values in the Min and Max properties. (In other words, it's perfectly legal for Min to be greater than Max.)

There are two key events for scrollbar controls: The Change event fires when you click on the scroll bar arrows or when you drag the indicator; the Scroll event fires while you drag the indicator. The reason for these two distinct possibilities is mostly historical. First versions of Visual Basic supported only the Change event, and when developers realized that it wasn't possible to have continuous feedback when users dragged the indicator, Microsoft engineers added a new event instead of extending the Change event. In this way, old applications could be recompiled without unexpected changes in their behavior. At any rate, this means that you must often trap two distinct events:

' Show the current scroll bar's value.

```
Private VScroll1_Change ()
Label1.Caption = VScroll1.Value
End Sub
Private VScroll1_Scroll ()
Label1.Caption = VScroll1.Value
End Sub
```

The example shown in the following figure uses three VScrollBar controls as sliders to control the individual RGB (red, green, blue) components of a color. The three scroll bars have their Min property set to 255 and their Max property set to 0, while their SmallChange is 1 and LargeChange is 16. This example is also a moderately useful program in itself because you can select a color and then copy its numeric value to the clipboard and paste it in your application's code as a decimal value, a hexadecimal value, or an RGB function.



Picture Control:

The Picture Box control is used for displaying images on the form. The Image property of the control allows you to set an image both at design time or at run time. Let's create a picture box by dragging a Picture Box control from the Toolbox and dropping it on the form.

PictureBox controls are among the most powerful and complex items in the Visual Basic Toolbox window. In a sense, these controls are more similar to forms than to other controls. For example, PictureBox controls support all the properties related to graphic output, including AutoRedraw, ClipControls,FontTransparent, CurrentX, CurrentY, and all the Drawxxxx, Fillxxxx, and Scalexxxx properties. PictureBox controls also support all graphic methods, such as Cls, PSet, Point, Line, and Circle and conversion methods, such as ScaleX, ScaleY, Text Width, and Text Height. In other words, all the techniques that I described for forms can also be used for PictureBox controls (and therefore won't be covered again in this section).

Loading images

{Once you place a Picture Box on a form, you might want to load an image in it, which you do by setting the Picture property in the Properties window. You can load images in many different graphic formats, including bitmaps (BMP), device independent bitmaps (DIB), metafiles (WMF), enhanced metafiles (EMF), GIF and JPEG compressed files, and icons (ICO and CUR). You can decide whether a control should display a border, resetting the BorderStyle to 0-None if necessary. Another property that comes handy in this phase is AutoSize: Set it to True and let the control automatically resize itself to fit the assigned image.}

You might want to set the Align property of a Picture Box control to something other than the 0-None value. By doing that, you attach the control to one of the four form borders and have Visual Basic automatically move and resize the Picture Box control when the form is resized. Picture Box controls expose a Resize event, so you can trap it if you need to move and resize its child controls too. You can do more interesting things at run time. To begin with, you can programmatically load any image in the control using the Load Picture function:

Picture1.Picture = LoadPicture("c:\windows\setup.bmp")

And you can clear the current image using either one of the following statements:

Picture1.Picture = LoadPicture ("")

Set Picture1.Picture = Nothing

The LoadPicture function has been extended in Visual Basic 6 to support icon files containing multiple icons. The new syntax is the following: LoadPicture (filename, [size], [colordepth], [x],[y])

{where values in square brackets are optional. If filename is an icon file, you can select a particular icon using the size or color depth arguments. Valid sizes are 0-vbLPSmall, 1-vbLPLarge (system icons whose sizes depend on the video driver), 2-vbLPSmallShell, 3-vbLPLargeShell (shell icons whose dimensions are affected by the Caption Button property as set in the Appearance tab in the screen's Properties dialog box), and 4-vbLPCustom (size is determined by x and y). Valid color depths are 0-vbLPDefault (the icon in the file that best matches current screen settings), 1-vbLPMonochrome, 2-vbLPVGAColor (16 colors), and 3-vbLPColor (256 colors).}

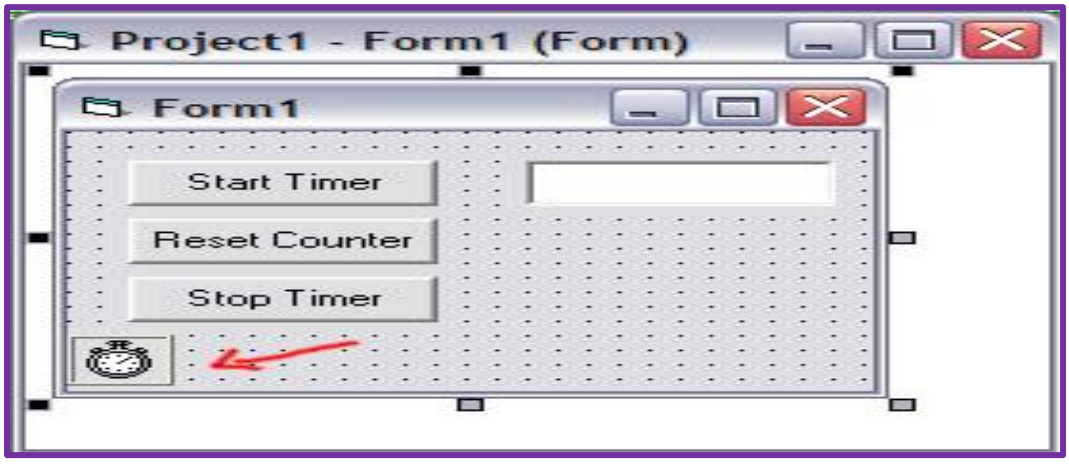
You can copy an image from one PictureBox control to another by assigning the target control's Picture property:

Picture2.Picture = Picture1.Picture

Sing Timer Control

Timer control is quite helpful. With this control can create programs/applications, which work with time limit. This control is not visible at run time and so no problem in placing the control in the Form.

A Timer control is invisible at run time, and its purpose is to send a periodic pulse to the current application. You can trap this pulse by writing code in the Timer's Timer event procedure and take advantage of it to execute a task in the background or to monitor a user's actions. This control exposes only two meaningful properties: Interval and Enabled. Interval stands for the number of milliseconds between subsequent pulses (Timer events), while Enabled lets you activate or deactivate events.



Name, Enabled and Interval property is mainly used while creating application. Interval property is measured in milliseconds.

Only a single event, Timer. It has no methods.

When you place the Timer control on a form, its Interval is 0, which means no events. Therefore, remember to set this property to a suitable value in the Properties window or in the Form Load event procedure:

```
Private Sub Form_Load()
Timer1.Interval = 500 ' Fire two Timer events per
second.
End Sub
```

File Control

Three of the controls on the Toolbox let you access the computer's file system. They are DriveListBox, DirListBox and FileListBox controls (see below figure) , which are the basic blocks

for building dialog boxes that display the host computer's file system. Using these controls, user can traverse the host computer's file system, locate any folder or files on any hard disk, even on network drives. The files are controls are independent of one another, and each can exist on its own, but they are rarely used separately. The files controls are described next.

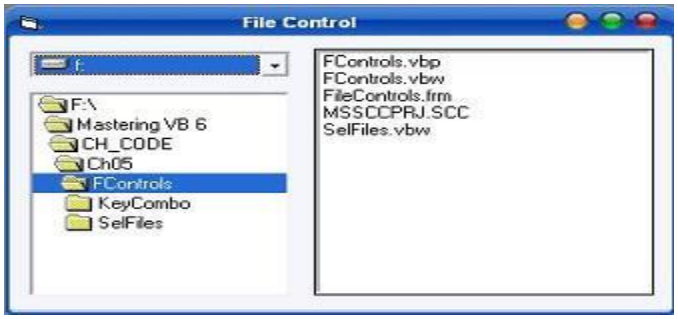
In a nutshell, the DriveListBox control is a combo box-like control that's automatically filled with your drive's letters and volume labels. The DirListBox is a special list box that displays a directory tree. The FileListBox control is a special-purpose ListBox control that displays all the files in a given directory, optionally filtering them based on their names, extensions, and attributes.

These controls often work together on the same form; when the user selects a drive in a DriveListBox, the DirListBox control is updated to show the directory tree on that drive. When the user selects a path in the DirListBox control, the FileListBox control is filled with the list of files in that directory. These actions don't happen automatically, however—you must write code to get the job done.

After you place a DriveListBox and a DirListBox control on a form's surface, you usually don't have to set any of their properties; in fact, these controls don't expose any special property, not in the Properties window at least. The FileListBox control, on the other hand, exposes one property that you can set at design time—the Pattern property. This property indicates which files are to be shown in the list area: Its default value is *.* (all files), but you can enter whatever specification you need, and you can also enter multiple specifications using the semicolon as a separator. You can also set this property at run time, as in the following line of code:

File1.Pattern = "*.txt; *.doc; *.rtf"

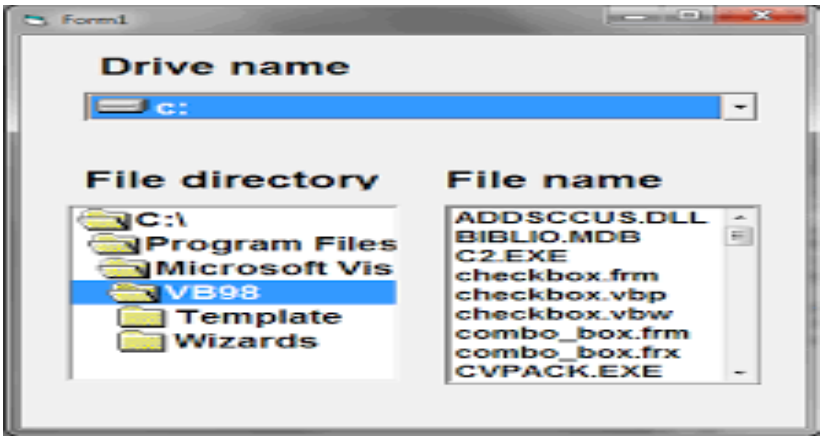
Following figure shows three files controls are used in the design of Forms that let users explore the entire structure of their hard disks.



- **DriveListBox** : Displays the names of the drives within and connected to the PC. The basic property of this control is the drive property, which set the drive to be initially selected in the control or returns the user's selection. Generally, DriveListBox displays the available drives. Actually, it works with DirListBox control and FileListBox control. When the user selects the drive from the DriveListBox the directories are displayed in the DirListBox and when the directory is selected from the DirListBox then the files are displayed in the FileListBox. This process is interrelated between these controls.

In general Name and drive property is frequently used.

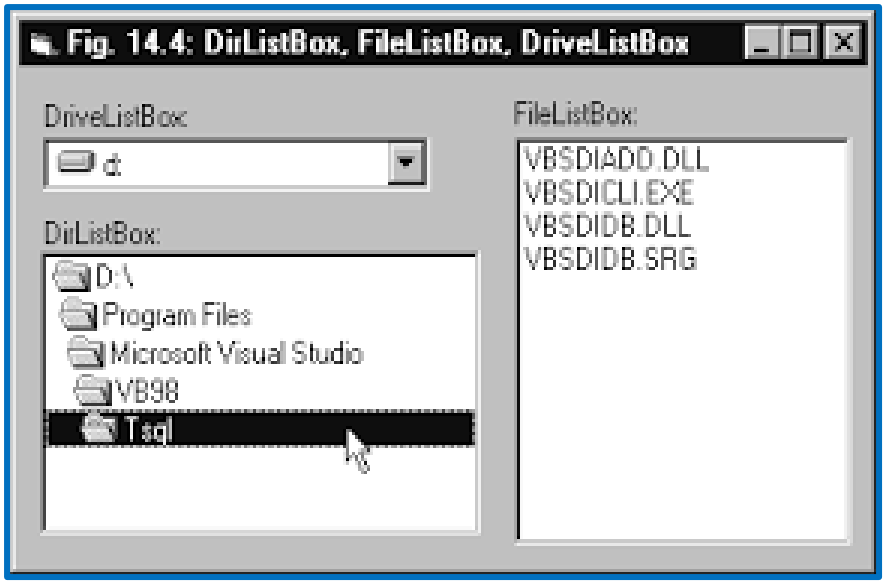
Change () event is one of important events for DriveListBox. By changing the drive users are triggering to Change () event.



- **DirListBox:** Displays the folders of current Drive. The basic property of this control is the Path property, which is the name of the folder whose sub folders are displayed in the control. To select the directory DirListBox is used. While changing the drives Change () event is used often.
- **FileListBox**: Displays the files of the current folder. The basic property of this control is also called Path, and it's the path name of the folder whose files are displayed.

After the Name property Path property is important which displays the path of the file.

The important and frequently used events are Click (), DblClick (), PathChange () and Pattern Change ().



The three File controls are not tied to one another. If you place all three of them on a Form, you will see the names of all the folders under the current folder, and so on. Each time you select a folder in the DirlistBox by double clicking its name, its sub folders are displayed. Similarly, the FileListBox control will display the names of all files in the current folder. Selecting a drive in the DriveListBox control, however this doesn't affect the contents of the DirListBox.

To connect to the File controls, you must assign the appropriate values to the properties. To compel the DirListBox to display the folders of the selected drive in the DriveListBox, you must make sure that each time the user selects another drive, the Path property of the DirListBox control matches the Drive property of the DriveListBox. After these preliminary steps, you're ready to set in motion the chain of events. When the user selects a new drive in the DriveListBox control, it fires a Change event and returns the drive letter (and volume label) in its Drive property. You trap this event and set the DirListBox control's Path property to point to the root directory of the selected drive:

```
Private Sub Drive1_Change ()
Dir1.Path = Left$(Drive1.Drive, 1) & ":\"
End Sub
```

When the user double-clicks on a directory name, the DirListBox control raises a Change event; you trap this event to set the FileListBox's Path property accordingly:

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

Finally, when the user clicks on a file in the FileListBox control, a Click event is fired (as if it were a regular ListBox control), and you can query its Filename property to learn which file has been selected. Note how you build the complete path:

```
Filename = File1.Path
If Right$(Filename, 1) <> "\" Then Filename = Filename & "\"
Filename = Filename & File1.Filename
```

The DirListBox and FileListBox controls support most of the properties typical of the control they derive from—the List Box control—including the List Count and the List Index properties and the Scroll event. The FileListBox control supports multiple selection; hence you can set its MultiSelect property in the Properties window and query the SelCount and Selected properties at run time.

The FileListBox control also exposes a few custom Boolean properties, Normal, Archive, Hidden, Read Only, and System, which permit you to decide whether files with these attributes

should be listed. (By default, the control doesn't display hidden and system files.) This control also supports a couple of custom events, Path Change and Pattern Change, that fire when the corresponding property is changed through code. In most cases, you don't have to worry about them, and I won't provide examples of their usage.

The problem with the DriveListBox, DirListBox and FileListBox controls is that they're somewhat outdated and aren't used by most commercial applications any longer. Moreover, these controls are known to work incorrectly when listing files on network servers and sometimes even

on local disk drives, especially when long file and directory names are used. For this reason, I discourage you from using them and suggest instead that you use the

Common Dialog controls for your File Open and File Save dialog boxes. But if you need to ask the user for the name of a directory rather than a file, you're out of luck because—while Windows does include such a system dialog box, named BrowseForFolders dialog—Visual Basic still doesn't offer a way to display it (unless you do some advanced API programming). Fortunately, Visual Basic 6 comes with a new control—the Image Combo control—that lets you simulate the appearance of the DriveListBox control. It also offers you a powerful library—the Filesystem Object library—that completely frees you from using these three controls, if only as hidden controls that you use just for quickly retrieving information on the file system.