MAR GREGORIOS COLLEGE OF ARTS & SCIENCE

Block No.8, College Road, Mogappair West, Chennai – 37

Affiliated to the University of Madras
Approved by the Government of Tamil Nadu
An ISO 9001:2015 Certified Institution

DEPARTMENT OF COMPUTER APPLICATION

SUBJECT NAME: VISUAL PROGRAMMING

SUBJECT CODE: SEE5A

SEMESTER: V

PREPARED BY: PROF.M.DAISY RANI

Unit 1: Customizing a Form - Writing Simple Programs - Toolbox - Creating Controls - Name Property - Command Button - Access Keys - Image Controls - Text Boxes - Labels - Message Boxes - Grid - Editing Tools - Variables - Data Types - String - Numbers.

Unit-2: Displaying Information - Determinate Loops - Indeterminate Loops - Conditionals - Built-in Functions - Functions and Procedures.

Unit 3: Lists - Arrays - Sorting and Searching - Records - Control Arrays - Combo Boxes - Grid Control - Projects with Multiple forms - DoEvents and Sub Main - Error Trapping.

Unit-4: VB Objects - Dialog Boxes - Common Controls - Menus - MDI Forms - Testing, Debugging and Optimization - Working with Graphics.

Unit-5 : Monitoring Mouse activity - File Handling - File System Controls - File System Objects - COM/OLE - automation - DLL Servers - OLE Drag and Drop.

Unit 1

Customizing a Form

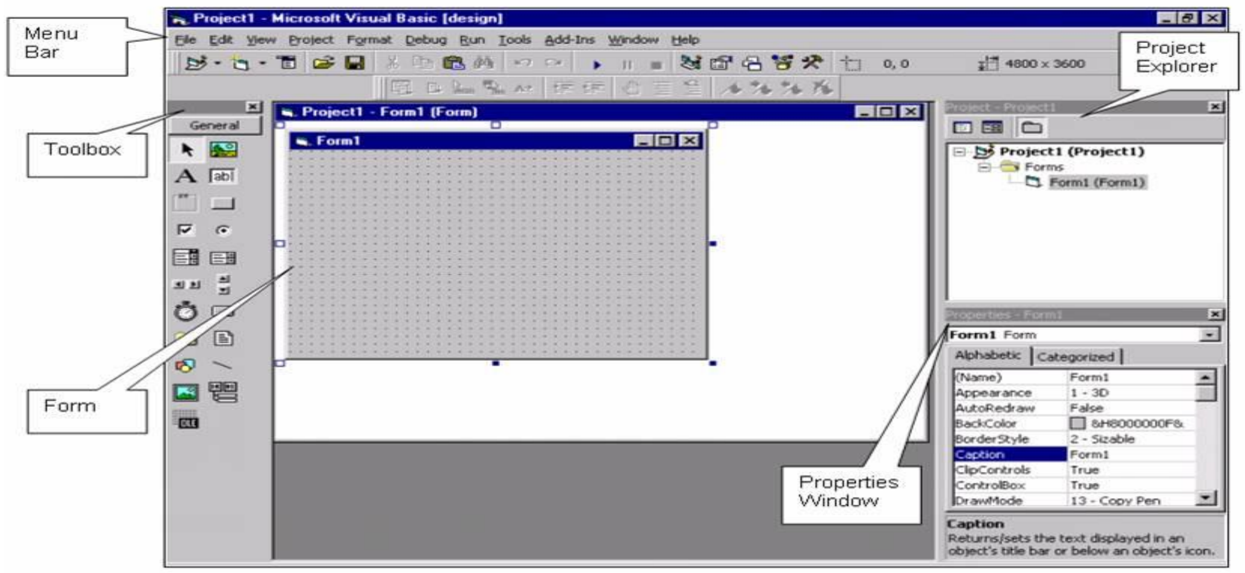
Introduction to Visual Basic 6

Visual Basic, derived from the Basic language, is an object-based and eventdriven programming language from Microsoft. This language is relatively easy to learn. It enables you to create GUI (Graphical user interface) applications easily using the Rapid Application Development (RAD) technique. The one most interesting feature of this language is that it comes with a designer called Integrated Development Environment (IDE). The easy-to-use tools of the IDE enable you to easily create buttons, textbox, and other controls for your desktop application.

Visual Basic 6.0 is a very powerful programming language. It enables GUI application development, provides access to databases and enables the creation of ActiveX controls. In addition, Visual Basic 6 is Event-driven because we need to write code in order to perform some tasks in response to certain events. The events usually comprises but not limited to the user's inputs.

Some of the events are load, click, double click, drag and drop, pressing the keys and more. We will learn more about events in later lessons. Therefore, a VB6 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.

VB IDE (integrated development environment)



(Name)	Form1
Appearance	1 - 3D
AutoRedraw	False
BackColor	&H8000000F&
BorderStyle	2 - Sizable
Caption	Form1
ClipControls	True
ControlBox	True
DrawMode	13 - Copy Pen

Caption
Returns/sets the text displayed in an object's title bar or below an object's icon.

Title bar:

- Title bar is the horizontal bar located at the top of the screen
- It gives the name of the application
- Interaction between the user and the title bar and handled by windows

- Everything below the title and menu bars in the windows application is called the "client area"

Menu bar:

- Selecting items from the pull-down menus listed on a menu bar.
- The menu bar gives the tools needed to develop, test and save the application.
- Some menu items have short cut keys.
- A short cut key is usually a combination of keys

Tool bar:

- The tool bars are customizable
- The four built-in tool bars are in VB. They are standard, Edit, Debug and form editor.

Tool box:

- The tool box is located at the left side of the screen.
- It contains the controls to build the interface for the VB application.
- We can add new components by choosing project -> components

Form window:

- We can customize the form window by adding controls and changing its size

Project explorer

- Visual basic organizes its components into projects.
- Each project can have multiple forms; the code that activates the controls on a form is stored with the form in separated files.
- The general programming code shared by all the forms can be divided into different modules.
- The project explorer will display all the customizable forms and general code (module) that makeup the application.
- Three tools are available at the top of the project explorer window. They are
- View code
- View object
- Toggle folders
- By clicking the view code button, it will display the code associated with that form (or) object
- Visual basic stores all the files in a project file, they have .vbp extension.
- VB allows to have multiple project at the same time. This is called project group. The extension of project group is .vbg

The properties window:

- The properties window located above the form layout window on the right hand side of the VB environment.
- By pressing F4, we can make properties window visible.
- The second column of the properties window always indicates the current setting of the property. Hence the right-hand column of the properties window is called the setting box.
- The default arrangement in the properties is alphabetical. Click the categorized tab in the properties window to list out the properties based on functionality.

<u>COMMON FORM PROPERTIES:</u>

Caption:

- The caption property sets the title of the form.

Name

- This property is used to identify the control in coding

Appearance

- This property determines whether the form will have a three-dimensional look.
- The default value of this property is 1 for 3d look.
- If the value is 0, then the form will appear flat.

Border Style

There are six values for this property.

1. None
2. Fixed size
3. Sizable
4. Fixed double
5. Fixed tool window
6. Sizable tool window

- If the border style value is 0, then the application (form) will show no border and therefore no minimize, maximize or control buttons. Because of this, a form created without a border cannot be moved, resizes. This setting is useful for splash screen.
- If the setting is 1, the user will not be able to resize the window.
- If the value of the border style property is 2(sizable), then the user ca resize the form via the hot spot located on the boundary of the form.
- If the value is 3-fixe double, then it is used for design a dialog box.
- If the setting is 4- fixed tool window, then it will display the form with a close window, and the text on the title bar will be display in reduced size.
- If the setting is 5-sizable tool window, it works much like as fixed tool window.

Control Box

- Control box are located in the far left corner of the title bar. The changes in the control box property go into effect only at runtime . we can assign the value either ‘true’ or ‘false’ to the control box.

Enabled

- If the enabled property set to false, then the form cannot respond to any events.

StartupPosition

- This property decides the initial position of the form at runtime. The default value is 0.

Visible

- Setting the value of this property to false, the form will not bew visible at runtime.

Windowstate

- This property determines how the form will work at runtime there are three possible settings.

1. Normal (default setting)
2. Minimized (reduce the form into icon)
3. Maximized

Writing Simple Programs

visual basic program to print a string “Hello World”

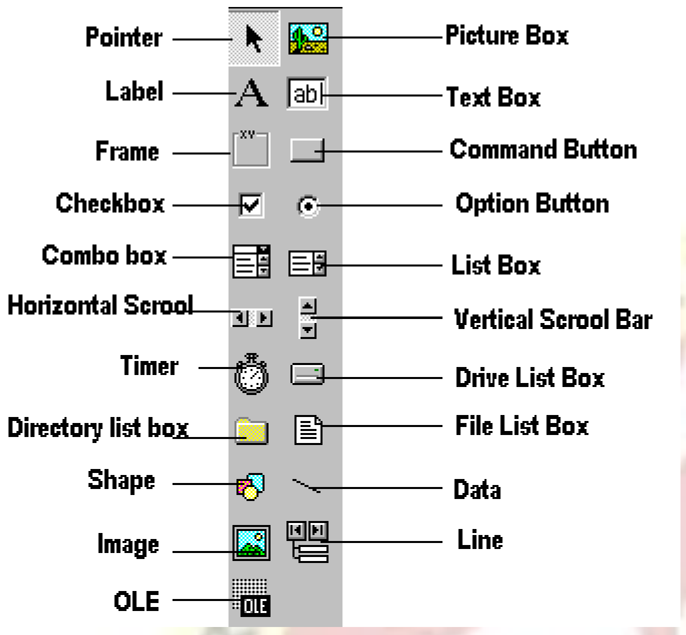
The below code will print the string value “Hello World”. **Console.WriteLine(” “)** is used to print any value as an output and the **Console.ReadLine()** is used to read the next line here we are using it to hold the screen.

```
Module Module1
Sub Main()
Console.WriteLine("Hello World")
Console.ReadLine()
End Sub
End Module
```

Output: Hello World

Toolbox

The **Toolbox** window displays controls that you can add to Visual Studio projects. To open **Toolbox**, choose **View** > **Toolbox** from the menu bar, or press **Ctrl**+**Alt**+**X**.



You can drag and drop different controls onto the surface of the designer you are using, and resize and position the controls.

Creating Controls

1. start a new project.
2. Choose ActiveX control from the New Project dialog box.
3. Add a picture box to the usercontrol – it looks like a form without a border.
4. Add a timer with the interval property set to 50.
5. Rename the picture box 'ScrollBox' and resize it to about 4000 wide and 700 in height.
6. Rename the usercontrol to 'ScrollControl' and resize it so it's just slightly larger than the picture box.
7. Select a bitmap for the 'ToolboxBitmap' property, this will be the image that eventually represents your control on the toolbox.
8. Now we're ready for the code.

Name property

The **Name property** is a string used by clients to identify, find, or announce an object for the user. All objects support the **Name property**. For example, the text on a button control is its **name**, while the **name** for a list box or edit control is the static text that immediately precedes the control in the tabbing order.

Command Button

The CommandButton control is simply a button that we see in our daily-use software. When the end-user clicks the CommandButton, the program behaves according to the code assigned in the CommonButton.

PROPERTIES OF COMMAND BUTTON

Name

This property is used to identify the control at coding time. This property determines the name visual basic uses for the event procedure. The name property for any control is more important. Microsoft suggests using a prefix to start the name of any control. For command button the prefix is “cmd”.

The limitations on assigning control name are:

- The name must begin with a letter
- After that we can use any combination of letters, digits and underscores.
- No spaces are allowed
- The name cannot be longer than 40 characters.

Caption

- The caption property determines what the user sees on the face of button.
- The caption property can use any symbols available in a single font.
- The caption on a command button is automatically centered within the button.

Visible

- This property determines whether the command button is visible or not at runtime. The command button is visible at design time even the visible property is set to false.

Enabled

- This property determines whether the button can respond to any event. If this property set to false, VB will not respond to any event concerning that button.

Font

- This property controls which font is used for the caption of the button.
- All the font characteristic – bold, italic, etc. can be set independently for each command button.

Height, Width

- These property define the height and width of the command button.
- We can change the setting for these properties directly from the properties window.

Disabled Pictures, Down Pictures, Picture, Style

- To display the picture in the command button, set the value of the style property to 1- graphical.
- The default value of the style property is 0-standard.
- In addition to setting a picture for a command button in tis normal state, we can set a special picture when the control is disabled (or) when it is clicked.
- The command button allows all the standard picture type (bitmaps, icons, jpegs, gif, metafile, etc.)
- The command button wont resize itself to fit the image.

Cancel

- A user can activate an escape command button by pressing 'esc' key on the keyboard.
- By setting the cancel property to 'true ', we can make the command button as escape command button.

Default

- We can set the default command button by setting the value of default property of the command button to 'true'.

Access Keys

- To set the access key for a command button place an ampersand (&) in front of the letter that we want to create the access key, in the caption.
- We can activate the access key by pressing key combination with ALT key.
- It is possible to have the same access key for more than one control.
- An access key is an underlined character in the text of a menu, menu item, or the label of a control such as a button. With an access key, the user can "click" a button by pressing the Alt key in combination with the predefined access key. An access key is an underlined character in the text of a menu, menu item, or the label of a control such as a button. With an access key, the user can "click" a button by pressing the Alt key in combination with the predefined access key.

- For example, if a button runs a procedure to print a form, and therefore its Text property is set to "Print," adding an ampersand before the letter "P" causes the letter "P" to be underlined in the button text at run time. The user can run the command associated with the button by pressing Alt+P. Controls that cannot receive focus can't have access keys. Let's say that you have a TextBox representing the Department name on an entry screen. The Label to the left of the TextBox duly contains the Caption "Department." If you put an ampersand (&) in front of the "D" in "Department," the "D" will appear underlined. When users see this, they will think, of course, that pressing Alt-d will place the cursor in the TextBox. Unfortunately, the "D" represents an access key for the Label—and moreover, the Label is incapable of receiving focus.

<u>IMAGE CONTROL</u>

- Imagecontrol hold pictures
- Imagecontrol are user to display icons (or) picture
- The Loadpicture function will reset the value of the picture property
- The most important property of an image control is the stretch property
- This property determines whether the image control adjusts to fit the picture (or) the picture adjust to fit the control

Image Control: 1) it is not act as container control 2) not use of memory to store the picture 3) editing of picture is not possible in picture box 4) Not having auto size property 5) Having stretch property The Image control is a lightweight control that has no device context (or hDC) or it's own. The Image control lets you display a picture as part of the data in a form. For example, you might use an Image to display employee photographs in a personnel form.

The Image lets you crop, size, or zoom a picture, but does not allow you to edit the contents of the picture. For example, you cannot use the Image to change the colors in the picture, to make the picture transparent, or to refine the image of the picture. You must use image editing software for these purposes. The Image supports the following file formats: *.bmp• *.cur• *.gif• *.ico• *.jpg• *.wmf

<u>TEXT BOX PROPERTIES</u>

- The text boxes are the primary control for accept input and display output. The content of the boxes are treated as String. For numeric calculation, transforming a string into a number using built-in-function is required.
- Microsoft prefix for the name property of a text box is “txt”.
- Special properties of text boxes:

Text

- Text property holds the information entered in the text bar.
- The default value for this property is set to text1, text2, and so on.

Alignment

- This property controls how the text is displayed in a text box.
- The default value is 0 – left, which leaves the text left aligned.
- Use 1-right, 2-center are the value for other alignment.

Multiline

- This property determines whether a text box can accept more than one line of text.
- The limit of the multiline text box is approximately 32000 characters.
- This property usually combined with resetting the value of the scroll bar property
- If the multiline property is set to “true”, a user can always use the standard methods in window to move through the text box.
- VB automatically word wraps when the user types more than one line of text.
- Use enter key to separate lines.

Scrollbars

- This property determines whether a text box has horizontal (or) vertical scroll bar
- These are useful to accept long (or) multiple lines of character from a single text box box.
- The four possible setting for the scroll bar property are

1. None (this will display no scroll bar and this the default value)
2. Horizontal (horizontal text limits inside the text box to a total of 255 characters)
3. Vertical (text box has vertical scroll bars only)
4. Both (text box has both vertical an d horizontal scroll bars)

BorderStyle

- The default value of the border style property is 1-fixed single, which give the single width border to the text box.
- If the value of the border style property 0 – none, the border disappears.

Maxlength

- This determines the maximum number of character that the textbox will accept.
- Any setting other than 0 will limit the user’s ability to enter data into that textbox to that number of characters.

Passwordchar

- This property limits the textbox display.
- If the value of this property is set to ‘*’, all the character entered in the box will display as row asterisks.

Locked

- This property is used to prevent users from changing the contents of the textbox.
- User can scroll and highlight but won’t be able to change it.
- Its advantage over setting the enabled property to false is that textbox is not grayed.

LABELS

- Labels are used to display the information but the user cannot be able to change.
- The most common use for label is to identify a textbox or other controls by describing its contents.
- Microsoft suggested prefix for the name property of the label is ‘lbl’.
- The enabled property is not often used for labels.

Alignment

- The alignment property for a label has three possible setting.
- The default value is 0 – left, which leaves the text left aligned.
- Use 1-right, 2-center are the value for other alignment.

Border Style, Back Style

- The border style property has the same two possible values as textbox.
- The difference is that the default value is 0
- Set the border style property to 1, the label resembles a textbox.
- This property is used for displaying outputs avoids the problem of textboxes being changed by the user.
- The back style property determines whether the label is transparent or opaque.

Autosize, Wordwrap

- By setting auto size property value to true, labels can b made to grow automatically in a horizontal direction.
- The default value of this property is set to false
- By setting word-wrap property to true, the label will grow in the vertical direction.

<u>**MESSAGE BOXES**</u>

- The message boxes are used to displays information in a dialog box super imposed on the form.
- The message box will for the user to choose a button before returning to application.
- Without responding to the message, the user cannot switch to any other forms.
- The simplest form of the message box is

Msgbox “welcome”

- The complete syntax for the msgbox command is,

Msgbox message_dispolayed_in_msbox, type_of_box, title_of_box

- The user can combine three different groups of built-in integer contents to specify the kind of the message box.
- The three groups are,
  - Type of button
  - Type of icon
  - Default button selected

The first group of numbers controls what type of button will be displayed in the message box.

VALUE	SYMBOLIC CONSTANT	DESCRIPTION
0	VbOkOnly	Display only the ok button
1	VbOkCancel	Display ok and cancel button
2	VbAbortRetryTgnore	Display abort, retry and ignore buttons
3	VbYesNoCancel	Display yes, no and cancel buttons
4	VbYesNo	Display yes, no buttons
5	VbRetryCancel	Display retry and cancel buttons

- The second group of numbers control what type of icon will be displayed in the message box

VALUE	SYMBOLIC CONSTANTS	DESCRIPTION
16	VbCritical	Display critical message icon
32	VbQuestion	Display warning query icon
48	VbExclamation	Display warning message icon
64	VbInformation	Display information message icon

- The next group of numbers controls which button is the default button for the box.

VALUE	SYMBOLIC CONSTANTS	DESCRIPTION
0	VbDefaultButton1	First button is default
256	VbDefaultButton2	second button is default
512	VbDefaultButton3	third button is default

Example:

```
Op = Msgbox("Do you want to delete?",VbYesNo+VbCritical+VbDefaultButton2,"Delete?")
If  Op= VbYes Then
'Write the code for Delete
End if
```

<u>GRID</u>

- The grid is important foir accurately positioning controls
- To control the grid, choose

Tools→option and then go to the general page

- The four properties that the user should control the grid are described below

Show Grid

- The user can turn the grid on or off by changing the show grid setting
- The default setting is on

Grid Width, Grid Height

- The height and width setting is used to set the distance (in twips) between grid marks
- The default is 120 twips

Align Controls To Grid

- This check box determines whether controls automatically move to the next grid mark (or) whether they can be placed etween grid marks.
- The alternate way for aligning controls to grid is, choose

Format→align→to grid

VB EDITING TOOLS

THE EDITOR FORMAT PAGE

- Choose tools→ option and click on the editor format tab
- The code colors frame contains list of possible objects that the user can change

(ex: normal text, comment text, keyboard text and so on)

Font List Box

- This will display the complete list of fonts that is already installed in the system.

Size List Box

- The user can type a point size for the font directly in the size list box
- This option is used to change the font size of the VB environment

Foreground, Background And Indicator List Box

- The three code colors boxes determines the foreground and back ground colors used for each type of code
- The third list box used for indicators in the margin such as book marks.

Sample Box

- This box is used to see a sample text in the font, size and color setting that are currently set.

THE EDITOR PAGE:

Auto Syntax Check

- Visual basic automatically check the code when the programmer press ENTER and there by detect certain kinds of mistakes immediately

Require Variable Declaration

- Keep this option checked always

Tab Width Box

- Use the tab width box to set the number of spaces the user gets when the user press TAB key
- This can be anything from 1 to 32.
- The default is 4 spaces
- Tf the user have selected text,
- Pressing TAB shift all the text forward one tab stop
- Pressing SHIFT+ TAB all the text back one tab stop

Auto Indent Check Box

- If the auto indent box is checked, pressing ENTER after you use the TAB key to indent a line makes the subsequent line start at the same place as the line above it.
- It is useful to make programming structure cleaner

Default To Full Module View

- When this option is checked, the user will see all the code as one unit the code window
- The user will scroll through the code with arrow keys
- If this option is unchecked, each procedure of code pops up in a separate window
- Use CTRL+↑ and CTRL+↓ to cycle through the different pieces of code.

Procedure Separator

- Use this check box in conjunction with full module view

THE EDIT TOOL BAR

- Many useful editing features have button equivalent on the edit tool bar
- The user can add the eedit tool bar, in the VB environment bu selecting

VIEW → tool bars

- The tools available in tool bar are discussed below

FUNCTION	SHORT CUT	DESCRIPTION
List properties methods	[CTRL + J]	Displays a pop-up list box with the properties and methods for the object preceding the period
List constants	[CTRL + SHIFT + J]	Display a pop-up list box with the valid constants
Quick info	[CTRL + I]	Gives the syntax for the procedure (or) method
Parameter Info	[CTRL + SHIFT + I]	Provides the parameter list for the current function call
Complete word	[CTRL+SPACEBAR]	Complete the keyword or object when enough information is there
Intend	[TAB]	Indents the selected text one tab stop. Use editor page to change the no. of spaces for the tab.
Out end	[SHIFT+TAB]	Moves the selected text back one tab stop
Toggle break point	[F9]	Used for debugging
Comment block		Used to comment the block of statement
Uncomment block		Used to uncomment the block of statement
Toggle bookmark		The editor allows the user to put book mark at specific places in the code. The user can jump from one bookmark to another easily
Next bookmark		Move to next bookmark

Previous bookmark		Move to previous bookmark
Clear all book marks		Used to clear all the bookmarks in the code

VARIABLE

- Variables are used to store information(values)
- When the user use a variable , VB set up an area in the computer's memory to store the information
- Variable name can be up to 255 characters
- The first letter of the variable should be an alphabet
- Any combination of letter, numbers and underscore can be given for a variable name
- The user can not use names reserved by VB
- Choose the meaning variable name
- The naming convention for styling variables name is, use capitals only at the beginning of the word.
- This convention is called mixed case variable name.

DATA TYPES

The data types available in VB are

1. Boolean
2. Byte
3. Integer
4. Long integer
5. Single precision
6. Double precision
7. Currency
8. String
9. Date
10. Variant

BOOLEAN:

- Use the Boolean data type when you need to store either TRUE or FALSE

BYTE:

- This data type was added to VB 5
- It will hold integer values between 0 to 255

INTEGER:

- Integer data type are used to store integer values between -32768 and 32767
- Integer arithmetic is very fast
- The integer variable are capable of holding integer within the range
- Use % sign at the end of integer variable name

LONG INTEGER:

- The long integer variable hold integer value between -2147483648 and 2147483647
- The identifier that the user can use for the variable is the ampersand(&)
- Long integer arithmetic is also fast

SINGLE PRECISION (!):

- This data type is used to store numbers with decimal points
- The range of the number is up to 38 digits
- The calculation will always be approximate for these types of variables
- This is used for numbers with only seven digit accuracy

DOUBLE PRECISION(#):

- The double precision data type is used when the user need up to 16 places of accuracy
- The range is more than 300 digits
- The calculation are also approximately for these variables
- Calculation are relatively slow with double precision numbers
- The identifier used for double for double precision variable is a pound sign.

CURRENCY:

- The currency data type is designed to award certain problems in switching from binary fractions to decimal fraction.
- The currency data type can have 4 digits to the right of the decimal place and up to 15 to the left of the decimal point
- Arithmetic calculation will be exact with in this range
- This is the preferred data type to use for financial calculation

STRING($):

- The string data type holds characters
- A variable holding a string is called a string variable
- A string variable can hold about 2 billion character
- Due to memory constraint, the no. of characters may vary.
- The most common use of string variables is to store the information contained in a text box.

DATE:

- This is a data type used to store both date and time
- We can store the data value between midnight on Jan 1, 100 and midnight on Dec 31, 9999.
- Use # symbol at the beginning and ending of the data values assignment

Ex:dob= #31-05-1980#

- If the time is not include in the date, VB assumes it is midnight
- We can use AM/PM for assigning time

Ex:dob= #31-05-1980 10:50AM#

UNIT 2

Displaying Information

- VB displays text on a form using 'print' function
- The general syntax of print method is

Formname.print expression

- Where expression is any VB expression that VB can convert to a string
- Use Autodraw property of the form to refresh printed text on the form
- Cls is a command used to clear the form

Determinate Loops - Indeterminate Loops

We can write a Visual Basic procedure that allows the program to run repeatedly until a condition or a set of conditions is met. This is procedure is known as looping . Looping is a very useful feature of Visual Basic because it makes repetitive works easier. There are three kinds of loops in Visual Basic, the **Do...Loop** ,the **For.......Next loop** and the **While.....Wend Loop.**

The Do Loop

The Do Loop statements have four different forms, as shown below:

a)

```
 Do While condition
  Block of one or more VB statements
Loop
```

b)

```
 Do
  Block of one or more VB statements
 Loop While condition
```

c)

```
 Do Until condition
  Block of one or more VB statements
 Loop
```

d)

```
Do
 Block of one or more VB statements
Loop Until condition
```

Example 9.1

```
Do while
counter <=1000
num.Text=counter
counter =counter+1
Loop
```

* The above example will keep on adding until counter > 1000

The above example can be rewritten as

```
Do
counter=counter+1
Loop until counter>1000
```

Exiting the Loop

Sometime we need exit to exit a loop earlier when a certain condition is fulfilled. The keyword to use is **Exit Do**. You can examine Example for its usage.

Example

```
Dim sum, n as Integer
Private Sub Form_Activate()
List1.AddItem "n" & vbTab & "sum"
Do
n=n+1
sum=sum+n-resize
List1.AddItem n & vbTab & sum
If n=100 Then
Exit Do
End If
Loop
End Sub
```

Explanation

In the above example, we compute the summation of 1+2+3+4+……+100. In the design stage, you need to insert a ListBox into the form for displaying the output, named List1. The program uses the AddItem method to populate the ListBox. The statement List1.AddItem "n" & vbTab & "sum" will display the headings in the ListBox, where it uses the vbTab function to create a space between the headings n *and sum.*

The For....Next Loop

The For....Next Loop event procedure is written as follows:

```
For counter=startNumber to endNumber (Step increment)

 One or more VB statements

Next
```

Example a

```
Private Sub Command1_Click()
Dim counter As Integer
For counter = 1 To 10
List1.AddItem counter
Next
End Sub
```

Example b

```
Private Sub Command1_Click()
```

```
Dim counter As Integer
For counter = 1 To 1000 Step 10
counter = counter + 1
Print counter
Next
End Sub
```

Example c

```
For counter=1000 to 5 step -5
counter=counter-10
If counter<50 then
Exit For
Else
Print "Keep Counting"
End If
Next
```

Example d

```
Private Sub Form_Activate( )
For n=1 to 10
If n>6 then
Exit For
Else
Print n
Enf If
Next
End Sub
```

Sometimes the user might want to get out from the loop before the whole repetitive process is executed, the command to use is **Exit For**. To exit a For….Next Loop, you can place the **Exit For** statement within the loop; and it is normally used together with the If…..Then… statement.

The While….Wend Loop

The structure of a While….Wend Loop is very similar to the Do Loop. it takes the following form:

```
 While condition
  Statements
 Wend
```

The above loop means that while the condition is not met, the loop will go on. The loop will end when the condition is met. Let’s examine the program listed in example

Example

```
Dim sum, n  As Integer
Private Sub Form_Activate()
List1.AddItem "n" & vbTab & "sum"
While n <> 100
n = n + 1
Sum = Sum + n
List1.AddItem n & vbTab & Sum
Wend
End Sub
```

If.....Then.....Else Statements with Operators

To effectively control the VB program flow, we shall use **If...Then...Else** statement together with the conditional operators and logical operators.

```
If conditions Then

VB expressions

Else

VB expressions

End If
```

Example :

This program simulates a sign in process. If the username and password are correct, sign in is successful else sign in failed. Start VB6 and insert two textboxes on the form, rename them UsrTxt and pwTxt, the first textbox is to accept username input and the second one for password input. For pwTxt, set the PasswordChr(password characters) property to * so that the password will appear as * instead of the actual character. We have written the code so that both username and password must be correct to enable sign in if either one of them incorrect sign in will fail.

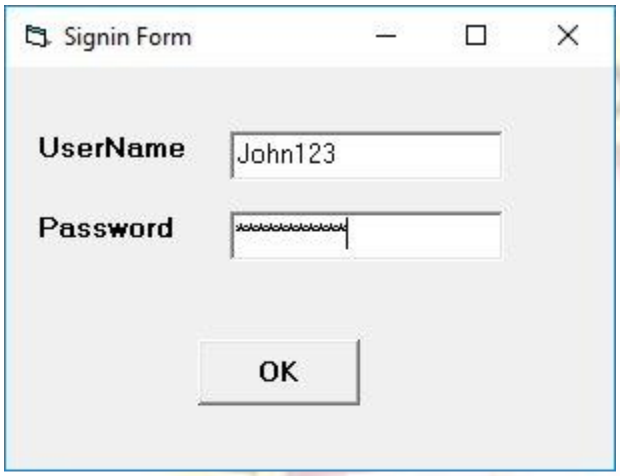
The Code

```
Private Sub OK_Click()
Dim username, password As String
username = "John123"
password = "qwertyupi#@"
```

```
If UsrTxt.Text = username And pwTxt.Text = password Then
MsgBox ("Sign in sucessful")
ElseIf UsrTxt.Text <> username Or pwTxt.Text <> password Then

MsgBox ("Sign in failed")
End If
End Sub
```

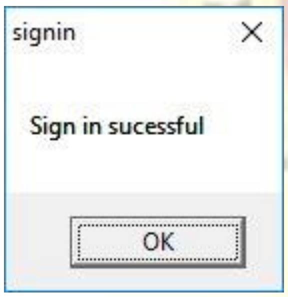
The Output



John123

****

Figure 7.1



Sign in sucessful

Select Case

we shall examine another way to control the program flow, that is, the **Select Case** control structure. The Select Case control structure is slightly different from the If....ElseIf control structure .The difference is that the **Select Case** control structure can handle conditions with multiple outcomes in an easier manner than the **If...Then...ElseIf** control structure.

The syntax of the Select Case control structure is shown below:

```
Select Case expression
   Case value1
            Block of one or  more VB statements

            Case value2
            Block of one or more VB Statements

   Case Else
            Block of one or more VB Statements
End Select
```

Example

```
Dim grade As String
Private Sub Compute_Click( )
grade=txtgrade.Text
Select Case grade
Case "A"
result.Caption="High Distinction"
Case "A-"
result.Caption="Distinction"
Case "B"
result.Caption="Credit"
Case "C"
result.Caption="Pass"
Case Else
result.Caption="Fail"
End Select
End Sub
```

<u>Built-in Functions</u>

Mathematical Functions

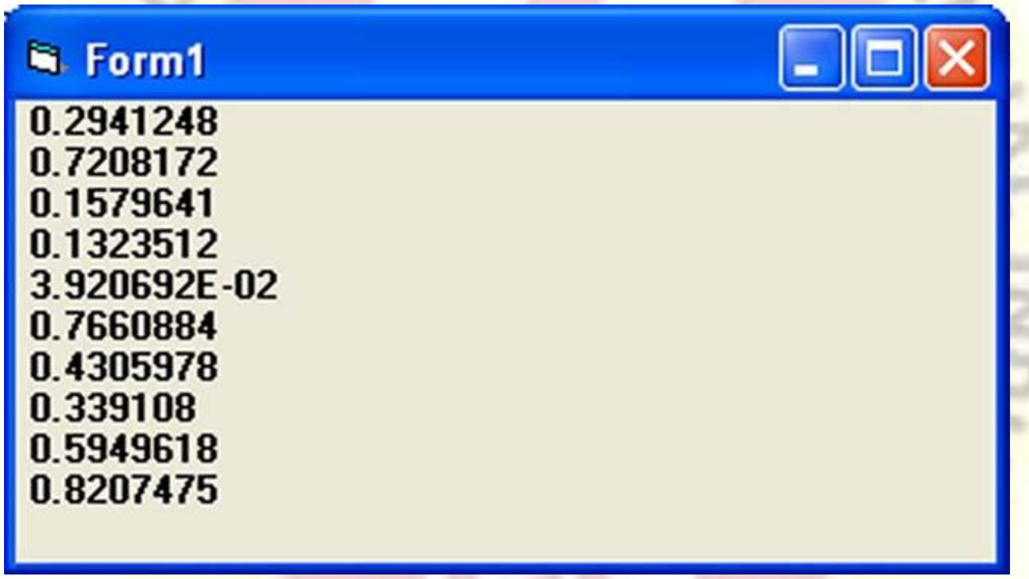
The mathematical functions are very useful and important in programming because very often we need to deal with mathematical concepts in programming such as chance and probability, variables, mathematical logics, calculations, coordinates, time intervals and etc. The common mathematical functions in Visual Basic are **Rnd, Sqr, Int, Abs, Exp, Log, Sin, Cos, Tan , Atn, Fix** and **Round.**

The Rnd Function

Rnd is is very useful function for dealing with the concept of chance and probability. The Rnd function returns a random value between 0 and 1. In Example 11.1. When you run the program, you will get an output of 10 random numbers between 0 and 1. Randomize Timer is to randomize the process.

Example Random Number Generation

```
Private Sub Form_Activate
Dim x as integer
For x=1 to 10
Print Rnd
Next
End Sub
```



```
0.2941248
0.7208172
0.1579641
0.1323512
3.920692E-02
0.7660884
0.4305978
0.339108
0.5949618
0.8207475
```

The Numeric Functions

The numeric functions are **Int**, **Sqr**, **Abs**, **Exp**, **Fix**, **Round** and Log.

a) **Int** is the function that converts a number into an integer by truncating its decimal part and the resulting integer is the largest integer that is smaller than the number. For example, Int(2.4)=2, Int(4.8)=4, Int(-4.6)= -5, Int(0.032)=0 and so on.

b) **Sqr** is the function that computes the square root of a number. For example, Sqr(4)=2, Sqr(9)=2 and etc.

c) **Abs** is the function that returns the absolute value of a number. So Abs(-8) = 8 and Abs(8)= 8.

d) **Exp** of a number x is the value of e^x. For example, Exp(1)=e^1 = 2.7182818284590

e) **Fix** and **Int** are the same if the number is a positive number as both truncate the decimal part of the number and return an integer. However, when the number is negative, it will return the smallest integer that is larger than the number. For example, Fix(-6.34)= -6 while Int(-6.34)=-7.

f) **Round** is the function that rounds up a number to a certain number of decimal places. The Format is Round (n, m) which means to round a number n to m decimal places. For example, Round (7.2567, 2) =7.26

g) **Log** is the function that returns the natural Logarithm of a number. For example,

Log 10= 2.302585

The Formatting Functions

The Tab function

The syntax of a Tab function is **Tab (n); x**

The item x will be displayed at a position that is n spaces from the left border of the output form. There must be a semicolon in between Tab and the items you intend to display (VB will actually do it for you automatically).

The Space function

The **Space** function is very closely linked to the Tab function. However, there is a minor difference. While Tab (n) means the item is placed n spaces from the left border of the screen, the Space function specifies the number of spaces between two consecutive items. For example, the procedure

Example 12.2

```
Private Sub Form_Activate()
Print "Visual"; Space(10);"Basic"
End Sub
```

Means that the words Visual and Basic will be separated by 10 spaces

The Format function

The **Format** function is a very powerful formatting function which can display the numeric values in various forms. There are two types of Format function, one of them is the built-in or predefined format while another one can be defined by the users.

(a) The syntax of the predefined Format function is

Format (n, “style argument”)

where n is a number and the list of style arguments is given in Table

Table : List of Style Arguments

Style argument	Explanation	Example
General Number	To display the number without having separators between thousands.	Format(8972.234, "General Number")=8972.234
Fixed	To display the number without having separators between thousands and rounds it up to two decimal places.	Format(8972.2, "Fixed")=8972.23
Standard	To display the number with separators or separators between thousands and rounds it up to two decimal places.	Format(6648972.265, "Standard")= 6,648,972.27
Currency	To display the number with the dollar sign in front has separators between thousands as well as rounding it up to two decimal places.	Format(6648972.265, "Currency")= $6,648,972.27
Percent	Converts the number to the percentage form and displays a % sign and rounds it up to two decimal places.	Format(0.56324, "Percent")=56.32 %

The syntax of the user-defined Format function is

Format (n, “user’s format”)

Although it is known as user-defined format, we still need to follows certain formatting styles. Examples of user-defined formatting style are listed in Table

Table: User-Defined Formatting Functions

Format	Description	Output
Format(781234.576,"0")	Rounds to whole number without separators between thousands	781235
Format(781234.576,"0.0")	Rounds to 1 decimal place without separators between thousands	781234.6

Format(781234.576,"0.00")	Rounds to 2 decimal place without separators between thousands	781234.58
Format(781234.576,"#,##0.00")	Rounds to 2 decimal place with separators between thousands	781,234.58
Format(781234.576,"$#,##0.00")	Displays dollar sign and Rounds to 2 decimal place with separators between thousands	$781,234.58
Format(0.576,"0%")	Converts to percentage form without decimal place	58%
Format(0.5768,"0%")	Converts to percentage form with two decimal places	57.68%

String Manipulation Functions

The Len Function

The Len function returns an integer value which is the length of a phrase or a sentence, including the empty spaces. The syntax is

Len (“Phrase”)

For example,

Len (VisualBasic) = 11 and Len (welcome to VB tutorial) = 22

The Len function can also return the number of digits or memory locations of a number that is stored in the computer. For example,

X=sqr (16)

Y=1234

Z#=10#

Then Len(x)=1, Len(y)=4, and Len (z)=8

The reason why Len(z)=8 is because z# is a double precision number and so it is allocated more memory spaces.

The Right Function

The Right function extracts the right portion of a phrase. The syntax is

Right (“Phrase”, n)

Where n is the starting position from the right of the phrase where the portion of the phrase is going to be extracted. For example,

Right(“Visual Basic”, 4) = asic

The Left Function

The Left$ function extract the left portion of a phrase. The syntax is

Left(“Phrase”, n)

Where n is the starting position from the left of the phase where the portion of the phrase is going to be extracted. For example,

Left (“Visual Basic”, 4) = Visu

The Ltrim Function

The Ltrim function trims the empty spaces of the left portion of the phrase. The syntax is

Ltrim(“Phrase”)

.For example,

Ltrim (“ Visual Basic”, 4)= Visual basic

The Rtrim Function

The Rtrim function trims the empty spaces of the right portion of the phrase. The syntax is

Rtrim(“Phrase”)

.For example,

Rtrim (“Visual Basic   ”, 4) = Visual basic

The Trim function

The Trim function trims the empty spaces on both side of the phrase. The syntax is

Trim(“Phrase”)

.For example,

Trim (“   Visual Basic   ”) = Visual basic

The Mid Function

The **Mid** function extracts a substring from the original phrase or string. It takes the following format:

Mid(phrase, position, n)

Where position is the starting position of the phrase from which the extraction process will start and n is the number of characters to be extracted. For example,

Mid(“Visual Basic”, 3, 6) = ual Bas

The InStr function

The **InStr** function looks for a phrase that is embedded within the original phrase and returns the starting position of the embedded phrase. The syntax is

Instr (n, original phase, embedded phrase)

Where n is the position where the Instr function will begin to look for the embedded phrase. For example

Instr(1, “Visual Basic”,” Basic”)=8

The Ucase and the Lcase functions

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

Ucase(“Visual Basic”) =VISUAL BASIC

Lcase(“Visual Basic”) =visual basic

The Str and Val functions

The **Str** is the function that converts a number to a string while the **Val** function converts a string to a number. The two functions are important when we need to perform mathematical operations.

The Chr and the Asc functions

The **Chr** function returns the string that corresponds to an ASCII code while the **Asc** function converts an ASCII character or symbol to the corresponding ASCII code. ASCII stands for “American Standard Code for Information Interchange”. Altogether there are 255 ASCII codes and as many ASCII characters. Some of the characters may not be displayed as they may represent some actions such as the pressing of a key or produce a beep sound. The syntax of the Chr function is

Chr(charcode)

and the syntax of the Asc function is

Asc(Character)

The following are some examples:

Chr(65)=A, Chr(122)=z, Chr(37)=% , Asc("B")=66, Asc("&")=38

FUNCTION AND PROCEDURE

- To attach the function with the current form, open the code window, and then choose tools menu and select add procedure
- The first line of the function is called the header of the function

Public function random1to(x) as integr

Randomize

Random1to – int (x * rnd)+1

End function

Public function add2no (x as integer, y as integer) as single

Add2no=csgl(x+y)

End function

- The keyword public is called as access specifier
- Where x is a parameter in random to function and x, y are the parameter in add2no function
- In the body of the function, assign a value to the function using its name

  For example,

  Add2no=csgl(x+y)
- The value the function gets is depends on the parameter

USER DEFINED FUNCTION

The simplest form of a function definition is,

Public function functionName (para1, para2,…)

Statements

Function name= expression

===

Statements

===

Function name= expresio0n

End function

Where parameter1, parameter2 ,… are variables

- These variables are referred to as the parameter (or) arguments of the function
- The types of the parameters can specified by type-declaration
- After process the statements, VB sends the information to the function definition
- The last value assigned to the functionname inside the body of the function is the result of the function
- Each variable to send to a parameter must be of the same type

SCOPE OF VARIABLES USED IN FUNCTION PROCEDURE

- As with events procedure, the user can setup own local variable inside function procedure
- Any variable declared within the body of the function using 'dim' keyword will be local to the procedure regardless of whether there is a form level variable with the same name
- The parameter of the function are automatically local to the procedure and no need to declare the parameter inside the procedure

SUB PROCEDURES

- To add new sub procedure, select tools menu and choose add procedure
- The structure of the simplest sub procedure is,

Public sub procedurename()

Statements

End sub

- The first line of the sub procedure is called as header
- Public is the access identifier
- The next in the header is the keyword ‘sub’ followed by the procedure name
- The parameters list should be enclosed within the parenthesis
- The ‘end sub’ keyword are used to indicate the end of a general procedure
- The procedure list will be used to communicate between the program and the procedure
- To use the procedure in the main program, the ‘call’ keyword is used
- To use the call keyword, the user must give parenthesis around the argument list

<u>USES OF PROCEDURE AND FUNCTION</u>

PASSING BY REFERENCE, PASSING BY VALUE

- When the user call a function (or) procedure, there are two ways to pass in a variable as an argument
- They are
- Passing by reference
- Passing by value
- When the user pass an arguments variable by reference, any changes to the corresponding parameters inside the procedure <u>will change the value of the original argument</u>
- When the user pass an argument by value, then the original variable retains its original value.

Example:

Sub triple(num as integer)

Num=3*num

Me.print ”inside the procedure the parameter value is ” & num

Me.print

End sub

```
Sub form_load()

Dim amt as integer /* this is local variable to this procedure */

Amt=2

Me.print ”from form load procedure the parameter value is ” & Amt

Me.print

Triple amt

Me.print ”After procedure call the parameter value is ” & Amt

End sub
```

From parameter pass by value

Change the statement

Triple (Amt)

- Here the variable is passed by reference because it is surrounded by parenthesis

<u>OUTPUT</u>

	Call by reference	Call by value
Form load	2	2
Inside procedure	6	6
After procedure call	6	2

Unit 3

<u>Arrays</u>

An array is a collection of items of the same data type. All items have the same name and they are identified by a subscript or index. When you need to work with several similar data values, you can use array to eliminate the difficulties of declaring so many variables. For example, if you want to compute the daily sales and sum the sales amount after 30 days, you don't need to have 30 variables.

Declaring an array

Syntax: Dim Variable_Name(index) As [Type]

Example:

```
 Dim month(10) As Integer '11 elements '
```

Or

Dim month(1 to 12) as Integer '12 elements

In the first line, month(10) is a collection of 11 integer values or items. month(0) is the 1st item and month(10) is the 10th & last item of the array. So 0 and 10 are respectively the lower bound and upper bound of the array.

In the other line, month(1 to 12) is a collection of 12 integer values or elements or items where month(1) is the 1st item and month(12) is the last. So 1 and 12 are respectively the lower bound and upper bound of the array.

Types of array

The array used in the example is a one-dimensional and fixed-size array. An array can have more than one dimension. The other types of arrays are multidimensional arrays, Dynamic arrays and Control arrays.

Fixed-Size Array: We know the total number of items the array in the above example holds. So that is a Fixed-Size array.

The LBound and UBound functions

The LBound and Ubound functions return the lower bound and upper bound of an array respectively.

Example:

```
Private Sub cmdDisplay_Click()
 Dim arr(10) As Integer
a = LBound(arr)
b = UBound(arr)
 MsgBox "Lower bound = " & a & " Upper bound = " & b
 End Sub
```

Initializing an array

You can use For Loop to initialize an array.

Example:

```
Dim day(10) As Integer, i As Integer
 For i = 0 To 10
day(i) = InputBox("Enter day value")
```

Next i

You can also initialize each array item separately in the way a variable is initialized.

Static array

Basically, you can create either static or dynamic arrays. Static arrays must include a fixed number of items, and this number must be known at compile time so that the compiler can set aside the necessary amount of memory. You create a static array using a Dim statement with a constant argument:

This is a static array.

Dim Names(100) As String

Visual Basic starts indexing the array with 0. Therefore, the preceding array actually holds 101 items.

Dynamic Array

In case of a fixed size array, we have seen that the size of the array is fixed or unchanged, but there may be some situations where you may want to change the array size. A dynamic arraycan be resized at run time whenever you want.

Declaring dynamic arrays

Declare the array with empty dimension list.
Example : Dim arr() As Integer
1. .
Resize the array with the ReDim keyword. Example :
2.
ReDim arr(5) As Integer or, ReDim arr(2 To 5) As Integer

LIST

The **List** class is used to store generic types of collections objects. By using a generic class on the **list**, we can store one type of object. The **List** size can be dynamically different depending on the need of the application, such as adding, searching or inserting elements into a **list**.

Sorting and Searching

The ListView control provides the mechanisms for sorting its ms and searching for specific items, To sort the ListItems in the' control, you. must assign the value True to the Sorted property. This is a Boolean value that determines whether the Listltems in the collectipn will be sorted. Two related properties a.rethe So~er . and SortKey properties. The SortOrder property determines whether the Listltems are sorted in ascending or descending order, and the SortKey property determines the sorting key. USortKey is 0, the list is sorted according to the item's Text property. . If you want to sort the list according to a subitem, assign the subitem's Index value . to the SortKey property.

It’s common to sort a list when the column header is clicked. For this reason, the SortKey property is commonly set from within the ColumnClick event to sort the list using the clicked column

Arrays can be searched in two ways: with the BinarySearch method, which works on sorted arrays and is extremely fast, and with the IndexOf (and LastIndexOf) methods, which work regardless of the order of the elements. All three methods search for an instance of an item and return its index, and they’re all reference methods. The IndexOf and LastIndexOf methods are similar to the methods by the same name of the String class. They return the index of the first (or last) instance of an object in the array, or the value −1 if the object isn’t found in the array. Both methods are overloaded, and the simplest form of the IndexOf method is the following, where arrayName is the name of the array to be searched and object is the item you’re searching for:

itemIndex = System.Array.IndexOf(arrayName, object)
The LastIndexOf method’s syntax is identical, but the LastIndexOf method starts searching from the end of the array. If the item you’re searching for is unique in the array, both methods will return the same index.

Another form of the IndexOf and LastIndexOf methods allows you to begin the search at a specific index:

This form of the method starts searching in the segment of the array from startIndex to the end of the array. Finally, you can specify a range of indices in which the search will take place by using the following form of the method:

itemIndex = System.Array.IndexOf(arrayName, object, startIndex, endIndex)
You can search large arrays more efficiently with the BinarySearch method if the array is sorted. The simplest form of the BinarySearch method is the following:

System.Array.BinarySearch(arrayName, object)

The BinarySearch method returns an integer value, which is the index of the object you’re searching for in the array. If the object argument is not found, the method returns a negative value, which is the negative of the index of the next larger item minus one. This transformation, the negative of a number minus one, is called the one’s complement, and other languages provide an operator for it: the tilde (~). The one’s complement of 10 is −11, and the one’s complement of −3 is 2.

Recordset:

The current group of records associated with a data control; may be a table recordset , a dynaset, or a snapshot.

Visual Basic supports 3 kinds of Record Set as follows:-

Table Record Set:-

Table Record Set represents a single table as it exist in a Database file. Table Record Set are usually updatable unless the file is locked or open for read only.

Dynaset:-

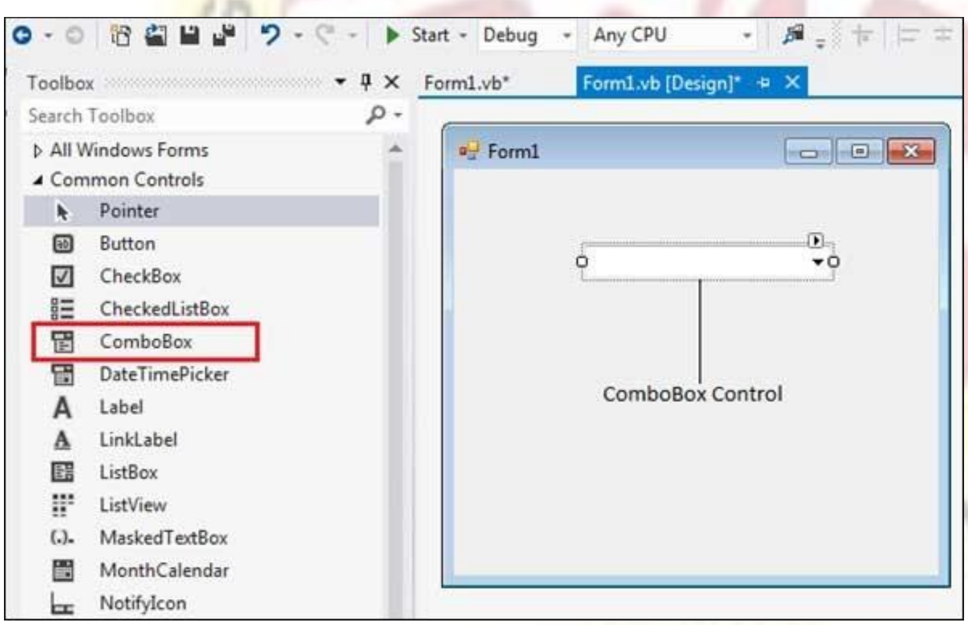
A Dynaset is temporary set of data taken from one or more table in the form one of many table in the underlined file. A Dynaset may be a query. That was defined in a access of table of result of joining multiple tables. Like a table, a Dynaset is updatable if file is not locked or open for read only. Data in Dynaset is live that is any changes made to data as project is executing will appear in Record Set.

Snapshots:-

Snapshot Record Set, like a dynaset, may be taken from one or more table. The difference is that snapshot is not updatable and also not live. A snapshot is like photograph a picture of reality a give point.

Control Array

}



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

Properties of the ComboBox Control

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

Sr.No.	Property & Description
1	**AllowSelection** Gets a value indicating whether the list enables selection of list items.
2	**AutoCompleteCustomSource** Gets or sets a custom System.Collections .Specialized.StringCollection to use when the AutoCompleteSourceproperty is set to CustomSource.
3	**AutoCompleteMode** Gets or sets an option that controls how automatic completion works for the ComboBox.
4	**AutoCompleteSource** Gets or sets a value specifying the source of complete strings used for automatic completion.
5	**DataBindings** Gets the data bindings for the control.
6	**DataManager** Gets the CurrencyManager associated with this control.
7	**DataSource** Gets or sets the data source for this ComboBox.
8	**DropDownHeight** Gets or sets the height in pixels of the drop-down portion of the ComboBox.
9	**DropDownStyle** Gets or sets a value specifying the style of the combo box.
10	**DropDownWidth** Gets or sets the width of the of the drop-down portion of a combo box.
13	**ItemHeight** **Gets or sets the height of an item in the combo box.**

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

Events of the ComboBox Control

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

Sr.No.	**Event & Description**
1	**DropDown** Occurs when the drop-down portion of a combo box is displayed.
2	**DropDownClosed** Occurs when the drop-down portion of a combo box is no longer visible.
3	**DropDownStyleChanged** Occurs when the DropDownStyle property of the ComboBox has changed.

4	**SelectedIndexChanged** Occurs when the SelectedIndex property of a ComboBox control has changed.
5	**SelectionChangeCommitted** Occurs when the selected item has changed and the change appears in the combo box.

Grid Control

The Databound grid control in VB 6.0 adds power and flexibility to your DataBase programs. you can easily provide grid access to any available DataBase.

You can provide simple display only access for used with summary data and on-screen reports.You can also provide editing capability to your dara grid including modifying only, add rights or delete rights.

It’s very easy to create a data grid form

A Grid Control-Step-by-Step

STEP 1: Begin a new project and widen the form.You may want to close the Project Explorer an Form Layout windows and float the properties

window to allow room to work on the wide form.

STEP 2: Add a data control along the bottom of the form.Set the control’s Name property to datBooks and its DatabaseName property to RnrBooks.mdb.Set the

RescordSource property to Books and verify that the RecordsetType is 1 – Dynaset.

STEP 3: Select project/Components to display the Components dialog box.Then locate Microsoft Data Bound Grid Control 5.0, select it , and close the dialog box.You should see the new tool in the toolbox.

STEP 4: Click on the DBGrid tool and draw a large grid on the form.Then , using the properties window , change the control’s Name property to dgbBooks and its DataSource property to datBooks.

STEP 5:Create the menu bar. It should have a File menu with only an Exit command.

STEP 6: Create the large label at the top of the form with the form’s title: Book List.Change the font and size to something like.

DoEvents function

Syntax

DoEvents( )

The **DoEvents** function returns an Integer representing the number of open forms in stand-alone versions of Visual Basic, such as Visual Basic, Professional Edition. **DoEvents** returns zero in all other applications.

DoEvents passes control to the operating system. Control is returned after the operating system has finished processing the events in its queue and all keys in the **SendKeys** queue have been sent.

DoEvents is most useful for simple things like allowing a user to cancel a process after it has started, for example a search for a file. For long-running processes, yielding the processor is better accomplished by using a Timer or delegating the task to an ActiveX EXE component. In the latter case, the task can continue completely independent of your application, and the operating system takes care of multitasking and time slicing.

Any time you temporarily yield the processor within an event procedure, make sure the procedure is not executed again from a different part of your code before the first call returns; this could cause unpredictable results. In addition, do not use **DoEvents** if other applications could possibly interact with your procedure in unforeseen ways during the time you have yielded control.

Example

This example uses the **DoEvents** function to cause execution to yield to the operating system once every 1000 iterations of the loop. **DoEvents** returns the number of open Visual Basic forms, but only when the host application is Visual Basic.

VBCopy
```
' Create a variable to hold number of Visual Basic forms loaded
' and visible.
Dim I, OpenForms
For I = 1 To 150000    ' Start loop.
    If I Mod 1000 = 0 Then     ' If loop has repeated 1000 times.
        OpenForms = DoEvents    ' Yield to operating system.
    End If
Next I    ' Increment loop counter.
```

Error Trapping

As you are writing your code, Visual Basic informs you of syntactical errors. However, once the program is running, you may encounter unexpected runtime errors in many circumstances.

For example, suppose you try to open a text file that the user has deleted. When a compiled program has an error like this, an error message is displayed and the program ends.

Although you cannot predict and write code for every possible type of error, "*File Not Found*" errors are fairly easy to handle. If you do not write code to work around the error, you can at least provide a message that makes more sense before ending the program.

The "On Error" Statement

The most common way to handle error conditions is to use Visual Basic's "*On Error*" statement. The "*On Error*" statement interrupts the normal flow of your program when an error occurs and begins executing your error handling code. A typical use is as follows :

```
On Error Goto FileOpenError
```

When this statement is executed, any errors that occur in subsequent statements cause Visual Basic to stop normal line-by-line execution and jump to the statement labeled as "*FileOpenError*".

Labeling Code Lines

Line labels in Visual Basic are similar to the line numbers of early BASIC. In Visual Basic, line labels can include text if you want, but each label must be unique. They are followed by a colon (:), as in the following example :

```
Private Sub Form_Load ()

On Error Goto FileOpenError

    Open "C:\SOMEFILE.TXT" For Unput As #1
    Line Input #1, sData
    Exit Sub

FileOpenError:
    MsgBox "There was a problem opening the file. Stop for coffee!"
    End

End Sub
```

In the preceding sample code, if the "*Open*" or "*Line Input*" statements cause an error, the statements starting at the label "*FileOpenError*" are executed, causing the message to be displayed and ending the program.

You should note a few points about the sample code. First, note the location and style of the error handling routine. It is usually placed near the end of the subroutine, with the label not indented to indicate a special section of code. Second, and more important, note the "*Exit Sub*" statement after the "*Open*" statement. It is necessary to prevent the error handler routine from executing even when the "*Open*" statement was successful.

Controlling Program Flow After an Error

In the preceding code example, you simply end the program if an error occurs. However, you can handle the error in several (better) ways :

- **Exit the subroutine after informing the user of the error, and allow the program to continue running with limited functionality.**
- **Resume execution with the next statement following the error.**
- **Provide a way for the user to correct the error and retry the offending statement.**

You can also have multiple labels within a procedure and set the current error handler multiple times. For example, you can add a line to the code sample after the "*Open*" statement that specifies a new label, "*FileInputError*". You can also turn off error handling with the following statement :

```
On Error Goto 0
```

The "*On Error*" statement goes hand in hand with the "*Resume*" statement. For example, this statement causes errors to be ignored and the program to proceed through each line of the code anyway :

```
On Error Resume Next
```

You should use the preceding line of code sparingly because it really just ignores errors rather than handles them. A better use of "*Resume*" is to go to another section of code. as in the following example :

```
Private Sub Form_Load ()

       On Error Goto FileOpenError
RetryHere:
       Open "C:\SOMEFILE.TXT" For Unput As #1
       Line Input #1, sData
       Exit Sub

FileOpenError:
       Dim sMessage As String

       sMessage =  "There was a problem opening the file. " & VbCrLf
       sMessage = sMessage & "Press Retry to try again, or Cancel to quit."

       If MsgBox (sMessage, vbRetryCancel + vbCritical, "Error!") = vbRetry Then
               Resume RetryHere
       Else
               End
       End If

End Sub
```

You can though use "*On Error Resume Next*" if you are trying to connect to AutoCAD

Determining The Type of Error

After an error has occurred, your code can find out more information about the error in several ways :

- **Err - Contains a number that represents the error.**
- **Error - Contains a string describing the error.**
- **Err Object - Contains error number, description and additional information.**

Also used to raise your own custom errors.

UNIT 4

Dialog boxes

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

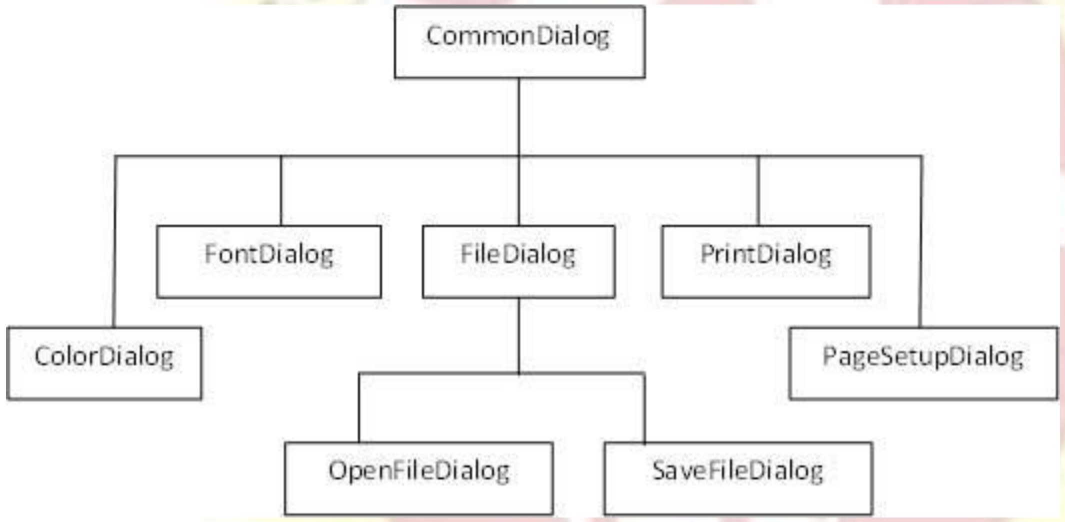
All of these dialog box control classes inherit from the **CommonDialog** class and override the *RunDialog()* function of the base class to create the specific dialog box.

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

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

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

The following diagram shows the common dialog class inheritance −



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

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

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

Sr.No.	Control & Description
1	ColorDialog It represents a common dialog box that displays available colors along with controls that enable the user to define custom colors.

2	FontDialog It prompts the user to choose a font from among those installed on the local computer and lets the user select the font, font size, and color.
3	OpenFileDialog It prompts the user to open a file and allows the user to select a file to open.
4	SaveFileDialog It prompts the user to select a location for saving a file and allows the user to specify the name of the file to save data.
5	PrintDialog It lets the user to print documents by selecting a printer and choosing which sections of the document to print from a Windows Forms application.

Microsoft windows common control

To add this control or any external control for VB project and for further understand follow the
steps:

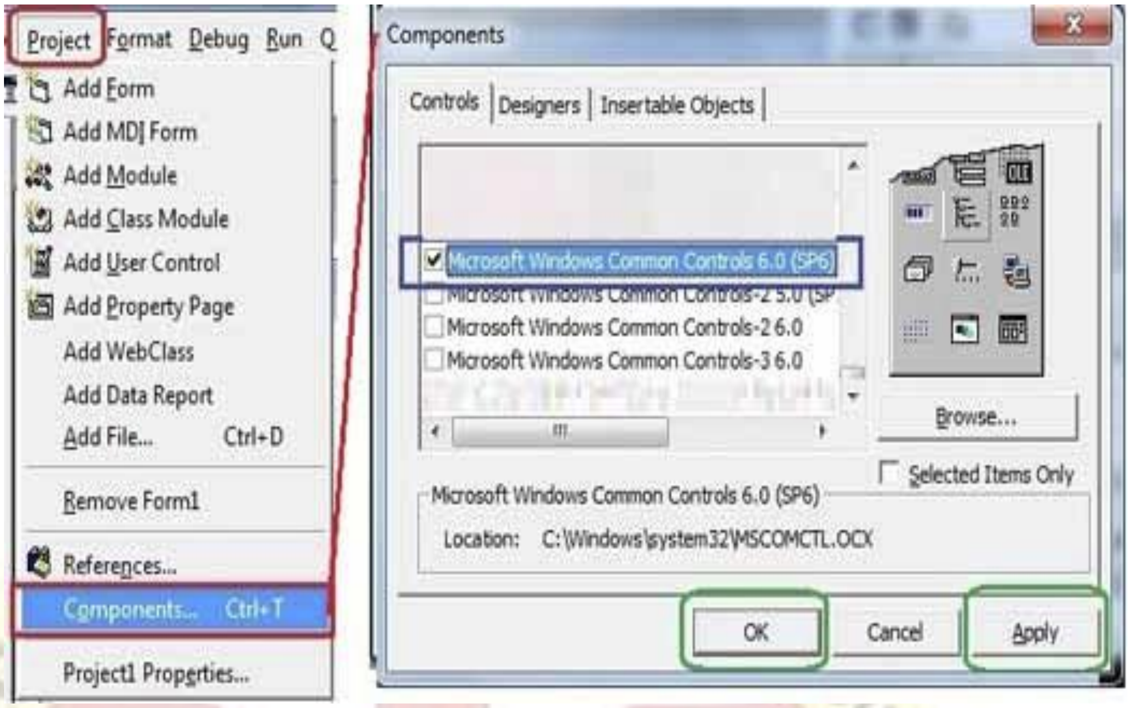
Steps: Project (Menu) >> Components (Window) >> Controls (Tab) >> Chose Component

First, find and go to "Project" menu and click on it.

Find "Components" option from "Project" menu and click on it. "Components" window will appear. You can also press (Ctrl + T) from keyboard to open this window.

This window contain 3 Tabs (Controls, Designers, Insertable Objects). Find "Microsoft Windows Common Control 6.0(SP6)" from "Controls" tab and check it.

Then click on "OK" or "Apply" butto



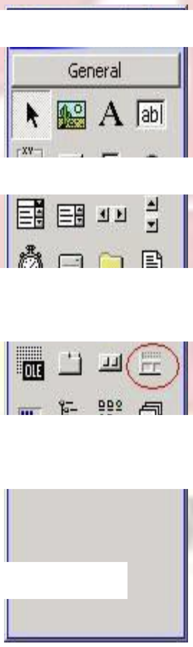


IMAGE LIST CONTROL IN VB6.0

Image List control:

- An ImageList Control contains a collection of images that can be used by other Windows Common Control
- It does not appeare on the form at run time.

It serve as a container for icon that are accessed by other control such as ListView,TreeView,TabStrip and ToolBar controls.

To add Image list control in the tool box:
1.Click on "Start" . Then select "All Programs, Microsoft Visual Studio 6.0 and Microsoft Visual Basic 6.0." .

2. Select "Standard EXE" from the list in the New Project dialog box and click on "Open." Click on "Project" on the menu bar, then select "Components" from the drop-down menu.

3. Scroll down the list in the box until Microsoft Windows Common Controls 6.0 (SP4) is visible. Click on the checkbox to select the component and then click on "OK." All components show in the Toolbox.

To add Image List control in the form:

- Choose the "ListView1" control from the list of controls in the toolbox. Place the control on the form in Visual Basic.

To add images to the image list control:

- Right click on Image control a popup menu is open choose properties option.A Properties Page is open .
- Click on Images Tab and insert the image as you want.
- To insert more than one picture then click on Insert Picture Button

LISTVIEW CONTROL

Text: Input of header name
Alignment: Set Alignment (Left, Center, Right)
Width: Increase or decrease width by giving width Value. Key: Type header key value.
Tag: Tag header by giving value.
Icon Index: Define icon index number.

PROGRESS BAR

First we need to find out about Controls, Properties and use. For including this control in VB6.0 Form, you need to find component with given steps:

Project (menu)>> component>> Controls(Tab)>> Microsoft Windows Common Controls 6.0 (SP6)
ProgressBar Scrolling (Bar Style) Options
0 - ccScrollingStandard
1 - ccScrollingSmooth
Style (Bar style Options)
Blocks

Continuous
Marquee

StatusBar

A StatusBar control is a frame that can consist of several panels which inform the user of the status of an application. The StatusBar can be divided up into a maximum of sixteen Panel objects that are contained in a Panels collection. Various kinds of status data can be displayed in a panel: text, keyboard states such as Caps Lock, Num Lock, Scroll Lock, and Kana (on Japanese systems only), the current date, and the current time. The only type of panel that can be modifed at run-time is a Text panel. The other types update themselves automatically without you writing any code. Additionally, the control has a "simple" style (set with the Style property)

The Index property is automatically maintained as you insert or remove panels.

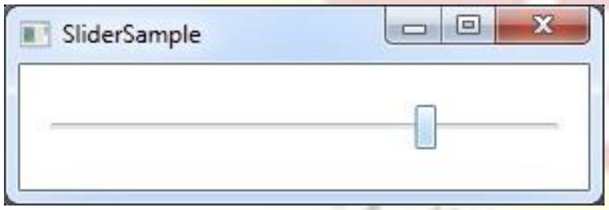
With the Text property (applicable only when the Style of the panel is set to 0 - sbrText), specify any initial text you want the panel to display.

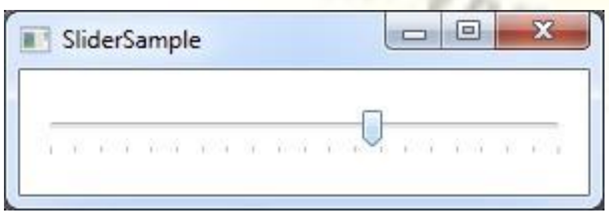
The Key property is a string that uniquely identifies the panel (as with all collections, a member can be identified either by its numeric Index property or its string Key property).

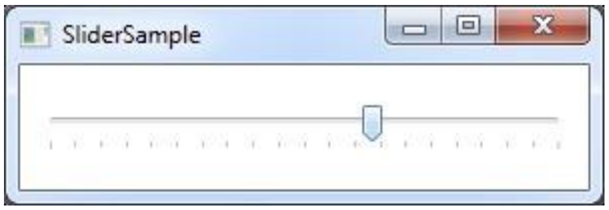
The Alignment property lets you specify whether you want the text justified left, center, or right within the panel.

The Slider control

The Slider control allows you to pick a numeric value by dragging a thumb along a horizontal or vertical line. You see it in a lot of user interfaces, but it can still be a bit hard to recognize from the description alone, so here's a very basic example:

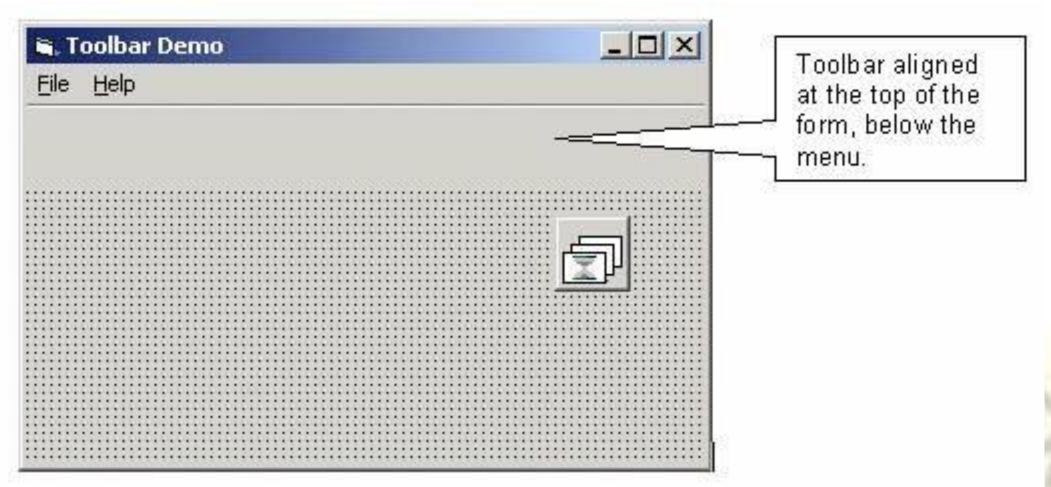


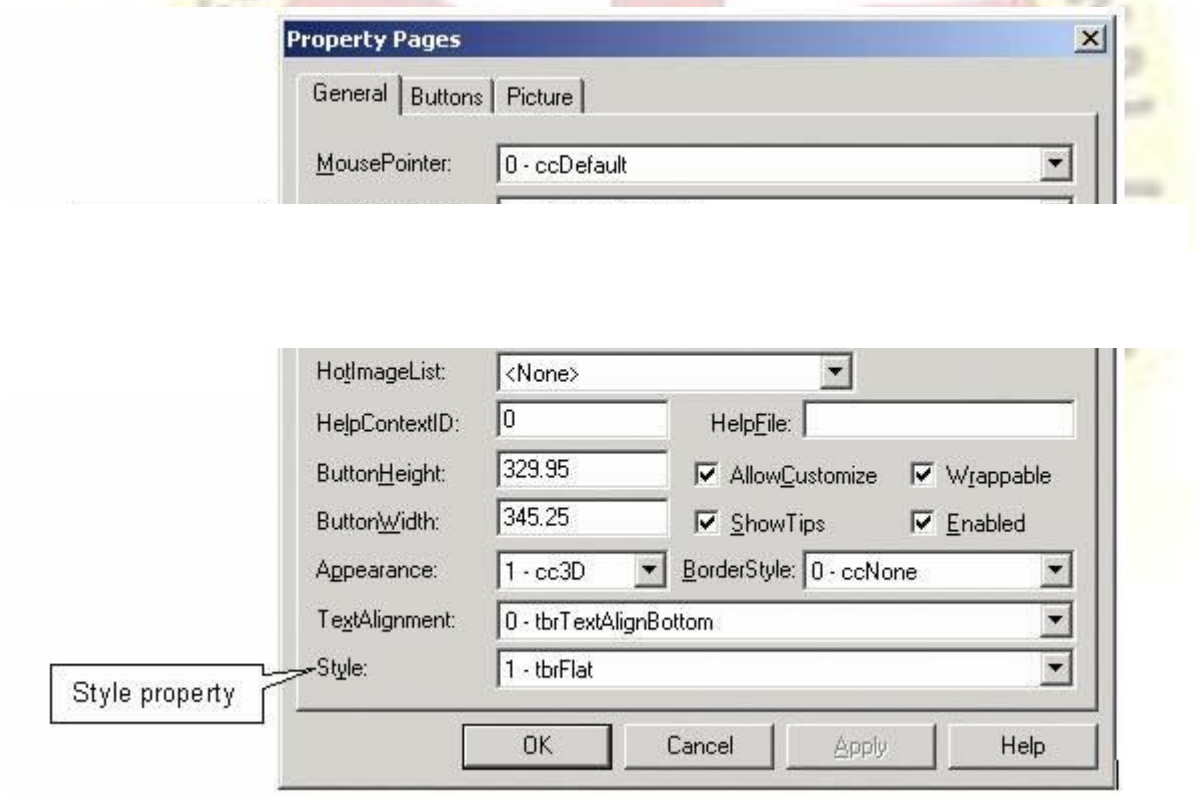




You can use a TabStrip to view different sets of information for related controls.

For example, the controls might represent information about a daily schedule for a group of individuals, with each set of information corresponding to a different individual in the group.





MousePointer: 0 - ccDefault

HotImageList: <None>

HelpContextID: 0

HelpFile:

ButtonHeight: 329.95

ButtonWidth: 345.25

AllowCustomize ✓

Wrappable ✓

ShowTips ✓

Enabled ✓

Appearance: 1 - cc3D

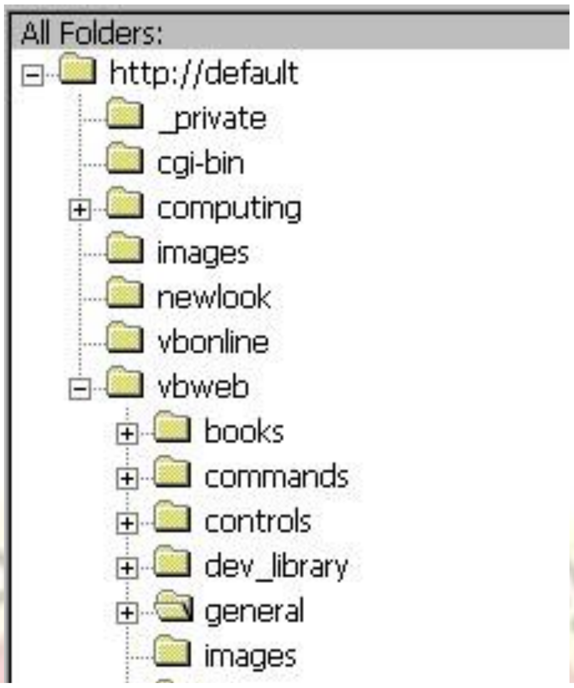
BorderStyle: 0 - ccNone

TextAlignment: 0 - tbrTextAlignBottom

Style: 1 - tbrFlat

TreeView Control

The Tree View control is a Visual Basic version of the control you see used in many programs, including Explorer and FrontPage, used to list the folders on your hard disk. This control allows you to add nodes to a tree, each of which can have sub items. Below is an image of the Tree View control in use.



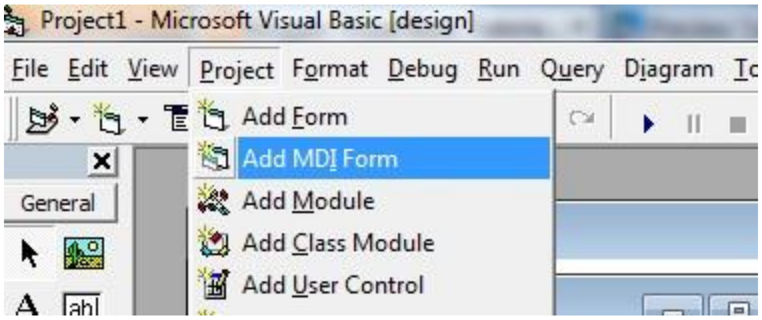
The Tree View control comes with all editions of Visual Basic, except the standard version.

MDI form

What is an MDI form?

MDI stands for Multiple Document Interface. You have probably seen many MDI applications. When you want to handle multiple documents, MDI forms are useful in a Windows program.

How to add an MDI form to the current project?


Project1 - Microsoft Visual Basic [design]
File Edit View Project Format Debug Run Query Diagram
Add Form
Add MDI Form
Add Module
Add Class Module
Add User Control
General

Project -> Add MDI form. Click Project from the menu bar, and click Add MDI form. It's simple! Remember, a project can have only one MDI form.

Restrictions of the MDI form

1. You can have only one MDI form per project.
2. You can't place most controls on an MDI form. The only controls that can be placed on the surface of the MDI form are Menus, Timer, CommonDialog, PictureBox, ToolBar, and StatusBar.

These restrictions are there because MDI forms are the special type of forms, especially used to handle multiple child forms.

How does the MDI form work?

There can be only one MDI parent form in a project with one or more MDI child forms (or simply child forms).

- **MDI child form:** To add a child form, you have to add a regular form, and set the MDIchild property to True. You can have many child forms and can show an MDI child form using the Show method.
- **AutoShowChildren property of an MDI form:** The default value of the AutoShowChildren property is True. When it is True, the MDI child forms are displayed once they are loaded. When the value is False only then you can keep it hidden after loading, otherwise not.
- **Restrictions of the MDI child forms:**
  1. You can't display an MDI child form outside its parent.
  2. You can't display a menu bar on the MDI child form.

Now coming to the point - how the MDI form works. The parent form contains a menu bar on top of it. From there, the user opens or creates a new document. In this way, the user accomplishes his/her work in one or multiple documents, then saves and closes the document (form). You can create instances of a single form in the code using the Set keyword (Using the object variables).

```
'Inside the MDIForm module
Private Sub mnuFileNew_Click()
   Dim frm As New Form1
   frm.Show
End Sub
```

- **ActiveForm property:** This is the Object type read-only property of the MDI form. You can apply this property to one of the children. For example, you can close the active form using this property from the Close menu command of the menu bar.

```
'In the MDI form
Private Sub mnuFileClose_Click()
   If Not (ActiveForm Is Nothing) Then Unload ActiveForm
End Sub
```

Parent and Child Menus

MDI Form cannot contain objects other than child Forms, but MDI Forms can have their own menus. However, because most of the operations of the application have meaning only if there is at least one child Form open, there's a peculiarity about the MDI Forms. The MDI Form usually has a menu with two commands to load a new child Form and to quit the application. The child Form can have any number of commands in its menu, according to the application. When the child Form is loaded, the child Form's menu replaces the original menu on the MDI Form

Following example illustrates the above explanation.

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

* Design a menu that has the following structure.

<> MDIMenu Menu caption

- MDIOpen opens a new child Form
- MDIExit terminates the application

* Then design the following menu for the child Form

<> ChildMenu Menu caption

- Child Open opens a new child Form
- Child Save saves the document in the active child Form
- Child Close Closes the active child Form

Debugging And Testing

In w0riting VB software, three types of errors can occur:

- Syntax errors (Compile errors)
- Run-time errors
- Logic errors

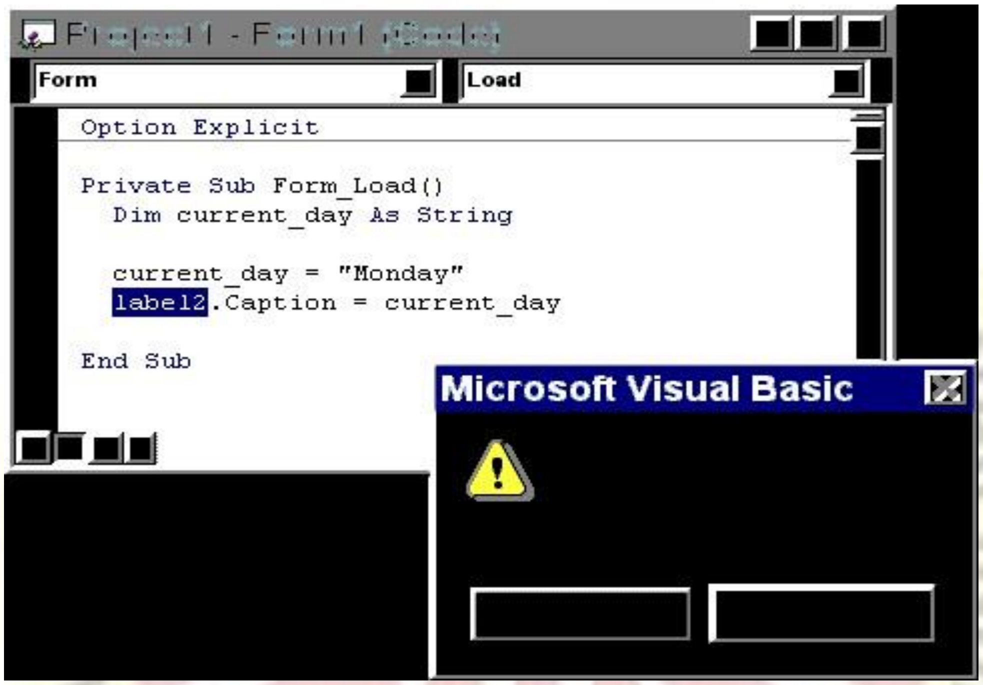
Syntax errors

These are grammatical errors in the formulation of statements and are picked up by the interpreter while you are typing in the code (providing the syntax checking option - under environment options - is set to yes). In the example below the syntax of the assignment statement is incorrect - the property of Label1 is missing (the statement should be Label1.Caption = current_day). Visual Basic has signalled this error at design time by placing the cursor at the point of omission and displaying a pop up error message.

Run-time errors

These are errors that cannot be detected until the program is running. The syntax of the statements is correct, but on execution they cause a situation to arise that results in a crash or an undefined value. Error handlers can be used to trap such errors and deal with them (these are beyond the scope of this course).

Examples of run-time errors are attempted division by zero or trying to access a non-existent object as in the example below (where Label2 has not been created).



```
Option Explicit

Private Sub Form_Load()
  Dim current_day As String

  current_day = "Monday"
  label2.Caption = current_day

End Sub
```

Logic errors

These are errors that cause the program to behave incorrectly. They generally arise through failure on the part of the programmer to arrive at a correct algorithm for the task. Typical problems might be incorrect ordering of statements, failure to initialise or re-initialise a variable, assignment to an incorrect variable, use of ‘<’ instead of ‘<=’, use of ‘and’ instead of ‘or’, or omission of a crucial step in the processing. Logic errors may lurk in a program even when it appears to work - they may only surface under certain conditions. This is why careful testing is so important.

Using debugging tools

Debugging tools are designed to help with:

- Stopping the execution of a program at specific points
- Detecting run-time and logic errors

- Understanding the behaviour of error-free code

Program errors can be very tricky to isolate. Sometimes they occur at the end of a long series of calculations or after many iterations of a loop. Programs execute very quickly and there is generally no opportunity to verify exactly what has been executed unless you build in diagnostic statements (such as print statements added to produce a trail, or to display intermediate calculations).

Using Break Mode

At design time you can change the interface or code but the effect on the executing program cannot be seen until run time. At run time you can observe the behaviour of the program, but not change it.

Break mode halts execution of the software and gives a snapshot of its state at a particular moment. Variable and property settings are preserved and may be inspected. In break mode you can:

- Observe the state of the interface.
- Determine which procedures are active.
  Control which statement will be executed next.
- Watch the values of variables and properties.

Using the Debug window

Sometimes debugging requires analysis of what is happening to data. A problem may have been traced to the value of a variable or property but you need to know where the incorrect assignment occurred. The Debug window lets you display the value of variables either while a program is running or after it has been halted. The window consists of two panes:

1. Watch pane

Located directly beneath the Debug window title bar, this pane is visible only if you have selected a watch expression to be monitored during code execution. The displayed information includes the context of the watch expression, the expression itself and its value at the time of the transition to break mode. If the context of the expression is not in scope when going to break mode, the current value is not displayed.

2. Immediate pane

Located below the title bar, or below the Watch pane if it is displayed, the Immediate pane is where you enter code to execute it immediately. While working in the Immediate pane:

- You can execute only one line of code at a time.
- You can type or paste a line of code and press to run it.
- You cannot save code, but you can copy and paste it into the Code window.
- In Break mode, a statement in the Debug window is executed in the context displayed in the Debug window title bar. For example, if you type Print variable name your output is the value of a local variable. This is the same as if the Print method had actually occurred in the procedure you were executing when the program halted.

The Debug window can do different things depending on what mode Visual Basic is in:

Break Mode

When execution of a program has been temporarily halted and VB is in Break mode the Debug window can be used to execute single line commands entered by the user. For example, the user can click on the Debug window, type 'Print Time$', and hit . The result is the current system time is displayed in the Debug window. Almost any line of code can be executed from the Debug window. You can also use the Debug window to monitor the value of expressions you have selected as watch expressions.

Run time Mode

At run time, although you can't enter statements into the Debug window, you can use it to display internal values without printing them to the actual forms. The Debug window is an object which has a single Print method. For example, the following line will send the name of the font used in 'Label1' to the debug window at run time:

```
Debug.Print Label1.FontName
```

Variables or constants can also be printed in the same way:

```
Debug.Print intCurrentDay
Debug.Print "Reached here in the code"
```

Working with graphics

Working with graphics is easy in Visual Basic 6. VB6 gives you the flexibility and power to make graphical applications in easy steps. It is rich in graphics related features. The built-in methods, properties and events allow you to use these features most effectively.

In this tutorial, you will learn about various graphic methods, properties and techniques. The graphic methods and properties let you perform some graphical operations. More specifically, they allow you to draw points, lines, circles, rectangles, ellipses and other shapes. You will also learn how to display text and images. Finally, this tutorial introduces you to the Paint event.

This tutorial is for the beginner learners. So the concepts are presented in the simplest way possible. Code examples have been used wherever they became necessary.

Graphic methods

The graphic methods allow you to draw on the form and the PictureBox control. In Visual Basic 6, graphic methods are only supported by the form object and the PictureBox control. However, the later versions of Visual Basic allow you to use them with other objects and controls.

First of all, the graphic methods in this section are discussed only to give you a basic idea about them. Later in this tutorial, I will show you how to use them in your code to perform certain operations like printing text, drawing shapes etc.

The common graphic methods are explained below.

- Print: Print is the simplest graphic method in Visual Basic 6. This method has been used throughout the earlier versions of the language. It prints some text on the form or on the PictureBox control. It displays texts.
- Cls: The Cls method is another simple graphic method that is used to clear the surface of the form or the PictureBox control. If some texts are present, you can use the Cls method to remove the texts. It clears any drawing created by the graphic methods.
- Point: The Point method returns the color value from an image for a pixel at a particular point. This method is generally used to retrieve color values from bitmaps.
- Refresh: The refresh method redraws a control or object. In other words, it refreshes the control. Generally, controls are refreshed automatically most of the times. But in some cases, you need to refresh a control's appearance manually by explicitly invoking the Refresh method.

- PSet: The PSet method sets the color of a single pixel on the form. This method is used to draw points.
- Line: The Line method draws a line. Using the Line method, you can also draw other geometric shapes such as rectangle, triangle etc.
- Circle: The Circle method draws a circle. Using the Circle method, you can also draw other geometric shapes such ellipses, arcs etc.
- PaintPicture: The PaintPicture method displays an image on the form at run-time.
- TextHeight: The TextHeight method returns the height of a string on the form at run-time.
- TextWidth: The TextWidth method returns the width of a string on the form at run-time.

The LoadPicture function

The LoadPicture function loads a picture to the form or to the PictureBox control. It sets the picture to the control in order to display it. The function takes the file path as an argument. The LoadPicture function allows you to set pictures at run-time.

Example:

Code:

```
Picture1 = LoadPicture("C:\MyPic.JPG")
```

Here, Picture1 is the PictureBox control. When this code is executed the picture is loaded and set to the PictureBox control. If Visual Basic cannot find the picture in the specified location, it throws a run-time error '53'.

The RGB function

The RGB function returns an integer, a color code which is used to set colors in Visual Basic code. The RGB color code is a combination of red, green and blue colors. Consider the following example to understand RGB function in VB6.

Example:

Code:

```
Form1.BackColor = RGB (120, 87, 55)
```

The RGB color is set as the background color of the form object. The first, second and the third arguments represent red, green and blue colors respectively. The color value is an integer. These values are in a range of 0 to 255. So you can use any value between 0 and 255 to obtain a color.

Graphic properties

The graphic properties are useful while working with the graphic methods. Some of the form's properties and some of the PictureBox's properties are the graphics properties.
The common graphic properties are discussed in this section. You'll learn more about them using code examples later in this tutorial.

Consider the following graphic properties.

- DrawMode: The DrawMode property sets the mode of drawing for the appearance of output from the graphic methods. In the DrawMode property, you can choose from a variety of values.
- DrawStyle: The DrawStyle property sets the line style of any drawing from any graphic methods. It allows you to draw shapes of different line styles such as solid, dotted, dashed shapes etc.
- DrawWidth: The DrawWidth property sets the line width of any drawing from any graphic methods. While drawing shapes, you can control the thickness of the lines using this property.
- FillColor: The FillColor property is used to fill any shapes with a color. You may use the symbolic color constants to fill your shapes. You may also use the color codes as well as the RGB function.
- FillStyle: The FillStyle property lets you fill shapes in a particular filling style.
- ForeColor: The ForeColor property is used to set or return the foreground color.
- AutoRedraw: Set the AutoRedraw property to True to get a persistent graphics when you're calling the graphic methods from any event, but not from the Paint event.
- ClipControls: Set the ClipControls property to True to make the graphic methods repaint an object.
- Picture: The Picture property is used to set a picture. Pictures can be set both at design time and run-time.

Run-time graphic properties

CurrentX and CurrentY are the run-time properties which are used to set and return the position of a shape or point at run-time.

- CurrentX: The CurrentX property sets or returns the horizontal coordinate or X-coordinate of the current graphic position at run-time.
- CurrentY: The CurrentY property sets or returns the vertical coordinate or Y-coordinate of the current graphic position at run-time.

Unit 5

Monitoring Mouse activity

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

File Handling

A **file** is a collection of data stored in a disk with a specific name and a directory path. When a file is opened for reading or writing, it becomes a **stream**.

The stream is basically the sequence of bytes passing through the communication path. There are two main streams: the **input stream** and the **output stream**. The **input stream** is used for reading data from file (read operation) and the **output stream** is used for writing into the file (write operation).

VB I/O Classes

The System.IO namespace has various classes that are used for performing various operations with files, like creating and deleting files, reading from or writing to a file, closing a file, etc.

The following table shows some commonly used non-abstract classes in the System.IO namespace −

I/O Class	Description
BinaryReader	Reads primitive data from a binary stream.
BinaryWriter	Writes primitive data in binary format.
BufferedStream	A temporary storage for a stream of bytes.
Directory	Helps in manipulating a directory structure.
DirectoryInfo	Used for performing operations on directories.
DriveInfo	Provides information for the drives.
File	Helps in manipulating files.
FileInfo	Used for performing operations on files.
FileStream	Used to read from and write to any location in a file.
MemoryStream	Used for random access of streamed data stored in memory.
Path	Performs operations on path information.
StreamReader	Used for reading characters from a byte stream.

StreamWriter	Is used for writing characters to a stream.
StringReader	Is used for reading from a string buffer.
StringWriter	Is used for writing into a string buffer.

The FileStream Class

The **FileStream** class in the System.IO namespace helps in reading from, writing to and closing files. This class derives from the abstract class Stream.

You need to create a **FileStream** object to create a new file or open an existing file. The syntax for creating a **FileStream** object is as follows −

Dim <object_name> As FileStream = New FileStream(<file_name>, <FileMode Enumerator>, <FileAccess Enumerator>, <FileShare Enumerator>)

For example, for creating a FileStream object **F** for reading a file named **sample.txt** −

Dim f1 As FileStream = New FileStream("sample.txt", FileMode.OpenOrCreate, FileAccess.ReadWrite)

Parameter	Description
FileMode	The **FileMode** enumerator defines various methods for opening files. The members of the FileMode enumerator are − • **Append** − It opens an existing file and puts cursor at the end of file, or creates the file, if the file does not exist. • **Create** − It creates a new file. • **CreateNew** − It specifies to the operating system that it should create a new file. • **Open** − It opens an existing file. • **OpenOrCreate** − It specifies to the operating system that it should open a file if it exists, otherwise it should create a new file. • **Truncate** − It opens an existing file and truncates its size to zero bytes.
FileAccess	**FileAccess** enumerators have members: **Read**, **ReadWrite** and **Write**.

FileShare	**FileShare** enumerators have the following members − • **Inheritable** − It allows a file handle to pass inheritance to the child processes • **None** − It declines sharing of the current file • **Read** − It allows opening the file for reading • **ReadWrite** − It allows opening the file for reading and writing • **Write** − It allows opening the file for writing

Example

The following program demonstrates use of the **FileStream** class −

```
Imports System.IO
Module fileProg
  Sub Main()
    Dim f1 As FileStream = New FileStream("sample.txt", _ FileMode.OpenOrCreate, FileAccess.ReadWrite)
    Dim i As Integer

    For i = 0 To 20
      f1.WriteByte(CByte(i))
    Next i
    f1.Position = 0

    For i = 0 To 20
      Console.Write("{0} ", f1.ReadByte())
    Next i
    f1.Close()
    Console.ReadKey()
  End Sub
End Module
```

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

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 -1
```

File Controls

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 DriveListBox control is a combobox-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

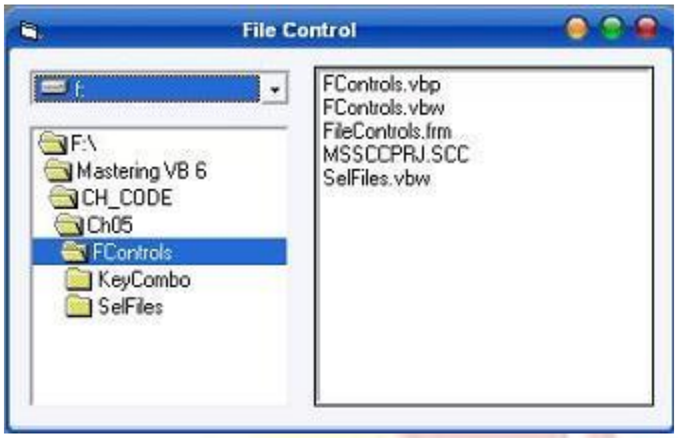
ListBox control that displays all the files in a given directory, optionally filtering them based on their names, extensions, and attributes.

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.
- **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.
- **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.

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.

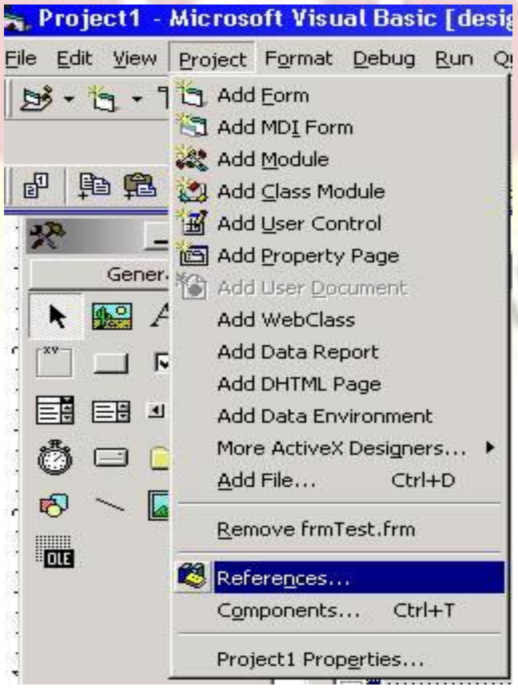
To connect to the File controls, you must assign the appropriate values to the properties. To compel the DirListBox to display the folders of the selected drive in the DriveListBox, you must make sure that each time the user selects another drive, the Path property of the DirListBox control matches the Drive property of the DriveListBox.

File System Objects

The File System Object (FSO) object model provides an object-based tool for working with folders and files. Using "object.method" syntax, it exposes a comprehensive set of properties and methods to perform file system operations such as creating, moving, deleting, and providing information about folders and files. The FSO also provides methods for reading and writing sequential text files, however it does NOT have methods for processing binary or random files.

The FSO is (or should be) used primarily with VBScript. VBScript is a scripting language used with ASP for web development; VBScript is also used for Windows scripting. (Windows scripting files, which have a ".vbs" extension, can be thought of as a modern-day replacement for the MS-DOS "BAT" files used in the days of yore). VBScript is a pared-down version of Visual Basic, and as such does not have all of the functionality of "VB proper". One of the things missing in VBScript is the set of native VB file processing statements and functions discussed in the last several topics – so in VBScript, the FSO must be used to manipulate files and folders. However, VB proper can make use of the FSO in addition to its native file processing commands.

There are some trade-offs in using the FSO with Visual Basic. On the one hand, the FSO can make certain tasks easier to program with smoother and less arcane syntax than the native VB statements. On the other hand, using the FSO requires adding an additional dependancy to your project, it is slower, it does not support the reading or writing of random and binary files, and it can be disabled by system administrators concerned about security.



To use the FSO with your VB project, you must add a reference to "Microsoft Scripting Runtime" (which is the system file "SCRRUN.DLL"). To do this, from the VB IDE, first go to the **Project** menu, and select **References**, as shown below:

From the References dialog box, check **Microsoft Scripting Runtime**, as shown below, and click **OK**.
The tables below show the various objects, properties, and methods available with the FSO.

FSO Objects

Object	***Description***
FileSystemObject	The FSO itself, highest level of the FSO object model. Allows the programmer to interact with **Files**, **Folders** and **Drives**. The programmer can use the FSO objects to create directories, move files, determine whether or not a **Drive** exists, etc.
Drive	The **Drive** object is used to examine information on disk, CD-ROM, RAM disk, and network drives; the **Drives** collection provides a list of physical and logical drives on the system.
File object	The **File** object is used to examine and manipulate files; the **Files** collection provides a list of files in a folder.
Folder object	The **Folder** object is used to examine and manipulate folders; the **Folders** collection provides a list of subfolders in a folder.
TextStream object	Used to read and write text files.

Property **of the FileSystemObject**

Property	***Description***
Drives	Returns a **Drives** collection, which is a list of physical and logical drives on the system.

Methods **of the FileSystemObject**

Method	***Description***
BuildPath	Appends file path information to an existing file path.
CopyFile	Copies files from one location to another.
CopyFolder	Copies folders and their contents from one location to another.
CreateFolder	Creates a folder.
CreateTextFile	Creates a text file and returns a TextStream object.
DeleteFile	Deletes a file.
DeleteFolder	Deletes a folder and all of its contents.
DriveExists	Determines if a drive exists.
FileExists	Determines if a file exists.
FolderExists	Determines if a folder exists.
GetAbsolutePathName	Returns the full path to a file or folder.
GetBaseName	Returns the base name of a file or folder.
GetDrive	Returns a drive object.
GetDriveName	Returns a drive name.
GetExtensionName	Returns a file extension from a path.
GetFile	Returns a file object.

GetFileName	Returns a filename from a path.
GetFolder	Returns a folder object.
GetParentFolderName	Returns the parent folder name from a path.
GetSpecialFolder	Returns an object pointer to a special folder.
GetTempName	Returns a temporary (randomly generated) file or folder name that can be used with CreateTextFile.
MoveFile	Moves files from one location to another.
MoveFolder	Moves folders and their contents from one location to another.
OpenTextFile	Opens an existing text file and returns a TextStream object.

***Properties* of the Drive Object**

Property	***Description***
AvailableSpace	The amount of available **Drive** space in bytes.
DriveLetter	A string containing the letter of the **Drive** without a colon (e.g., "C").
DriveType	An integer value indicating the **Drive** type. Possible values are 0 (Unknown), 1 (Removable), 2 (Fixed), 3 (Remote), 4 (CD-ROM) and 5 (RAM Disk).
FileSystem	A string indicating the file system **Drive** description ("FAT", "FAT32", "NTFS", etc.).
FreeSpace	Same as **AvailableSpace** .
IsReady	A Boolean indicating whether or not a **Drive** is ready for use.
Path	A string containing the **Drive**'s path (e.g., "C:\")
RootFolder	A **Folder** object containing the root folder of **Drive**.
SerialNumber	A long value containing the **Drive** serial number.
ShareName	With network drives, returns a string containing the network share name.
TotalSize	The total **Drive** size in bytes.
VolumeName	A string value containing the **Drive** volume name.

***Properties* of the Folder Object**

Property	***Description***
Attributes	An integer value indicating **Folder**'s attributes. Possible values are 0 (Normal), 1 (ReadOnly), 3 (Hidden), 4 (System), 8 (Volume), 16 (Directory), 32 (Archive), 64 (Alias), and 128 (Compressed).
DateCreated	The date the folder was created.
DateLastAccessed	The date the folder was last accessed.
DateLastModified	The date the folder was last modified.
Drive	The **Drive** where the folder is located.
IsRootFolder	A Boolean indicating whether or not a **Folder** is the root folder.
Name	A string containing the **Folder**'s name.
ParentFolder	A string containing the **Folder**'s parent folder name.
Path	A string containing the **Folder**'s path.
ShortName	A string containing the **Folder**'s name expressed as an MS-DOS compliant ("8.3") short name.
ShortPath	A string containing the **Folder**'s path expressed as a short (MS-DOS compliant) path.
Size	The total size in bytes of all subfolders and files.
Type	A string containing the **Folder** type (e.g., "File Folder").

***Methods* of the Folder Object**

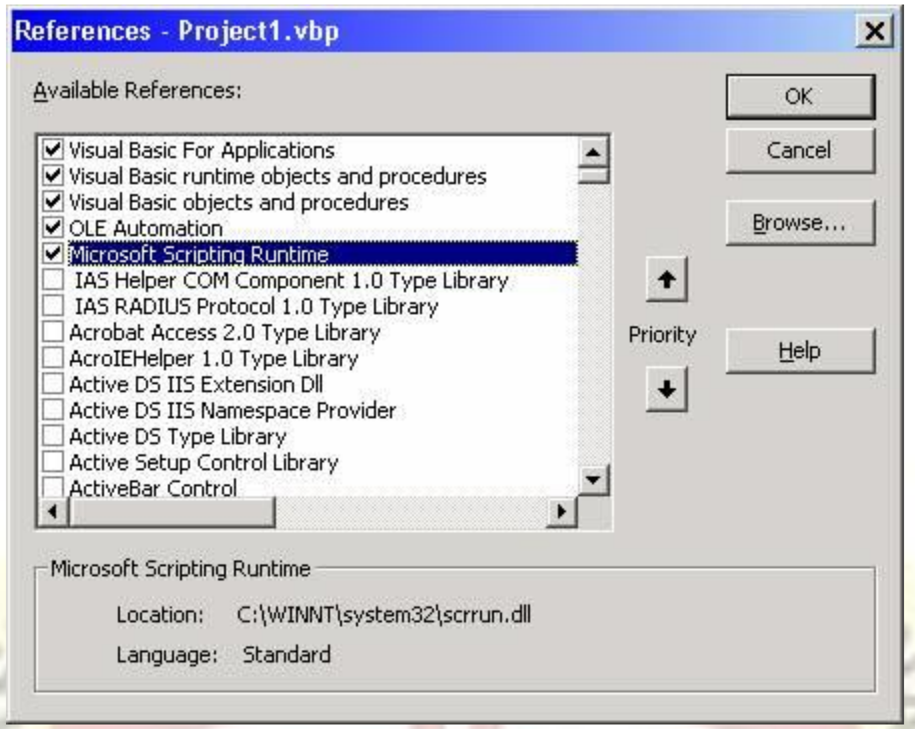
Method	*Description*
Delete	Deletes the **Folder**. Same as **DeleteFolder** of **FileSystemObject**.
Move	Moves the **Folder**. Same as **MoveFolder** of **FileSystemObject**.
Copy	Copies the **Folder**. Same as **CopyFolder** of **FileSystemObject**.

***Properties* of the File Object**

Property	*Description*
Attributes	An integer value indicating **File**'s attributes. Possible values are 0 (Normal), 1 (ReadOnly), 3 (Hidden), 4 (System), 8 (Volume), 16 (Directory), 32 (Archive), 64 (Alias), and 128 (Compressed).
DateCreated	The date the file was created.
DateLastAccessed	The date the file was last accessed.
DateLastModified	The date the file was last modified.
Drive	The **Drive** where the file is located.
Name	A string containing the **File**'s name.
ParentFolder	The **Folder** object of the file's parent folder.
Path	A string containing the **File**'s path.
ShortName	A string containing the **File**'s name expressed as a short (MS-DOS compliant "8.3") name.
ShortPath	A string containing the **File**'s path expressed as a short (MS-DOS compliant) path.
Size	The total size in bytes of the file.
Type	A string containing the **File** type (e.g., "Microsoft Word Document").

***Methods* of the File Object**

Method	*Description*
Delete	Deletes the **File**. Same as **DeleteFile** of **FileSystemObject**.
Move	Moves the **File**. Same as **MoveFile** of **FileSystemObject**.
Copy	Copies the **File**. Same as **CopyFile** of **FileSystemObject**.
CreateTextFile	Returns a **TextStream** object that can be used to work with the newly created file.
OpenAsTextStream	Opens an existing text file and returns a TextStream object.



☑ Visual Basic For Applications
☑ Visual Basic runtime objects and procedures
☑ Visual Basic objects and procedures
☑ OLE Automation
☑ Microsoft Scripting Runtime
☐ IAS Helper COM Component 1.0 Type Library
☐ IAS RADIUS Protocol 1.0 Type Library
☐ Acrobat Access 2.0 Type Library
☐ AcroIEHelper 1.0 Type Library
☐ Active DS IIS Extension Dll
☐ Active DS IIS Namespace Provider
☐ Active DS Type Library
☐ Active Setup Control Library
☐ ActiveBar Control

Microsoft Scripting Runtime

Location: C:\WINNT\system32\scrrun.dll

Language: Standard

Once you have done the above, you can use the FSO in your VB project. In your code, you must declare an object variable for the FSO and instantiate it. The most concise way of doing this is use the "New" keyword in your declaration, as shown below. (Note: The "Scripting." qualifier is optional here.)

COM/OLE

OLE definition is that, OLE (pronounced as oh-leh) was originally mean as '*object linking and embedding*', it is a Microsoft compound document technology based on **Component Object Model** (**COM**), It is introduced into Microsoft Windows 3.1. '**OLE (object linking and embedding)**' gives all Windows applications a standard way to create compound documents that create objects within one microsoft application and embed them into other document. OLE object meaning is graphic,spreadsheet,msword, etc. that can be embedded into a document called the "**container application**." If the object allowed to be edited, the application associated with it is called "**server application**".

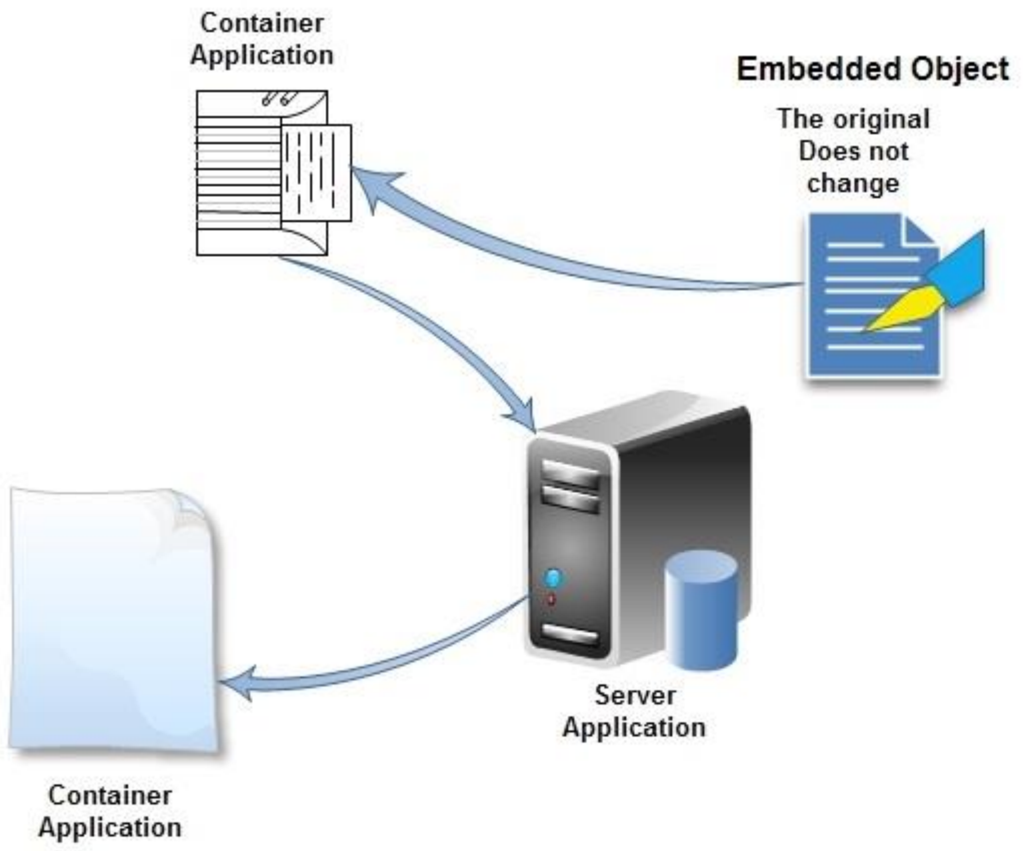
One of the key point is that OLE transformed software development from procedural programming languages to **object-oriented programming**.we can create self-contained Modules, or objects With the help of OLE, that simplify programming approach to building large applications. OLE Means that objects created from different formats that can be linked and embedding application data. The basic example of OLE is that, When we can insert Excel spreadsheet into a Word application.

We'll be covering the following topics in this tutorial:

- *OLE Embedding Meaning*
- *OLE Linking Meaning*

OLE Embedding Meaning

The Definition of Embedding meaning is that if One window application document contains a copy of other window application document,if Changes made that affect only the application document that contains it.



An object might be a passage of formatted text, a part of a spreadsheet, some sounds, or a picture. Unlike information that you copy from one document and paste into another the standard way, a linked or embedded object retains a connection to the application that originally created it. You can return to that application to edit the object whenever you want to just by double-clicking on the object-you don't have to bother with finding the icon for the application, loading the right file, and so on. Better yet, the changes you make automatically appear in the document where you linked or embedded the object.

When you embed an object, you place a copy of the information into your document. This copy is connected to the original application, but not to a particular document in that application. The only advantage to embedding an object instead of copying the information the ordinary way is that you can edit the object more conveniently.

OLE only works if both applications involved have been designed to use it, and even then it may only work in one direction (like, you can link a graphic into a text document, but not text into a graphic document). And it doesn’t work exactly the same way in every application. Even so, it’s easier and more consistent than the old method, called DDE.

OLE Linking Meaning

Linking means that the container application that contains a pointer to the original file,when the linked object is changed, the original document that link is automatically updated.

OLE Automation

Some applications provide objects that support OLE Automation. You can use Visual Basic to manipulate the data in these objects through programming. Some objects that support OLE Automation also support linking and embedding. If an object in an OLE container control supports OLE Automation, you can access its properties and methods using the Object property. If you draw the object directly on a form or create it in - code, you can directly access the properties and methods of the object.\

Dynamic Link Libraries (DLL)

A DLL file is often given a ".dll" file name suffix. DLL files are dynamically linked with the program that uses them during program execution rather than being compiled into the main program.

The advantage of DLL files is space is saved in random access memory (RAM) because the files don't get loaded into RAM together with the main program. When a DLL file is needed, it is loaded and run. For example, as long as a user is editing a document in Microsoft Word, the printer DLL file does not need to be loaded into RAM. If the user decides to print the document, the Word application causes the printer DLL file to be loaded and run.

A program is separated into modules when using a DLL. With modularized components, a program can be sold by module, have faster load times and be updated without altering other parts of the program. DLLs help operating systems and programs run faster, use memory efficiently and take up less disk space.

DLL advantages

The following list describes some of the advantages that are provided when a program uses a DLL:

- Uses fewer resources

When multiple programs use the same library of functions, a DLL can reduce the duplication of code that is loaded on the disk and in physical memory. It can greatly influence the performance of not just the program that is running in the foreground, but also other programs that are running on the Windows operating system.

- Promotes modular architecture

A DLL helps promote developing modular programs. It helps you develop large programs that require multiple language versions or a program that requires modular architecture. An example of a modular program is an accounting program that has many modules that can be dynamically loaded at run time.

- Eases deployment and installation

When a function within a DLL needs an update or a fix, the deployment and installation of the DLL does not require the program to be relinked with the DLL. Additionally, if multiple programs use the same DLL, the multiple programs will all benefit from the update or the fix. This issue may more frequently occur when you use a third-party DLL that is regularly updated or fixed.

OLE Drag and Drop

The drag-and-drop feature of OLE is primarily a shortcut for copying and pasting data. When you use the Clipboard to copy or paste data, a number of steps are required. You select the data, and choose **Cut** or **Copy** from the **Edit** menu. Then you move to the destination app or window, and place the cursor in the target location. Finally, you choose **Edit** > **Paste** from the menu.

The OLE drag-and-drop feature is different from the File Manager drag-and-drop mechanism. The File Manager can only handle filenames, and is designed specifically to pass filenames to applications. Drag and drop in OLE is much more general. It allows you to drag and drop any data that could also be placed on the Clipboard.

When you use OLE drag and drop, you remove two steps from the process. You select the data from the source window (the "drop source"), then you drag it to the destination (the "drop target"). You drop it by releasing the mouse button. The operation eliminates the need for menus, and it's quicker than the copy/paste sequence. There's only one requirement: Both the drop source and drop target must be open, and at least partially visible on the screen.

Using OLE drag and drop, data can be transferred easily from one location to another: Within a document, between different documents, or between applications. It may be implemented in either a container or a server application. Any application could be a drop source, a drop target, or both. If an application implements both drop-source and drop-target support, you can drag and drop between child windows, or within one window. This feature makes your application much easier to use.

The Data objects and data sources (OLE) articles explain how to implement data transfer in your applications.

Implement a drop target

It takes slightly more work to implement a drop target than a drop source, but it's still relatively simple.

To implement an OLE drop target

1. If it isn't already there, add a call to AfxOleInit in your application's InitInstance member function. This call is required to initialize the OLE libraries.
2. Add a member variable to each view in the application that you want to be a drop target. This member variable must be of type COleDropTarget or a class derived from it.
3. From your view class's function that handles the **WM_CREATE** message (typically OnCreate), call the new member variable's Register member function. Revoke will be called automatically for you when your view is destroyed.
4. Override the following functions. If you want the same behavior throughout your application, override these functions in your view class. If you want to modify behavior in isolated cases or want to enable dropping on non-CView windows, override these functions in your COleDropTarget-derived class.

TABLE 1

verride	**o allow**
nDragEnter	rop operations to occur in the window. Called when the cursor first enters the window.
nDragLeave	pecial behavior when the drag operation leaves the specified window.
nDragOver	rop operations to occur in the window. Called when the cursor is being dragged across the indow.
nDrop	andling of data being dropped into the specified window.
nScrollBy	pecial behavior for when scrolling is necessary in the target window.

Customize drag and drop

The default implementation of the drag-and-drop feature is sufficient for most applications. However, some applications may require you to change this standard behavior. This section explains the steps necessary to change these defaults. You can use this technique to make applications that don't support compound documents into drop sources.

If you're customizing standard OLE drag-and-drop behavior, or you have a non-OLE application, you must create a COleDataSource object to contain the data. When the user starts a drag-and-drop operation, your code should call the DoDragDrop function from this object instead of from other classes that support drag-and-drop operations.

You can override the following functions to customize drag-and-drop operations:

verride	**o customize**
nBeginDrag	ow the drag operation begins after you call DoDragDrop.
iveFeedback	isual feedback, such as cursor appearance, for different drop esults.
ueryContinueDrag	he termination of a drag-and-drop operation. This function nables you to check modifier key states during the drag peration.

*****